MenpoFit Documentation

Release 0.7.0

Joan Alabort-i-Medina, Epameinondas Antonakos, James Booth,

CONTENTS

1	User	Guide	3
	1.1	Quick Start	3
	1.2	Introduction	
	1.3	Building Models	4
	1.4	Fitting Models	6
		Fitting Result	
		Visualizing Objects	
	1.7	References	Ģ
2		MenpoFit API	13
		Deformable Models	
	2.2	Internal API	287
In	dex		455

Welcome to the MenpoFit documentation!

MenpoFit is a Python package for building, fitting and manipulating deformable models. It includes state-of-the-art deformable modelling techniques implemented on top of the **Menpo** project. Currently, the techniques that have been implemented include:

- Active Appearance Model (AAM)
 - Holistic, Patch-based, Masked, Linear, Linear Masked
 - Lucas-Kanade Optimisation
 - Cascaded-Regression Optimisation
- Active Pictorial Structures (APS)
 - Weighted Gauss-Newton Optimisation with fixed Jacobian and Hessian
- Active Template Model (ATM)
 - Holistic, Patch-based, Masked, Linear, Linear Masked
 - Lucas-Kanade Optimisation
- Lucas-Kanade Image Alignment (LK)
 - Forward Additive, Forward Compositional, Inverse Compositional
 - Residuals: SSD, Fourier SSD, ECC, Gradient Correlation, Gradient Images
- Unified Active Appearance Model and Constrained Local Model (Unified AAM-CLM)
 - Alternating/Project Out with Regularised Landmark Mean Shift
- Constrained Local Model (CLM)
 - Active Shape Model
 - Regularised Landmark Mean Shift
- Ensemble of Regression Trees (ERT) [provided by DLib]
- Supervised Descent Method (SDM)
 - Non Parametric
 - Parametric Shape
 - Parametric Appearance
 - Fully Parametric

Please see the to *References* for an indicative list of papers that are relevant to the methods implemented in MenpoFit.

CONTENTS 1

2 CONTENTS

CHAPTER

ONE

USER GUIDE

The User Guide is designed to give you an overview of the key concepts within MenpoFit. In particular, we want to try and explain some of the design decisions that we made and demonstrate why we think they are powerful concepts for building, fitting and analysing deformable models.

1.1 Quick Start

Here we give a very quick rundown of the basic links and information sources for the project.

1.1.1 Basic Installation

In the Menpo Team, we **strongly** advocate the usage of conda for scientific Python, as it makes installation of compiled binaries much more simple. In particular, if you wish to use any of the related Menpo projects such as *menpofit*, *menpo3d* or *menpodetect*, you will not be able to easily do so without using conda. The installation of MenpoFit using conda is as easy as

```
$ conda install -c menpo menpofit
```

Conda is able to work out all the requirements/dependencies of MenpoFit. You may for example notice that *menpo* is one of them. Please see the thorough installation instructions for each platform on the Menpo website.

1.1.2 API Documentation

Visit API Documentation

MenpoFit is extensively documented on a per-method/class level and much of this documentation is reflected in the API Documentation. If any functions or classes are missing, please bring it to the attention of the developers on Github.

1.1.3 Notebooks

Explore the Menpo and MenpoFit Notebooks

For a more thorough set of examples, we provide a set of Jupyter notebooks that demonstrate common use cases of MenpoFit. The notebooks include extensive examples regarding all the state-of-the-art deformable models that we provide. You may need to have a look at the Menpo notebooks in order to get an overview of the basic functionalities required by MenpoFit.

1.1.4 User Group and Issues

If you wish to get in contact with the Menpo developers, you can do so via various channels. If you have found a bug, or if any part of MenpoFit behaves in a way you do not expect, please raise an issue on Github.

If you want to ask a theoretical question, or are having problems installing or setting up MenpoFit, please visit the user group.

1.2 Introduction

This user guide is a general introduction to MenpoFit, aiming to provide a bird's eye of MenpoFit's design. After reading this guide you should be able to go explore MenpoFit's extensive Notebooks and not be too surprised by what you see.

1.2.1 What makes MenpoFit better?

The vast majority of **existing deformable modeling software** suffers from one or more of the following important issues:

- · It is released in binary closed-source format
- It does not come with training code; only pre-trained models
- It is not well-structured which makes it very difficult to tweak and alter
- It only focuses on a single method/model

MenpoFit overcomes the above issues by providing open-source *training* and *fitting* code for multiple state-of-the-art deformable models under a unified protocol. We **strongly** believe that this is the only way towards reproducable and high-quality research.

1.2.2 Core Interfaces

MenpoFit is an object oriented framework for building and fitting deformable models. It makes some basic assumptions that are common for all the implemented methods. For example, all deformable models are trained in *multiple scales* and the fitting procedure is, in most cases, *iterative*. MenpoFit's key interfaces are:

- $\bullet \ \textit{MultiScaleNonParametricFitter} \ \textbf{-} \ \textbf{multi-scale} \ \textbf{fitting} \ \textbf{class}$
- MultiScaleParametricFitter multi-scale fitting class that uses a parametric shape model
- MultiScaleNonParametricIterativeResult multi-scale result of an iterative fitting
- MultiScaleParametricIterativeResult multi-scale result of an iterative fitting using a parametric shape model

1.2.3 Deformable Models

- AAM, LucasKanadeAAMFitter, SupervisedDescentAAMFitter Active Appearance Model builder and fitters
- ATM, LucasKanadeATMFitter Active Template Model builder and fitter
- GenerativeAPS, GaussNewtonAPSFitter Active Pictorial Structures builder and fitter
- CLM, GradientDescentCLMFitter Constrained Local Model builder and fitter
- LucasKanadeFitter Lucas-Kanade Image Alignment
- SupervisedDescentFitter Supervised Descent Method builder and fitter
- DlibERT Ensemble of Regression Trees builder and fitter

1.3 Building Models

All MenpoFit's models are built in a **multi-scale** manner, i.e. in multiple resolutions. In all our core classes, this is controlled using the following three parameters:

reference_shape (*PointCloud*) First, the size of the training images is normalized by rescaling them so that the scale of their ground truth shapes matches the scale of this reference shape. In case no reference shape is provided, then the mean of the ground shapes is used. This step is essential in order to ensure consistency between the extracted features of the images.

diagonal (*int*) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. This rescaling takes place before normalizing the training images' size. Thus, *diagonal* controls the size of the model at the highest scale.

scales (*tuple* of *float*) A *tuple* with the scale value at each level, provided in ascending order, i.e. from lowest to highest scale. These values are proportional to the final resolution achieved through the reference shape normalization.

Additionally, all models have a **holistic_features** argument which expects the *callable* that will be used for extracting features from the training images.

Given the above assumptions, an example of a typical call for building a deformable model using HolisticAAM is:

Information about any kind of model can be retrieved by:

```
print(aam)
```

The next section (Fitting) explains the basics of fitting such a deformable model.

1.4 Fitting Models

1.4.1 Fitter Objects

MenpoFit has specialised classes for performing a fitting process that are called *Fitters*. All *Fitter* objects are subclasses of *MultiScaleNonParametricFitter* and *MultiScaleParametricFitter*. The main difference between those two is that a *MultiScaleParametricFitter* optimises over the parameters of a statistical shape model, whereas *MultiScaleNonParametricFitter* optimises directly the coordinates of a shape.

Their behaviour can differ depending on the deformable model. For example, a Lucas-Kanade AAM fitter (LucasKanadeAAMFitter) assumes that you have trained an AAM model (assume the *aam* we trained in the *Building* section) and can be created as:

The constructor of the *Fitter* will set the active shape and appearance components based on *n_shape* and *n_appearance* respectively, and will also perform all the necessary pre-computations based on the selected algorithm.

However, there are deformable models that are directly defined through a *Fitter* object, which is responsible for training the model as well. *SupervisedDescentFitter* is a good example. The reason for that is that the fitting process is utilised during the building procedure, thus the functionality of a *Fitter* is required. Such models can be built as:

Information about a *Fitter* can be retrieved by:

```
print(fitter)
```

1.4.2 Fitting Methods

All the deformable models that are currently implemented in MenpoFit, which are the state-of-the-art approaches in current literature, aim to find a *local optimum* of the cost function that they try to optimise, given an initialisation. The initialisation can be seen as an initial estimation of the target shape. MenpoFit's *Fitter* objects provide two functions for fitting the model to an image:

or

```
result = fitter.fit_from_bb(image, bounding_box, max_iters=20, gt_shape=None, return_costs=False, **kwargs)
```

They only differ on the type of initialisation. fit_from_shape expects a *PointCloud* as the *initial_shape*. On the other hand, the *bounding_box* argument of fit_from_bb is a *PointDirectedGraph* of 4 vertices that represents the initial bounding box. The bounding box is used in order to align the model's reference shape and use the resulting

PointCloud as the initial shape. Such a bounding box can be retrieved using the detection methods of **menpodetect**. The rest of the options are:

- max_iters (int or list of int) Defines the maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then it specifies the maximum number of iterations per scale. Note that this does not apply on all deformable models. For example, it can control the number of iterations of a Lucas-Kanade optimisation algorithm, but it does not affect the fitting of a cascaded-regression method (e.g. SDM) which has a predefined number of cascades (iterations).
- **gt_shape** (*PointCloud* or *None*) The ground truth shape associated to the image. This is *only* useful to compute the final fitting error. It is *not* used, of course, at any internal stage of the optimisation.
- **return_costs** (*bool*) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned *fitting_result*. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Thus, this option should only be used for research purposes. Finally, this argument does not apply to all deformable models.

kwargs (dict) Additional keyword arguments that can be passed to specific models.

The next section (*Result*) presents the basics of the fitting *result*.

1.5 Fitting Result

1.5.1 Objects

The fitting methods of the *Fitters* presented in the previous section return a result object. MenpoFit has three basic fitting result objects:

- Result: Basic fitting result object that holds the final shape, and optionally, the initial shape, ground truth shape and the image.
- *MultiScaleNonParametricIterativeResult*: The result of a multi-scale iterative fitting procedure. Apart from the final shape, it also stores the shapes acquired at each fitting iteration.
- MultiScaleParametricIterativeResult: The same as MultiScaleNonParametricIterativeResult with the difference that the optimisation was performed over the parameters of a statistical parametric shape model. Thus, apart from the actual shapes, it also stores the shape parameters acquired per iteration. Note that in this case, the initial shape that was provided by the user gets reconstructed using the shape model, i.e. it first gets projected in order to get the initial estimation of the shape parameters, and then gets reconstructed with those. The resulting shape is then used as initialisation for the iterative fitting process.

1.5.2 Attributes

The above result objects can provide some very useful information regarding the fitting procedure. For example, the various shapes can be retrieved as:

result.final_shape The final shape of the fitting procedure.

result.initial_shape The initial shape of the fitting procedure that was provided by the user.

result.reconstructed_initial_shape The reconstruction of the initial shape that was used to initialise the fitting procedure. It only applies for MultiScaleParametricIterativeResult.

result.image The image on which the fitting procedure was applied.

result.gt_shape The ground truth shape associated to the image.

1.5. Fitting Result 7

```
result.shapes The list of shapes acquired at each fitting iteration. It only applies on MultiScaleNonParametricIterativeResult and MultiScaleParametricIterativeResult.
```

result.costs() The cost values per iteration, if they were computed during fitting.

Also, a result can compute some error metrics, in case the *gt_shape* of the image exists:

result.final_error() The final fitting error.

result.initial_error() The initial fitting error.

```
result.errors() The list of errors acquired at each fitting iteration. It only applies on MultiScaleNonParametricIterativeResult and MultiScaleParametricIterativeResult.
```

1.6 Visualizing Objects

In Menpo, we take an opinionated stance that visualization is a key part of generating research on deformable models. Therefore, we tried to make the mental overhead of visualizing objects as low as possible.

We also took a strong step towards simple visualization by integrating some of our objects with visualization widgets for the Jupyter notebook. Remember that our widgets live on their own repository, called **menpowidgets**.

1.6.1 Visualizing Models

Without further ado, a quick example of visualising the AAM trained in the Building section with an interactive widget:

```
%matplotlib inline # This is only needed if viewing in a Jupyter notebook aam.view_aam_widget()
```

Fig. 1: Figure 1: Example of visualizing an AAM using an interactive widget.

One can visualize the only the multi-scale shape models:

```
%matplotlib inline aam.view_shape_models_widget()
```

or the appearance models:

```
%matplotlib inline
import menpo.io as mio
aam.view_appearance_models_widget()
```

The same visualization widgets can be found in other models, such as ATM, CLM etc.

1.6.2 Visualizing Fitting Result

The fitting result objects shown in *Building* can be easily visualized. Specifically, the initial and final shapes can be rendered as:

```
%matplotlib inline
result.view(render_initial_shape=True)
```

Similarly, the shapes acquired at each iteration can be visualized as:

```
%matplotlib inline fr.view_iterations()
```

and the corresponding errors as:

```
%matplotlib inline fr.plot_errors()
```

Finally, a fitting result can also be analysed through an interactive widget as:

```
%matplotlib inline fr.view_widget()
```

Fig. 2: Figure 2: Example of visualizing the iterations of a fitting procedure using an interactive widget.

1.7 References

This is an indicative list of papers relevant to the methods that are implemented in MenpoFit. They are listed in alphabetical order of the first author's surname.

- 1. J. Alabort-i-Medina, and S. Zafeiriou. "A Unified Framework for Compositional Fitting of Active Appearance Models", arXiv:1601.00199.
- 2. J. Alabort-i-Medina, and S. Zafeiriou. "Bayesian Active Appearance Models", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
- 3. J. Alabort-i-Medina, and S. Zafeiriou. "Unifying Holistic and Parts-Based Deformable Model Fitting", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
- 4. E. Antonakos, J. Alabort-i-Medina, G. Tzimiropoulos, and S. Zafeiriou. "Feature-based Lucas-Kanade and Active Appearance Models", IEEE Transactions on Image Processing, vol. 24, no. 9, pp. 2617-2632, 2015.
- 5. E. Antonakos, J. Alabort-i-Medina, G. Tzimiropoulos, and S. Zafeiriou. "HOG Active Appearance Models", IEEE International Conference on Image Processing (ICIP), 2014.
- 6. E. Antonakos, J. Alabort-i-Medina, and S. Zafeiriou. "Active Pictorial Structures", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.
- 7. A.B. Ashraf, S. Lucey, and T. Chen. "Fast Image Alignment in the Fourier Domain", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010.
- 8. A. Asthana, S. Zafeiriou, S. Cheng, and M. Pantic. "Robust discriminative response map fitting with constrained local models", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.
- 9. S. Baker, and I. Matthews. "Lucas-Kanade 20 years on: A unifying framework", International Journal of Computer Vision, vol. 56, no. 3, pp. 221-255, 2004.

1.7. References 9

- 10. P.N. Belhumeur, D.W. Jacobs, D.J. Kriegman, and N. Kumar. "Localizing parts of faces using a consensus of exemplars", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 12, pp. 2930-2940, 2013.
- 11. D.S. Bolme, J.R. Beveridge, B.A. Draper, and Y.M. Lui. "Visual Object Tracking using Adaptive Correlation Filters", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010.
- 12. T.F. Cootes, G.J. Edwards, and C.J. Taylor. "Active Appearance Models", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 6, pp. 681–685, 2001.
- 13. T.F. Cootes, and C.J. Taylor. "Active shape models-'smart snakes'", British Machine Vision Conference (BMVC), 1992.
- 14. T.F. Cootes, C.J. Taylor, D.H. Cooper, and J. Graham. "Active Shape Models their training and application", Computer Vision and Image Understanding, vol. 61, no. 1, pp. 38-59, 1995.
- 15. D. Cristinacce, and T.F. Cootes. "Feature Detection and Tracking with Constrained Local Models", British Machine Vision Conference (BMVC), 2006.
- G.D. Evangelidis, and E.Z. Psarakis. "Parametric Image Alignment Using Enhanced Correlation Coefficient Maximization", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 10, pp. 1858-1865, 2008.
- 17. R. Gross, I. Matthews, and S. Baker. "Generic vs. person specific Active Appearance Models", Image and Vision Computing, vol. 23, no. 12, pp. 1080-1093, 2005.
- 18. V. Kazemi, and J. Sullivan. "One millisecond face alignment with an ``ensemble of regression trees", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
- 19. B.D. Lucas, and T. Kanade, "An iterative image registration technique with an application to stereo vision", International Joint Conference on Artificial Intelligence, 1981.
- 20. I. Matthews, and S. Baker. "Active Appearance Models Revisited", International Journal of Computer Vision, 60(2): 135-164, 2004.
- 21. G. Papandreou, and P. Maragos. "Adaptive and constrained algorithms for `inverse compositional active appearance model fitting", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2008.
- 22. D. Ross, J. Lim, R.S. Lin, and M.H. Yang. "Incremental Learning for Robust Visual Tracking". International Journal on Computer Vision, vol. 77, no. 1-3, pp. 125-141, 2007.
- 23. J.M. Saragih, S. Lucey, and J.F. Cohn. "Deformable model fitting by regularized landmark mean-shift", International Journal of Computer Vision, vol. 91, no. 2, pp. 200–215, 2011.
- 24. J.M. Saragih, and R. Goecke. "Learning AAM fitting through simulation", Pattern Recognition, vol. 42, no. 11, pp. 2628–2636, 2009.
- 25. G. Tzimiropoulos, J. Alabort-i-Medina, S. Zafeiriou, and M. Pantic. "Active Orientation Models for Face Alignment in-the-wild", IEEE Transactions on Information Forensics and Security, Special Issue on Facial Biometrics in-the-wild, vol. 9, no. 12, pp. 2024-2034, 2014.
- 26. G. Tzimiropoulos, and M. Pantic. "Gauss-Newton Deformable Part Models for Face Alignment In-the-Wild", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.
- 27. G. Tzimiropoulos, J. Alabort-i-Medina, S. Zafeiriou, and M. Pantic. "Generic Active Appearance Models Revisited", Asian Conference on Computer Vision, Springer, 2012.
- 28. G. Tzimiropoulos, M. Pantic. "Optimization problems for fast AAM fitting in-the-wild", IEEE International Conference on Computer Vision (ICCV), 2013.
- 29. G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. "Robust and efficient parametric face alignment", IEEE International Conference on Computer Vision (ICCV), 2011.

- 30. G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. "Subspace Learning from Image Gradient Orientations", IEEE Transactions on Pattern Analysis and Machine Intelligence. vol. 34, no. 12, pp. 2454-2466, 2012.
- 31. X. Xiong, and F. De la Torre. "Supervised descent method and its applications to face alignment", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.

1.7. References

CHAPTER

TWO

THE MENPOFIT API

This section attempts to provide a simple browsing experience for the MenpoFit documentation. In MenpoFit, we use legible docstrings, and therefore, all documentation should be easily accessible in any sensible IDE (or IPython) via tab completion. However, this section should make most of the core classes available for viewing online.

2.1 Deformable Models

2.1.1 menpofit.aam

Active Appearance Model

AAM is a generative model that consists of a statistical parametric model of the shape and the appearance of an object. MenpoFit has several AAMs which differ in the manner that they compute the warp (thus represent the appearance features).

AAM

Bases: object

Class for training a multi-scale holistic Active Appearance Model. Please see the references for a basic list of relevant papers.

- images (list of menpo.image.Image) The list of training images.
- group (str or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.

- **reference_shape** (*menpo.shape.PointCloud* or None, optional) The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If None, then the mean shape will be used.
- diagonal (*int* or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- transform (subclass of DL and DX, optional) A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- max_appearance_components (int, float, list of those or None, optional) The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (*bool*, optional) If True, then the progress of building the AAM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

References

appearance reconstructions (appearance parameters, n iters per scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a AAMResult object, in order to generate the appearance reconstructions of a fitting procedure.

Parameters

- appearance_parameters (*list* of (n_params,) ndarray) A set of appearance parameters per fitting iteration. It can be retrieved as a property of an AAMResult object.
- n_iters_per_scale (*list* of *int*) The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a AAMResult object.

Returns appearance_reconstructions (*list* of *menpo.image.Image*) – *List* of the appearance reconstructions that correspond to the provided parameters.

build fitter interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. LucasKanadeAAMFitter).

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained AAM with a new set of training images.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- shape_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- appearance_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the AAM will be printed.
- **batch_size** (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

instance (shape_weights=None, appearance_weights=None, scale_index=- 1)

Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- appearance_weights ((n_weights,) ndarray or list or None, optional) The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.
- scale index (int, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

random_instance (scale_index=- 1)

Generates a random instance of the AAM.

Parameters scale_index (int, optional) - The scale to be used.

2.1. Deformable Models 15

Returns image (*menpo.image.Image*) – The AAM instance.

property n_scales

Returns the number of scales.

Type int

HolisticAAM

menpofit.aam.HolisticAAM alias of AAM

MaskedAAM

Bases: AAM

Class for training a multi-scale patch-based Masked Active Appearance Model. The appearance of this model is formulated by simply masking an image with a patch-based mask.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo feature for a list of potential features.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- patch_shape ((int, int), optional) The size of the patches of the mask that is used to sample the appearance vectors.

- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- max_appearance_components (int, float, list of those or None, optional) The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- verbose (bool, optional) If True, then the progress of building the AAM will be printed.
- **batch_size** (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

appearance_reconstructions (appearance_parameters, n_iters_per_scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a AAMResult object, in order to generate the appearance reconstructions of a fitting procedure.

Parameters

- appearance_parameters (*list* of (n_params,) *ndarray*) A set of appearance parameters per fitting iteration. It can be retrieved as a property of an AAMResult object.
- n_iters_per_scale (*list* of *int*) The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a AAMResult object.

Returns appearance_reconstructions (*list* of *menpo.image.Image*) – *List* of the appearance reconstructions that correspond to the provided parameters.

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. LucasKanadeAAMFitter).

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained AAM with a new set of training images.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- shape_forgetting_factor([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing

batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.

- appearance_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the AAM will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

instance (shape_weights=None, appearance_weights=None, scale_index=- 1)

Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- appearance_weights ((n_weights,) ndarray or list or None, optional) The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.
- scale_index (int, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

```
random_instance (scale_index=- 1)
```

Generates a random instance of the AAM.

Parameters scale_index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

property n_scales

Returns the number of scales.

Type int

LinearAAM

Bases: AAM

Class for training a multi-scale Linear Active Appearance Model.

Parameters

• images (list of menpo.image.Image) – The list of training images.

- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo feature for a list of potential features.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape
 so that the diagonal of its bounding box matches the provided value. In other words, this
 parameter controls the size of the model at the highest scale. If None, then the reference
 shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- transform (subclass of DL and DX, optional) A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- max_appearance_components (int, float, list of those or None, optional) The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (*bool*, optional) If True, then the progress of building the AAM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

appearance_reconstructions (appearance_parameters, n_iters_per_scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a AAMResult object, in order to generate the appearance reconstructions of a fitting procedure.

Parameters

• appearance_parameters (*list* of (n_params,) *ndarray*) – A set of appearance parameters per fitting iteration. It can be retrieved as a property of an AAMResult object.

2.1. Deformable Models 19

• n_iters_per_scale (*list* of *int*) – The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a AAMResult object.

Returns appearance_reconstructions (*list* of *menpo.image.Image*) – *List* of the appearance reconstructions that correspond to the provided parameters.

build fitter interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. LucasKanadeAAMFitter).

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter_interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained AAM with a new set of training images.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- shape_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- appearance_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the AAM will be printed.
- **batch_size** (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

instance (shape_weights=None, appearance_weights=None, scale_index=- 1)

Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- appearance_weights ((n_weights,) ndarray or list or None, optional) The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.
- scale_index (int, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

```
random instance (scale index=- 1)
```

Generates a random instance of the AAM.

Parameters scale index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

property n_scales

Returns the number of scales.

Type int

LinearMaskedAAM

```
class menpofit.aam.LinearMaskedAAM (images,
                                                                        holistic_features=<function
                                                        group=None,
                                                               reference_shape=None,
                                             no\_op>,
                                             agonal=None,
                                                                      scales=(0.5,
                                                                                             1.0),
                                                                          shape_model_cls=<class
                                             patch\_shape=(17,
                                                                  17).
                                             'menpofit.modelinstance.OrthoPDM'>,
                                             max_shape_components=None,
                                                                                   verbose=False,
                                             max_appearance_components=None,
                                             batch_size=None)
```

Bases: AAM

Class for training a multi-scale Linear Masked Active Appearance Model.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo, feature for a list of potential features.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (*int* or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- patch_shape ((int, int), optional) The size of the patches of the mask that is used to sample the appearance vectors.

- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- max_appearance_components (int, float, list of those or None, optional) The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (bool, optional) If True, then the progress of building the AAM will be printed.
- **batch_size** (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

appearance_reconstructions (appearance_parameters, n_iters_per_scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a AAMResult object, in order to generate the appearance reconstructions of a fitting procedure.

Parameters

- appearance_parameters (*list* of (n_params,) *ndarray*) A set of appearance parameters per fitting iteration. It can be retrieved as a property of an AAMResult object.
- n_iters_per_scale (*list* of *int*) The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a AAMResult object.

Returns appearance_reconstructions (*list* of *menpo.image.Image*) – *List* of the appearance reconstructions that correspond to the provided parameters.

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. LucasKanadeAAMFitter).

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained AAM with a new set of training images.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- shape_forgetting_factor([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing

batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.

- appearance_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the AAM will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

instance (shape_weights=None, appearance_weights=None, scale_index=- 1)

Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- appearance_weights ((n_weights,) ndarray or list or None, optional) The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.
- scale_index (int, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

```
random_instance (scale_index=- 1)
```

Generates a random instance of the AAM.

Parameters scale_index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

property n_scales

Returns the number of scales.

Type int

PatchAAM

```
holistic features=<function
class menpofit.aam.PatchAAM(images,
                                               group=None,
                                     reference_shape=None,
                                                             diagonal=None,
                                                                               scales=(0.5,
                                                                                              1.0),
                                                                     patch_normalisation=<function</pre>
                                     patch\_shape=(17,
                                                           17),
                                                           shape_model_cls=<class
                                                                                             'men-
                                     no\_op>,
                                     pofit.modelinstance.OrthoPDM'>, max_shape_components=None,
                                     max_appearance_components=None,
                                                                                    verbose=False,
                                     batch_size=None)
```

Bases: AAM

Class for training a multi-scale Patch-Based Active Appearance Model. The appearance of this model is formulated by simply sampling patches around the image's landmarks.

23

Parameters

2.1. Deformable Models

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo feature for a list of potential features.
- **reference_shape** (*menpo.shape.PointCloud* or None, optional) The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- patch_normalisation (*list* of *callable* or a single *callable*, optional) The normalisation function to be applied on the extracted patches. If *list*, then it must have length equal to the number of scales. If a single patch normalization *callable*, then this is the one applied to all scales.
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- max_appearance_components (int, float, list of those or None, optional) The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (*bool*, optional) If True, then the progress of building the AAM will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

appearance_reconstructions (appearance_parameters, n_iters_per_scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a AAMResult object, in order to generate the appearance reconstructions of a fitting procedure.

Parameters

- **appearance_parameters** (*list* of *ndarray*) A set of appearance parameters per fitting iteration. It can be retrieved as a property of a AAMResult object.
- n_iters_per_scale (*list* of *int*) The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a AAMResult object.

Returns appearance_reconstructions (*list* of *ndarray*) – List of the appearance reconstructions that correspond to the provided parameters.

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface. It only applies in case you wish to fit the AAM with a Lucas-Kanade algorithm (i.e. LucasKanadeAAMFitter).

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained AAM with a new set of training images.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- shape_forgetting_factor([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- appearance_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the appearance model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the AAM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

instance (shape weights=None, appearance weights=None, scale index=-1)

Generates a novel AAM instance given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

• **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) – The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.

- appearance_weights ((n_weights,) ndarray or list or None, optional) The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.
- **scale_index** (*int*, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

```
random_instance (scale_index=- 1)
```

Generates a random instance of the AAM.

Parameters scale_index (int, optional) - The scale to be used.

Returns image (menpo.image.Image) – The AAM instance.

property n_scales

Returns the number of scales.

Type int

Fitters

An AAM can be optimised either in a gradient descent manner (Lucas-Kanade) or using cascaded regression (Supervised Descent).

LucasKanadeAAMFitter

```
class menpofit.aam.LucasKanadeAAMFitter (aam, lk\_algorithm\_cls = < class 'menpofit.aam.algorithm.lk.WibergInverseCompositional'>, n\_shape=None, n\_appearance=None, sam-pling=None)
```

Bases: AAMFitter

Class for defining an AAM fitter using the Lucas-Kanade optimisation.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided *max_iters*.

- aam (AAM or subclass) The trained AAM model.
- **lk_algorithm_cls** (*class*, optional) The Lukas-Kanade optimisation algorithm that will get applied. The possible algorithms are:

Class	Method
AlternatingForwardCompositional	Alternating
AlternatingInverseCompositional	
ModifiedAlternatingForwardCompositional	Modified Alternating
ModifiedAlternatingInverseCompositional	
ProjectOutForwardCompositional	Project-Out
ProjectOutInverseCompositional	
SimultaneousForwardCompositional	Simultaneous
SimultaneousInverseCompositional	
WibergForwardCompositional	Wiberg
WibergInverseCompositional	

- n_shape (int or float or list of those or None, optional) The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_shape_components during training.
- n_appearance (int or float or list of those or None, optional) The number of appearance components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_appearance_components during training.
- **sampling** (*list* of *int* or *ndarray* or None) It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

appearance_reconstructions (appearance_parameters, n_iters_per_scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a AAMResult object, in order to generate the appearance reconstructions of a fitting procedure.

Parameters

- appearance_parameters (*list* of (n_params,) *ndarray*) A set of appearance parameters per fitting iteration. It can be retrieved as a property of an AAMResult object.
- n_iters_per_scale (*list* of *int*) The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a AAMResult object.

27

Returns appearance_reconstructions (*list* of *menpo.image.Image*) – *List* of the appearance reconstructions that correspond to the provided parameters.

fit_from_bb (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

Parameters

• image (menpo.image.Image or subclass) – The image to be fitted.

2.1. Deformable Models

- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within an <code>AAMResult</code>.

- image (menpo.image.Image or subclass) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

property aam

The trained AAM model.

Type AAM or subclass

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *closure*

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

SupervisedDescentAAMFitter

Bases: SupervisedDescentFitter

Class for training a multi-scale cascaded-regression Supervised Descent AAM fitter.

- images (list of menpo.image.Image) The list of training images.
- aam (AAM or subclass) The trained AAM model.
- **group** (*str* or None, optional) The landmark group that will be used to train the fitter. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- bounding_box_group_glob (glob or None, optional) Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- n_shape (int or float or list of those or None, optional) The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will

be applied for all scales. If *list*, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to *max_shape_components* during training.

- n_appearance (int or float or list of those or None, optional) The number of appearance components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_appearance_components during training.
- **sampling** (*list* of *int* or *ndarray* or None) It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.
- **sd_algorithm_cls** (*class*, optional) The Supervised Descent algorithm to be used. The possible algorithms are:

Class	Features	Regression
MeanTemplateNewton	Mean Template	IRLRegression
<i>MeanTemplateGaussNewton</i>		IIRLRegression
ProjectOutNewton	Project-Out	IRLRegression
ProjectOutGaussNewton		IIRLRegression
AppearanceWeightsNewton	App. Weights	IRLRegression
AppearanceWeightsGaussNewton		IIRLRegression

- n_iterations (*int* or *list* of *int*, optional) The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.
- n_perturbations (*int* or None, optional) The number of perturbations to be generated from the provided bounding boxes.
- **perturb_from_gt_bounding_box** (*callable*, optional) The function that will be used to generate the perturbations.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.
- **verbose** (*bool*, optional) If True, then the progress of training will be printed.
- **fit_from_bb** (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

- **image** (*menpo.image.Image* or subclass) The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.

- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- **image** (*menpo.image.Image* or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- bounding_box_group_glob (glob or None, optional) Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- **verbose** (*bool*, optional) If True, then the progress of training will be printed.

2.1. Deformable Models 31

• **batch_size** (*int* or None, optional) – If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a MultiScaleParametricIterativeResult.

Parameters

- image (menpo.image.Image or subclass) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *closure*

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type list of int or float

Lucas-Kanade Optimisation Algorithms

AlternatingForwardCompositional

 $\verb|class| menpofit.aam. AlternatingForwardCompositional| (aam_interface, eps=1e-05)|$

Bases: Alternating

Alternating Forward Compositional (AFC) Gauss-Newton algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

- **image** (*menpo.image.Image*) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.

- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image* or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

AlternatingInverseCompositional

class menpofit.aam.AlternatingInverseCompositional(aam_interface, eps=1e-05)
Bases: Alternating

Alternating Inverse Compositional (AIC) Gauss-Newton algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image* or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

ModifiedAlternatingForwardCompositional

class menpofit.aam.ModifiedAlternatingForwardCompositional ($aam_interface$, eps=1e-05)

Bases: ModifiedAlternating

Modified Alternating Forward Compositional (MAFC) Gauss-Newton algorithm

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- **gt_shape** (*menpo.shape.PointCloud* or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type menpo.image.Image or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

ModifiedAlternatingInverseCompositional

class menpofit.aam.ModifiedAlternatingInverseCompositional ($aam_interface$, eps=1e-05)

Bases: ModifiedAlternating

Modified Alternating Inverse Compositional (MAIC) Gauss-Newton algorithm

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (bool, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image* or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

2.1. Deformable Models 35

ProjectOutForwardCompositional

class menpofit.aam.ProjectOutForwardCompositional(aam_interface, eps=1e-05)

Bases: ProjectOut

Project-out Forward Compositional (POFC) Gauss-Newton algorithm.

$project_out(J)$

Projects-out the appearance subspace from a given vector or matrix.

Type ndarray

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- **image** (*menpo.image.Image*) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type menpo.image.Image or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

37

ProjectOutInverseCompositional

class menpofit.aam.ProjectOutInverseCompositional (aam_interface, eps=1e-05) Bases: ProjectOut

Project-out Inverse Compositional (POIC) Gauss-Newton algorithm.

$project_out(J)$

Projects-out the appearance subspace from a given vector or matrix.

Type *ndarray*

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type menpo.image.Image or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

2.1. Deformable Models

SimultaneousForwardCompositional

class menpofit.aam.SimultaneousForwardCompositional(aam_interface, eps=1e-05)

Bases: Simultaneous

Simultaneous Forward Compositional (SFC) Gauss-Newton algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image* or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

SimultaneousInverseCompositional

class menpofit.aam.SimultaneousInverseCompositional(aam_interface, eps=1e-05)

Bases: Simultaneous

Simultaneous Inverse Compositional (SIC) Gauss-Newton algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image* or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

2.1. Deformable Models

WibergForwardCompositional

class menpofit.aam.WibergForwardCompositional(aam_interface, eps=1e-05)

Bases: Wiberg

Wiberg Forward Compositional (WFC) Gauss-Newton algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image* or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

WibergInverseCompositional

class menpofit.aam.WibergInverseCompositional(aam_interface, eps=1e-05)

Bases: Wiberg

Wiberg Inverse Compositional (WIC) Gauss-Newton algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- **gt_shape** (*menpo.shape.PointCloud* or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property template

Returns the template of the AAM (usually the mean of the appearance model).

Type *menpo.image.Image* or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

Supervised Descent Optimisation Algorithms

AppearanceWeightsNewton

```
class menpofit.aam. Appearance Weights Newton (aam\_interface, n\_iterations = 3, compute\_error = < function euclidean\_bb\_normalised\_error >, alpha = 0, bias = True)
```

Bases: AppearanceWeights

Class for training a cascaded-regression Newton algorithm using Incremental Regularized Linear Regression (IRLRegression) given a trained AAM model. The algorithm uses the projection weights of the appearance vectors as features in the regression.

Parameters

• aam_interface (The AAM interface class from *menpofit.aam.algorithm.lk.*) – Existing interfaces include:

'LucasKanadeStandardInterface'	Suitable for holistic AAMs
'LucasKanadeLinearInterface'	Suitable for linear AAMs
'LucasKanadePatchInterface'	Suitable for patch-based AAMs

- n_iterations (*int*, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- current_shapes (list of menpo.shape.PointCloud) The list of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- verbose (bool, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

project(J)

Projects a given vector or matrix onto the appearance subspace.

Type ndarray

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

• image (menpo.image.Image or subclass) – The image to be fitted.

- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (AAMAlgorithmResult) - The parametric iterative fitting result.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

AppearanceWeightsGaussNewton

Bases: AppearanceWeights

Class for training a cascaded-regression Gauss-Newton algorithm using Indirect Incremental Regularized Linear Regression (*IIRLRegression*) given a trained AAM model. The algorithm uses the projection weights of the appearance vectors as features in the regression.

Parameters

• aam_interface (The AAM interface class from *menpofit.aam.algorithm.lk.*) – Existing interfaces include:

'LucasKanadeStandardInterface'	Suitable for holistic AAMs
'LucasKanadeLinearInterface'	Suitable for linear AAMs
'LucasKanadePatchInterface'	Suitable for patch-based AAMs

- n_iterations (int, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- alpha2 (*float*, optional) The regularization parameter of the Hessian matrix.
- **bias** (*bool*, optional) Flag that controls whether to use a bias term.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

project(J)

Projects a given vector or matrix onto the appearance subspace.

Type *ndarray*

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

train (images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- verbose (bool, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

MeanTemplateNewton

```
class menpofit.aam.MeanTemplateNewton (aam\_interface, n\_iterations = 3, compute\_error = < function euclidean\_bb\_normalised\_error >, alpha = 0, bias = True)
```

Bases: MeanTemplate

Class for training a cascaded-regression Newton algorithm using Incremental Regularized Linear Regression (IRLRegression) given a trained AAM model. The algorithm uses the centered appearance vectors as features in the regression.

Parameters

aam_interface (The AAM interface class from menpofit.aam.algorithm.lk.) – Existing interfaces include:

'LucasKanadeStandardInterface'	Suitable for holistic AAMs
'LucasKanadeLinearInterface'	Suitable for linear AAMs
'LucasKanadePatchInterface'	Suitable for patch-based AAMs

- n_iterations (*int*, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type *subclass* of DL and DX

MeanTemplateGaussNewton

```
class menpofit.aam.MeanTemplateGaussNewton (aam\_interface, n\_iterations=3, compute\_error=<function euclidean\_bb\_normalised\_error>, alpha=0, alpha2=0, bias=True)
```

Bases: MeanTemplate

Class for training a cascaded-regression Gauss-Newton algorithm using Indirect Incremental Regularized Linear Regression (*IIRLRegression*) given a trained AAM model. The algorithm uses the centered appearance vectors as features in the regression.

Parameters

aam_interface (The AAM interface class from menpofit.aam.algorithm.lk.) – Existing interfaces include:

'LucasKanadeStandardInterface'	Suitable for holistic AAMs
'LucasKanadeLinearInterface'	Suitable for linear AAMs
'LucasKanadePatchInterface'	Suitable for patch-based AAMs

- n_iterations (int, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- alpha2 (*float*, optional) The regularization parameter of the Hessian matrix.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.

- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting result (AAMAlgorithmResult) – The parametric iterative fitting result.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

ProjectOutNewton

```
 \begin{array}{c} \textbf{class} \text{ menpofit.aam.ProjectOutNewton} \ (aam\_interface, \\ compute\_error = < function \\ clidean\_bb\_normalised\_error >, alpha = 0, bias = True) \\ \textbf{Bases:} \ \texttt{ProjectOut} \end{array}
```

Class for training a cascaded-regre

Class for training a cascaded-regression Newton algorithm using Incremental Regularized Linear Regression (IRLRegression) given a trained AAM model. The algorithm uses the projected-out appearance vectors as features in the regression.

Parameters

aam_interface (The AAM interface class from menpofit.aam.algorithm.lk.) – Existing interfaces include:

Class	AAM
'LucasKanadeStandardInterface'	Suitable for holistic AAMs
'LucasKanadeLinearInterface'	Suitable for linear AAMs
'LucasKanadePatchInterface'	Suitable for patch-based AAMs

- n_iterations (*int*, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- **bias** (*bool*, optional) Flag that controls whether to use a bias term.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- verbose (bool, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

$project_out(J)$

Projects-out the appearance subspace from a given vector or matrix.

Type ndarray

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

2.1. Deformable Models

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (list of menpo.shape.PointCloud) The list of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

ProjectOutGaussNewton

class menpofit.aam.ProjectOutGaussNewton ($aam_interface$, $n_iterations=3$, $compute_error=<function$ $euclidean_bb_normalised_error>$, alpha=0, alpha2=0, bias=True)

Bases: ProjectOut

Class for training a cascaded-regression Gauss-Newton algorithm using Indirect Incremental Regularized Linear Regression (*IIRLRegression*) given a trained AAM model. The algorithm uses the projected-out appearance vectors as features in the regression.

Parameters

• aam_interface (The AAM interface class from *menpofit.aam.algorithm.lk.*) – Existing interfaces include:

'LucasKanadeStandardInterface'	Suitable for holistic AAMs
'LucasKanadeLinearInterface'	Suitable for linear AAMs
'LucasKanadePatchInterface'	Suitable for patch-based AAMs

- n_iterations (int, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- alpha2 (*float*, optional) The regularization parameter of the Hessian matrix.
- **bias** (*bool*, optional) Flag that controls whether to use a bias term.

 $\verb|increment| (images, gt_shapes, current_shapes, prefix=", verbose=False)|$

Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (list of menpo.shape.PointCloud) The list of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

$project_out(J)$

Projects-out the appearance subspace from a given vector or matrix.

Type *ndarray*

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (AAMAlgorithmResult) – The parametric iterative fitting result.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

property appearance_model

Returns the appearance model of the AAM.

Type menpo.model.PCAModel

2.1. Deformable Models

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

Fitting Result

AAMResult

Bases: MultiScaleParametricIterativeResult

Class for storing the multi-scale iterative fitting result of an AAM. It holds the shapes, shape parameters, appearance parameters and costs per iteration.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

Parameters

- results (*list* of AAMAlgorithmResult) The *list* of optimization results per scale.
- **scales** (*list* or *tuple*) The *list* of scale values per scale (low to high).
- **affine_transforms** (*list* of *menpo.transform.Affine*) The list of affine transforms per scale that transform the shapes into the original image space.
- scale_transforms (list of menpo.shape.Scale) The list of scaling transforms per scale.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

displacements stats(stat type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError - type must be 'mean', 'median', 'min' or 'max'

errors (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-
                    line width=2,
                                    render markers=True,
                                                             marker style='o',
                                                                                  marker size=4,
               marker_face_colour='b',
                                           marker_edge_colour='k',
                                                                        marker\_edge\_width=1.0,
               render_axes=True,
                                          axes_font_name='sans-serif',
                                                                              axes\_font\_size=10,
                                              axes_font_weight='normal',
               axes_font_style='normal',
                                                                               axes_x_limits=0.0,
               axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(10, 6),
               render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of the cost function evolution at each fitting iteration.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- **new_figure** (*bool*, optional) If True, a new figure is created.
- render lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line style ({'-', '--', '-.', ':'}, optional) The style of the lines.
- line_width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.

• marker_style (marker, optional) – The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers.If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (*int*, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.

- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker edge width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes font size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_errors (compute_error=None,
                                        figure_id=None,
                                                            new_figure=False,
                                                                                render_lines=True,
                line colour='b',
                                      line_style='-',
                                                         line\_width=2,
                                                                             render_markers=True,
                marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
                marker\_edge\_width=1.0,
                                               render axes=True,
                                                                       axes_font_name='sans-serif',
                                         axes_font_style='normal',
                                                                        axes_font_weight='normal',
                axes\_font\_size=10,
                axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None,
                figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the error evolution at each fitting iteration.
```

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -:}
```

- line_width (float, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed_initial_error (compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed initial shapes[0]*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view (figure id=None, new figure=False, render image=True, render final shape=True, render_initial_shape=False, subplots enabled=True, $render_gt_shape=False,$ channels=None, interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt marker face colour='y', gt marker edge colour='k', gt line colour='y', render lines=True, line style='-', line width=2, render markers=True. marker style='o', marker size=4, render numbering=False, numbers horizontal align='center', $marker\ edge\ width=1.0,$ numbers vertical align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title="', legend_font_name='sans-serif', legend_font_style='normal', legend_font_weight='normal', legend font size=10, legend_marker_scale=None, end location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, *legend_border=True*, legend_border_padding=None, *legend_shadow=False*, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape,

ground truth shape and the image, id they were provided.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (bool, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (bool, optional) If True and the ground truth shape exists, then it gets rendered.
- subplots_enabled (bool, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- channels (int or list of int or all or None) If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- cmap_name (str, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (float, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.

• **final_marker_face_colour** (See Below, optional) - The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) - The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- line_style (str or list of str, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

2.1. Deformable Models

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (*bool*, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- **legend_font_size** (*int*, optional) The font size of the legend.
- legend_font_weight (See Below, optional)—The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- legend_border_axes_pad (*float*, optional) The pad between the axes and legend border
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.

- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view iterations (figure id=None, new figure=False, iters=None, render image=True, subplots enabled=False, channels=None. interpolation='bilinear', render lines=True, cmap name=None, alpha=1.0, masked=True. line_style='-', line width=2, line colour=None, render_markers=True, marker edge colour=None, marker face colour=None, marker style='o', $marker\ size=4$, marker_edge_width=1.0, render numbering=False, numbers horizontal align='center'. numbers vertical align='bottom'. numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes font size=10, axes font style='normal', axes font weight='normal', $axes \ x \ limits=None,$ axes y limits=None, $axes \ x \ ticks=None,$ axes y ticks=None, figure size=(7,7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (str (See Below), optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- ullet masked (bool, optional) If True, then the image is rendered as masked.
- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_colour (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_width (*float* or *list* of *float*, optional) – The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different

value per iteration shape.

- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (*str* (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend title (str, optional) The title of the legend.
- **legend_font_name** (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- legend_location (int, optional) The location of the legend. The predefined values
 are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's

2.1. Deformable Models 67

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property appearance_parameters

Returns the *list* of appearance parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type *list* of (n_params,) *ndarray*

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property n_iters_per_scale

Returns the number of iterations per scale of the fitting process.

Type list of int

property n_scales

Returns the number of scales used during the fitting process.

Type int

property reconstructed_initial_shapes

Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

Type list of menpo.shape.PointCloud

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type list of (n_params,) ndarray

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists) and *final_shape*.

Type *list* of *menpo.shape.PointCloud*

AAMAlgorithmResult

Bases: ParametricIterativeResult

Class for storing the iterative result of an AAM optimisation algorithm.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of shapes per iteration. The first and last members correspond to the initial and final shapes, respectively.
- **shape_parameters** (*list* of (n_shape_parameters,) *ndarray*) The *list* of shape parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.
- appearance_parameters (list of (n_appearance_parameters,) ndarray) The list of appearance parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.
- initial_shape (*menpo.shape.PointCloud* or None, optional) The initial shape from which the fitting process started. If None, then no initial shape is assigned.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- **gt_shape** (*menpo.shape.PointCloud* or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.
- **costs** (*list* of *float* or None, optional) The *list* of cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type list of ndarray

displacements_stats (stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be 'mean', 'median', 'min' or 'max'

errors (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error(compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (float) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-
                   line width=2,
                                     render_markers=True,
                                                             marker_style='o',
                                                                                  marker\_size=4,
               marker_face_colour='b',
                                            marker_edge_colour='k',
                                                                         marker\_edge\_width=1.0,
               render_axes=True,
                                          axes_font_name='sans-serif',
                                                                               axes\_font\_size=10,
               axes_font_style='normal',
                                              axes_font_weight='normal',
                                                                               axes\_x\_limits=0.0,
               axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6),
               render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of the cost function evolution at each fitting iteration.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'}, optional) The style of the lines.
- line_width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- \bullet marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}

or

(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.

- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_errors (compute_error=None, figure_id=None,
                                                           new_figure=False,
                                                                                render lines=True,
                line_colour='b',
                                      line_style='-',
                                                         line\_width=2,
                                                                             render_markers=True,
                marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
                marker\ edge\ width=1.0,
                                              render axes=True,
                                                                       axes font name='sans-serif',
                axes\_font\_size=10,
                                         axes_font_style='normal',
                                                                        axes_font_weight='normal',
                axes x limits=0.0, axes y limits=None, axes x ticks=None, axes y ticks=None,
                figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the error evolution at each fitting iteration.
```

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -:}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker edge width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes font name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes font size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
reconstructed_initial_error(compute_error=None)
```

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed_initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

2.1. Deformable Models 75

to_result(pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a Result instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (*bool*, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

render_final_shape=True, view (figure_id=None, new_figure=False, render_image=True, render_initial_shape=False, render_gt_shape=False, *subplots_enabled=True*, nels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line style='-', line width=2, marker style='o', render markers=True, $marker\ size=4,$ $marker_edge_width=1.0$, render numbering=False, numbers_horizontal_align='center', numbers vertical align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render legend=True, legend title=", legend font name='sans-serif', legend font style='normal', legend font size=10, legend font weight='normal', legend marker scale=None, end location=2. legend bbox to anchor=(1.05,legend border axes pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

- **figure_id** (*object*, optional) The id of the figure to be used.
- **new_figure** (*bool*, optional) If True, a new figure is created.
- render_image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (*bool*, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• interpolation (See Below, optional) — The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked(bool, optional) If True, then the image is rendered as masked.
- **final_marker_face_colour** (See Below, optional) The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) – The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **line_style** (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.

• numbers_font_name (See Below, optional) - The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- legend_font_size (int, optional) The font size of the legend.
- legend_font_weight (See Below, optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- legend_border_axes_pad (float, optional) The pad between the axes and legend border.
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the

Image's height. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view_iterations (figure_id=None, render_image=True, new_figure=False, iters=None. *subplots_enabled=False*, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line style='-', line width=2, line colour=None, render markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', marker size=4, $marker_edge_width=1.0$, render numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', *numbers_font_size=10*, bers_font_style='normal', numbers_font_weight='normal', numbers font colour='k', render legend=True, legend title=". legend font name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend_bbox_to_anchor=(1.05, *legend_location=2*, 1.0), legend_border_axes_pad=None, $legend_n_columns=1$, legend horizontal spacing=None, legend vertical spacing=None, legend border=True. *legend_border_padding=None*, legend shadow=False, legend rounded corners=False, render axes=False, axes font name='sansserif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_y_limits=None, axes_x_limits=None, $axes_x_ticks=None,$ axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

2.1. Deformable Models

• **interpolation** (*str* (See Below), optional) – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked(bool, optional) If True, then the image is rendered as masked.
- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:, :}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_colour (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (str (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (*str* (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• axes_font_size (int, optional) – The font size of the axes.

- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property appearance_parameters

Returns the *list* of appearance parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type list of (n_params,) ndarray

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type *menpo.shape.Image* or *subclass* or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

2.1. Deformable Models

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property reconstructed_initial_shape

Returns the initial shape's reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type menpo.shape.PointCloud

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *reconstructed_initial_shape* and *final_shape*.

Type *list* of (n_params,) *ndarray*

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists), *reconstructed_initial_shape* and *final_shape*.

Type *list* of *menpo.shape.PointCloud*

Pre-Trained Model

load balanced frontal face fitter

```
menpofit.aam.load balanced frontal face fitter()
```

Loads a frontal face patch-based AAM fitter that is a good compromise between model size, fitting time and fitting performance. The model returns 68 facial landmark points (the standard IBUG68 markup).

Note that the first time you invoke this function, menpofit will download the fitter from Menpo's server. The fitter will then be stored locally for future use.

The model is a PatchAAM trained using the following parameters:

Parameter	Value
diagonal	110
scales	(0.5, 1.0)
patch_shape	[(13, 13), (13, 13)]
holistic_features	menpo.feature.fast_dsift()
n_shape	[5, 20]
n_appearance	[30, 150]
lk_algorithm_cls	WibergInverseCompositional

It is also using the following sampling grid:

```
import numpy as np

patch_shape = (13, 13)
sampling_step = 4

sampling_grid = np.zeros(patch_shape, dtype=np.bool)
sampling_grid[::sampling_step, ::sampling_step] = True
sampling = [sampling_grid, sampling_grid]
```

Additionally, it is trained on LFPW trainset, HELEN trainset, IBUG and AFW datasets (3283 images in total), which are hosted in http://ibug.doc.ic.ac.uk/resources/facial-point-annotations/.

Returns fitter (LucasKanadeAAMFitter) – A pre-trained LucasKanadeAAMFitter based on a PatchAAM that performs facial landmark localization returning 68 points (iBUG68).

2.1.2 menpofit.aps

Active Pictorial Structures

APS is a model that utilises a Gaussian Markov Random Field (GMRF) for learning an appearance model with pairwise distributions based on a graph. It also has a parametric statitical shape model (either using PCA or GMRF), as well as a spring-like deformation prior term. The optimisation is performed using a weighted Gauss-Newton algorithm with fixed Jacobian and Hessian.

Generative APS

```
class menpofit.aps.GenerativeAPS (images,
                                                        group=None,
                                                                         appearance graph=None,
                                           shape_graph=None.
                                                                deformation graph=None,
                                           tic_features=<function no_op>, reference_shape=None,
                                           diagonal=None.
                                                           scales=(0.5,
                                                                         1.0), patch\_shape=(17,
                                           17),
                                                      patch_normalisation=<function</pre>
                                                                                         no\_op>,
                                           use_procrustes=True,
                                                                           precision_dtype=<class
                                           'numpy.float32'>,
                                                                     max_shape_components=None,
                                           n_appearance_components=None,
                                           can_be_incremented=False,
                                                                                   verbose=False,
                                           batch_size=None)
```

Bases: object

Class for training a multi-scale Generative Active Pictorial Structures model. Please see the references for a basic list of relevant papers.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the AAM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- appearance_graph (*list* of graphs or a single graph or None, optional) The graph to be used for the appearance *menpo.model.GMRFModel* training. It must be a *menpo.shape.UndirectedGraph*. If None, then a *menpo.model.PCAModel* is used instead.
- **shape_graph** (*list* of graphs or a single graph or None, optional) The graph to be used for the shape *menpo.model.GMRFModel* training. It must be a *menpo.shape.UndirectedGraph*. If None, then the shape model is built using *menpo.model.PCAModel*.
- **deformation_graph** (*list* of graphs or a single graph or None, optional) The graph to be used for the deformation *menpo.model.GMRFModel* training. It must be either a *menpo.shape.DirectedGraph* or a *menpo.shape.Tree*. If None, then the minimum spanning tree of the data is computed.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.

- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the APS. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- patch_normalisation (*list* of *callable* or a single *callable*, optional) The normalisation function to be applied on the extracted patches. If *list*, then it must have length equal to the number of scales. If a single patch normalization *callable*, then this is the one applied to all scales.
- **use_procrustes** (*bool*, optional) If True, then Generalized Procrustes Alignment is applied before building the deformation model.
- **precision_dtype** (*numpy.dtype*, optional) The data type of the appearance GMRF's precision matrix. For example, it can be set to *numpy.float32* for single precision or to *numpy.float64* for double precision. Even though the precision matrix is stored as a *scipy.sparse* matrix, this parameter has a big impact on the amount of memory required by the model.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- n_appearance_components (list of int or int or None, optional) The number of appearance components used for building the appearance menpo.shape.GMRFModel. If list, then it must have length equal to the number of scales. If a single int, then this is the one applied to all scales. If None, the covariance matrix of each edge is inverted using np.linalg.inv. If int, it is inverted using truncated SVD using the specified number of components.
- can_be_incremented (bool, optional) In case you intend to incrementally update the model in the future, then this flag must be set to True from the first place. Note that if True, the appearance and deformation menpo.shape.GMRFModel models will occupy double memory.
- verbose (bool, optional) If True, then the progress of building the APS will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

References

increment (images, group=None, batch_size=None, verbose=False)

Method that incrementally updates the APS model with a new batch of training images.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the APS. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.
- **verbose** (*bool*, optional) If True, then the progress of building the APS will be printed.

instance (*shape_weights=None*, *scale_index=-1*, *as_graph=False*) Generates an instance of the shape model.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- scale_index (int, optional) The scale to be used.
- as_graph (bool, optional) If True, then the instance will be returned as a menpo.shape.PointTree or a menpo.shape.PointDirectedGraph, depending on the type of the deformation graph.

random_instance (scale_index=-1, as_graph=False)
Generates a random instance of the APS.

Parameters

- scale_index (int, optional) The scale to be used.
- as_graph (bool, optional) If True, then the instance will be returned as a menpo.shape.PointTree or a menpo.shape.PointDirectedGraph, depending on the type of the deformation graph.

```
view deformation model(scale index=-
                                                  1.
                                                          n std=2.
                                                                        render colour bar=False,
                                                        image_view=True,
                                colour map='jet',
                                                                                  figure id=None,
                               new_figure=False, render_graph_lines=True, graph_line_colour='b',
                               graph line style='-', graph line width=1.0, ellipse line colour='r',
                                ellipse line style='-', ellipse line width=1.0, render markers=True,
                               marker style='o',
                                                     marker size=5,
                                                                          marker face colour='k',
                               marker_edge_colour='k',
                                                              marker\_edge\_width=1.0,
                               der_axes=False, axes_font_name='sans-serif', axes_font_size=10,
                                axes_font_style='normal',
                                                                       axes_font_weight='normal',
                                crop\_proportion=0.1, figure\_size=(7, 7))
```

Visualize the deformation model by plotting a Gaussian ellipsis per graph edge.

Parameters

- scale_index (int, optional) The scale to be used.
- n_std (*float*, optional) This defines the size of the ellipses in terms of number of standard deviations.

2.1. Deformable Models 89

- render_colour_bar (bool, optional) If True, then the ellipses will be coloured based on their normalized standard deviations and a colour bar will also appear on the side. If False, then all the ellipses will have the same colour.
- colour_map (str, optional) A valid Matplotlib colour map. For more info, please refer to matplotlib.cm.
- image_view (bool, optional) If True the ellipses will be rendered in the image coordinates system.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_graph_lines (bool, optional) Defines whether to plot the graph's edges.
- **graph_line_colour** (See Below, optional) The colour of the lines of the graph's edges. Example options:

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **graph_line_style** ({-, --, -., :}, optional) The style of the lines of the graph's edges.
- graph_line_width (*float*, optional) The width of the lines of the graph's edges.
- ellipse_line_colour (See Below, optional) The colour of the lines of the ellipses. Example options:

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- ellipse_line_style ({-, --, -., :}, optional) The style of the lines of the ellipses.
- ellipse_line_width (*float*, optional) The width of the lines of the ellipses.
- render_markers (bool, optional) If True, the centers of the ellipses will be rendered.
- marker_style (See Below, optional) The style of the centers of the ellipses. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the centers of the ellipses in points.
- marker_face_colour (See Below, optional) The face (filling) colour of the centers of the ellipses. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (See Below, optional) - The edge colour of the centers of the ellipses. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The edge width of the centers of the ellipses.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold,demibold, demi, bold, heavy, extra bold, black}
```

- **crop_proportion** (*float*, optional) The proportion to be left around the centers' pointcloud.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

property n_scales

Returns the number of scales.

Type int

Fitters

GaussNewtonAPSFitter

```
class menpofit.aps.GaussNewtonAPSFitter (aps, gn\_algorithm\_cls = < class 'menpofit.aps.algorithm.gn.Inverse'>, n\_shape = None, weight = 200.0, sampling = None)
```

Bases: APSFitter

A class for fitting an APS model with Gauss-Newton optimization.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided *max_iters*.

- aps (Generative APS or subclass) The trained model.
- gn_algorithm_cls (class, optional) The Gauss-Newton optimisation algorithm that will get applied. The possible algorithms are Inverse and Forward. Note that the

Forward algorithm is too slow. It is not recommended to be used for fitting an APS and is only included for comparison purposes.

- n_shape (int or float or list of those or None, optional) The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_shape_components during training.
- weight (float or list of float, optional) The weight between the appearance cost and the deformation cost. The provided value gets multiplied with the deformation cost. If float, then the provided value will be used for all scales. If list, then it should define a value per scale.
- **sampling** (*list* of *int* or *ndarray* or None) It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied. Note that depending on the model and the size of the appearance precision matrix, the sub-sampling may be impossible to be applied due to insufficient memory. This is because the sub-sampling of the appearance precision matrix involves converting it to *scipy.sparse.lil_matrix*, sub-sampling it and re-convert it back to *scipy.sparse.bsr_matrix*, which is a memory intensive procedure.
- **fit_from_bb** (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

Parameters

- **image** (*menpo.image.Image* or subclass) The image to be fitted.
- bounding_box (menpo.shape.PointDirectedGraph) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

fit_from_shape (image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)

Fits the multi-scale fitter to an image given an initial shape.

Parameters

• image (menpo.image.Image or subclass) – The image to be fitted.

- initial_shape (menpo.shape.PointCloud) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within an APSResult.

Parameters

- **image** (*menpo.image.Image* or *subclass*) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

property aps

The trained APS model.

Type GenerativeAPS or subclass

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type list of closure

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type list of int or float

Gauss-Newton Optimisation Algorithms

Inverse

class menpofit.aps.Inverse(aps_interface, eps=1e-05)

Bases: GaussNewton

Inverse Gauss-Newton algorithm for APS.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*) Execute the optimization algorithm.

Parameters

- **image** (*menpo.image.Image*) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

Returns fitting_result (APSAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance GMRF model.

Type menpo.model.GMRFModel

property deformation model

Returns the deformation GMRF model.

Type menpo.model.GMRFModel

property template

Returns the template (usually the mean appearance).

Type menpo.image.Image

property transform

Returns the motion model.

Type OrthoPDM

Forward

class menpofit.aps.**Forward**(*aps_interface*, *eps=1e-05*)

Bases: GaussNewton

Forward Gauss-Newton algorithm for APS.

Note: The Forward optimization is too slow. It is not recommended to be used for fitting an APS and is only included for comparison purposes. Use *Inverse* instead.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

Returns fitting_result (APSAlgorithmResult) – The parametric iterative fitting result.

property appearance_model

Returns the appearance GMRF model.

Type menpo.model.GMRFModel

property deformation_model

Returns the deformation GMRF model.

Type menpo.model.GMRFModel

property template

Returns the template (usually the mean appearance).

Type menpo.image.Image

property transform

Returns the motion model.

Type OrthoPDM

Fitting Result

APSResult

Bases: MultiScaleParametricIterativeResult

Class for storing the multi-scale iterative fitting result of an APS. It holds the shapes, shape parameters, appearance parameters and costs per iteration.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

Parameters

- results (list of APSAlgorithmResult) The list of optimization results per scale.
- **scales** (*list* or *tuple*) The *list* of scale values per scale (low to high).
- **affine_transforms** (*list* of *menpo.transform.Affine*) The list of affine transforms per scale that transform the shapes into the original image space.
- scale_transforms (list of menpo.shape.Scale) The list of scaling transforms per scale.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type list of ndarray

displacements_stats (stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be 'mean', 'median', 'min' or 'max'

errors (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='*line_width=2, render_markers=True,* marker_style='o', $marker_size=4$, marker_face_colour='b', marker_edge_colour='k', $marker_edge_width=1.0$, render axes=True, axes font name='sans-serif', axes font size=10, axes_font_weight='normal', axes_font_style='normal', $axes_x_limits=0.0$, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5) Plot of the cost function evolution at each fitting iteration.

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'}, optional) The style of the lines.
- line_width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (*int*, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (colour or None, optional) – The edge colour of the markers.If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.

• grid_line_width (*float*, optional) – The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True,
                         line colour='b', line style='-',
                                                          line width=2,
                                                                          render markers=True,
                         marker style='o',
                                                                        marker face colour='b',
                                                 marker size=4,
                         marker edge colour='k', marker edge width=1.0,
                                                                              render axes=True,
                         axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal',
                         axes font weight='normal',
                                                      axes_x_limits=0.0,
                                                                            axes y limits=None,
                         axes_x_ticks=None,
                                             axes_y_ticks=None, figure_size=(10,
                         der_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (float, optional) The width of the lines.
- render markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker edge width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes font name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes font size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_errors (compute_error=None, figure_id=None,
                                                            new_figure=False,
                                                                                render_lines=True,
                line colour='b',
                                      line_style='-',
                                                         line\_width=2,
                                                                             render_markers=True,
                marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
                marker\_edge\_width=1.0,
                                               render axes=True,
                                                                       axes_font_name='sans-serif',
                                         axes_font_style='normal',
                                                                        axes_font_weight='normal',
                axes\_font\_size=10,
                axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None,
                figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the error evolution at each fitting iteration.
```

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -:}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed_initial_error (compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed initial shapes[0]*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view (figure id=None, new figure=False, render image=True, render final shape=True, render_initial_shape=False, subplots enabled=True, $render_gt_shape=False,$ channels=None, interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt marker face colour='y', gt marker edge colour='k', gt line colour='y', render lines=True, line style='-', line width=2, render markers=True. marker style='o', marker size=4, render numbering=False, numbers horizontal align='center', $marker\ edge\ width=1.0,$ numbers vertical align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title="', legend_font_name='sans-serif', legend_font_style='normal', legend_font_weight='normal', legend font size=10, legend_marker_scale=None, end location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, *legend_border=True*, legend_border_padding=None, *legend_shadow=False*, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape,

ground truth shape and the image, id they were provided.

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (bool, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (bool, optional) If True and the ground truth shape exists, then it gets rendered.
- subplots_enabled (bool, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- channels (int or list of int or all or None) If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- cmap_name (str, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (float, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.

• **final_marker_face_colour** (See Below, optional) - The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) – The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour(See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- line_style (str or list of str, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (*bool*, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- **legend_font_size** (*int*, optional) The font size of the legend.
- legend_font_weight (See Below, optional)—The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- legend_border_axes_pad (*float*, optional) The pad between the axes and legend border
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.

- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- $figure_size$ ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view iterations (figure id=None, new figure=False, iters=None, render image=True, subplots enabled=False, channels=None. interpolation='bilinear', cmap name=None, render lines=True, alpha=1.0, masked=True. line_style='-', line width=2, line colour=None, render_markers=True, marker edge colour=None, marker face colour=None, marker style='o', $marker\ size=4$, marker_edge_width=1.0, render numbering=False, numbers horizontal align='center'. numbers vertical align='bottom'. numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=None, axes y limits=None, $axes \ x \ ticks=None,$ axes y ticks=None, figure size=(7,7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (str (See Below), optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:}
```

- line_width (*float* or *list* of *float*, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_colour (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_width (*float* or *list* of *float*, optional) – The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different

value per iteration shape.

- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (str (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend title (str, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property appearance_costs

Returns a list with the appearance cost per iteration. It returns None if the costs are not computed.

Type *list* of *float* or None

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property deformation costs

Returns a list with the deformation cost per iteration. It returns None if the costs are not computed.

Type *list* of *float* or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type *menpo.shape.PointCloud* or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property n_iters_per_scale

Returns the number of iterations per scale of the fitting process.

Type list of int

property n_scales

Returns the number of scales used during the fitting process.

Type int

property reconstructed_initial_shapes

Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

Type *list* of *menpo.shape.PointCloud*

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type list of (n_params,) ndarray

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists) and *final_shape*.

Type *list* of *menpo.shape.PointCloud*

APSAlgorithmResult

Bases: ParametricIterativeResult

Class for storing the iterative result of an APS optimisation algorithm.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of shapes per iteration. The first and last members correspond to the initial and final shapes, respectively.
- **shape_parameters** (*list* of (n_shape_parameters,) *ndarray*) The *list* of shape parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.
- initial_shape (*menpo.shape.PointCloud* or None, optional) The initial shape from which the fitting process started. If None, then no initial shape is assigned.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

- appearance_costs (*list* of *float* or None, optional) The *list* of the appearance cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.
- deformation_costs (list of float or None, optional) The list of the deformation cost
 per iteration. If None, then it is assumed that the cost function cannot be computed for the
 specific algorithm.
- **costs** (*list* of *float* or None, optional) The *list* of the total cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

displacements_stats (stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be 'mean', 'median', 'min' or 'max'

errors (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

• ValueError – Initial shape has not been set, so the initial error cannot be computed

ValueError – Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-
                                                              marker_style='o',
                    line width=2,
                                   render_markers=True,
                                                                                  marker\ size=4,
               marker_face_colour='b',
                                            marker_edge_colour='k',
                                                                         marker\ edge\ width=1.0,
               render axes=True,
                                          axes font name='sans-serif',
                                                                               axes font size=10,
               axes font style='normal',
                                              axes_font_weight='normal',
                                                                                axes x limits=0.0,
               axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(10, 6),
               render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the cost function evolution at each fitting iteration.
```

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'}, optional) The style of the lines.
- line width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (*colour* or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (float, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes y ticks (list or tuple or None, optional) The ticks of the y axis.
- figure size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_errors (compute_error=None,
                                       figure_id=None,
                                                          new_figure=False,
                                                                                render_lines=True,
                line colour='b',
                                      line style='-',
                                                         line width=2,
                                                                             render markers=True,
                marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
                marker\ edge\ width=1.0,
                                              render axes=True,
                                                                      axes font name='sans-serif',
                axes\_font\_size=10,
                                         axes_font_style='normal',
                                                                       axes_font_weight='normal',
                axes x limits=0.0, axes y limits=None, axes x ticks=None, axes y ticks=None,
                figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the error evolution at each fitting iteration.
```

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• marker_size (int, optional) – The size of the markers in points.

• marker_face_colour (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style({'-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed initial error (compute error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed_initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

render final shape=True, **view** (figure id=None, new figure=False, render image=True, render_gt_shape=False, render initial shape=False, *subplots enabled=True*, nels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, *line_style='-', line_width=2,* marker_style='o', marker_size=4, render_markers=True, $marker_edge_width=1.0$, render_numbering=False, numbers_horizontal_align='center', numbers_font_name='sans-serif', numbers_vertical_align='bottom', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', legend font size=10, legend font weight='normal', legend marker scale=None, end location=2, legend bbox to anchor=(1.05,1.0), legend border axes pad=None, $legend_n_columns=1$, *legend_horizontal_spacing=None*, legend_vertical_spacing=None, legend border=True, legend border padding=None, legend shadow=False, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=None, axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(7,7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape,

Parameters

• **figure_id** (*object*, optional) – The id of the figure to be used.

ground truth shape and the image, id they were provided.

- new_figure (bool, optional) If True, a new figure is created.
- render_image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.

- render_initial_shape (bool, optional) If True and the initial fitting shape exists, then it gets rendered.
- **render_gt_shape** (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- **final_marker_face_colour** (See Below, optional) The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) - The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **line_style** (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (*float* or *list* of *float*, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_markers (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (*bool*, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- **legend_font_size** (*int*, optional) The font size of the legend.
- legend_font_weight (See Below, optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **legend_marker_scale** (*float*, optional) – The relative size of the legend markers with respect to the original

• **legend_location** (*int*, optional) – The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes y ticks (list or tuple or None, optional) The ticks of the y axis.
- figure size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view_iterations (figure_id=None, new_figure=False, iters=None, render_image=True, subplots enabled=False, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line style='-'. line width=2, line colour=None, render markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', $marker\ size=4$, $marker_edge_width=1.0$, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers font colour='k', render legend=True, legend title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend bbox to anchor=(1.05, 1.0), end border axes pad=None, $legend \ n \ columns=1$, legend horizontal spacing=None, legend vertical spacing=None, legend border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes x limits=None,axes y limits=None, $axes \ x \ ticks=None,$ axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the

125

image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• interpolation (str (See Below), optional) — The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_style (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -., :}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_colour** (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_markers (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (*str* (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (*str* (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- **legend title** (*str*, optional) The title of the legend.
- **legend_font_name** (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- **legend_bbox_to_anchor** ((*float*, *float*) *tuple*, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend n columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

{serif, sans-serif, cursive, fantasy, monospace}

- axes_font_size (*int*, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style
 of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property appearance_costs

Returns a *list* with the appearance cost per iteration. It returns None if the costs are not computed.

Type *list* of *float* or None

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property deformation_costs

Returns a *list* with the deformation cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type *menpo.shape.Image* or *subclass* or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property reconstructed_initial_shape

Returns the initial shape's reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type menpo.shape.PointCloud

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *reconstructed_initial_shape* and *final_shape*.

Type *list* of (n_params,) *ndarray*

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists), *reconstructed_initial_shape* and *final_shape*.

Type *list* of *menpo.shape.PointCloud*

2.1.3 menpofit.atm

Active Template Model

ATM is a generative model that performs deformable alignment between a template image and a test image with respect to a statistical parametric shape model. MenpoFit has several ATMs which differ in the manner that they compute the warp (thus represent the appearance features).

ATM

Bases: object

Class for training a multi-scale holistic Active Template Model.

Parameters

- template (menpo.image.Image) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.

- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- **reference_shape** (*menpo.shape.PointCloud* or None, optional) The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape
 so that the diagonal of its bounding box matches the provided value. In other words, this
 parameter controls the size of the model at the highest scale. If None, then the reference
 shape does not get rescaled.
- scales (float or tuple of float, optional) The scale value of each scale. They must
 provided in ascending order, i.e. from lowest to highest scale. If float, then a single scale is
 assumed.
- transform (subclass of DL and DX, optional) A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (*bool*, optional) If True, then the progress of building the ATM will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

References

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface.

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

131

Returns fitter_interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained ATM with a new set of training shapes and a new template.

Parameters

- template (menpo.image.Image) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- shape_forgetting_factor([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the ATM will be printed.
- **batch_size** (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

```
instance (shape_weights=None, scale_index=- 1)
```

Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- **scale_index** (*int*, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

```
random_instance (scale_index=- 1)
```

Generates a random instance of the ATM.

Parameters scale_index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

property n_scales

Returns the number of scales.

Type int

HolisticATM

menpofit.atm.HolisticATM alias of ATM

MaskedATM

Bases: ATM

Class for training a multi-scale patch-based Masked Active Template Model. The appearance of this model is formulated by simply masking an image with a patch-based mask.

Parameters

- **template** (*menpo.image.Image*) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- patch_shape ((int, int), optional) The size of the patches of the mask that is used to sample the appearance vectors.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (bool, optional) If True, then the progress of building the ATM will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface.

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter_interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained ATM with a new set of training shapes and a new template.

Parameters

- **template** (*menpo.image.Image*) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- shape_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the ATM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

```
instance (shape_weights=None, scale_index=- 1)
```

Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- scale_index (int, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

```
random_instance (scale_index=- 1)
```

Generates a random instance of the ATM.

Parameters scale_index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

property n_scales

Returns the number of scales.

Type int

LinearATM

Bases: ATM

Class for training a multi-scale Linear Active Template Model.

Parameters

- **template** (*menpo.image.Image*) The template image.
- shapes (list of menpo.shape.PointCloud) The list of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape
 so that the diagonal of its bounding box matches the provided value. In other words, this
 parameter controls the size of the model at the highest scale. If None, then the reference
 shape does not get rescaled.
- scales (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- transform (subclass of DL and DX, optional) A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (*bool*, optional) If True, then the progress of building the ATM will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface.

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter_interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained ATM with a new set of training shapes and a new template.

Parameters

- **template** (*menpo.image.Image*) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- shape_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the ATM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

```
instance (shape_weights=None, scale_index=- 1)
```

Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- scale_index (int, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

```
random_instance (scale_index=- 1)
```

Generates a random instance of the ATM.

Parameters scale_index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

property n_scales

Returns the number of scales.

Type int

LinearMaskedATM

Bases: ATM

Class for training a multi-scale Linear Masked Active Template Model.

Parameters

- template (menpo.image.Image) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (*int* or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches of the mask that is used to extract the appearance vectors. If a list is provided, then it defines a patch shape per scale.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- verbose (bool, optional) If True, then the progress of building the ATM will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface.

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter_interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained ATM with a new set of training shapes and a new template.

Parameters

- **template** (*menpo.image.Image*) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- shape_forgetting_factor ([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the ATM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

```
instance (shape_weights=None, scale_index=- 1)
```

Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- scale_index (int, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

```
random_instance (scale_index=- 1)
```

Generates a random instance of the ATM.

Parameters scale_index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

property n_scales

Returns the number of scales.

Type int

PatchATM

Bases: ATM

Class for training a multi-scale Patch-Based Active Template Model.

Parameters

- **template** (*menpo.image.Image*) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo, feature for a list of potential features.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the ATM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (*bool*, optional) If True, then the progress of building the ATM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

build_fitter_interfaces (sampling)

Method that builds the correct Lucas-Kanade fitting interface.

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly

defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter_interfaces (*list*) – The *list* of Lucas-Kanade interface per scale.

Method to increment the trained ATM with a new set of training shapes and a new template.

Parameters

- template (menpo.image.Image) The template image.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used to train the ATM. If None and the *template* only has a single landmark group, then that is the one that will be used.
- shape_forgetting_factor([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the ATM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

instance (shape_weights=None, scale_index=- 1)

Generates a novel ATM instance given a set of shape weights. If no weights are provided, the mean ATM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- scale_index (int, optional) The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

random instance(scale index=-1)

Generates a random instance of the ATM.

Parameters scale index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The ATM instance.

property n_scales

Returns the number of scales.

Type int

Fitter

LucasKanadeATMFitter

class menpofit.atm.LucasKanadeATMFitter (atm, $lk_algorithm_cls = < class$ 'menpofit.atm.algorithm.InverseCompositional'>, $n \ shape = None, sampling = None$)

Bases: MultiScaleParametricFitter

Class for defining an ATM fitter using the Lucas-Kanade optimization.

Parameters

- atm (ATM or subclass) The trained ATM model.
- **lk_algorithm_cls** (*class*, optional) The Lukas-Kanade optimisation algorithm that will get applied. The possible algorithms are:

Class	Warp Direction	Warp Update
ForwardCompositional	Forward	Compositional
InverseCompositional	Inverse	

- n_shape (int or float or list of those or None, optional) The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_shape_components during training.
- **sampling** (*list* of *int* or *ndarray* or None) It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

fit_from_bb (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

- image (menpo.image.Image or subclass) The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a MultiScaleParametricIterativeResult.

Parameters

- image (menpo.image.Image or subclass) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (list of menpo.image.MaskedImage or ndarray) – The warped images.

property atm

The trained ATM model.

Type ATM or subclass

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type list of closure

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

Lucas-Kanade Optimisation Algorithms

ForwardCompositional

class menpofit.atm.ForwardCompositional(atm_interface, eps=1e-05)

Bases: Compositional

Forward Compositional (FC) Gauss-Newton algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (ParametricIterativeResult) – The parametric iterative fitting result.

property template

Returns the template of the ATM.

Type menpo.image.Image or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

InverseCompositional

class menpofit.atm.InverseCompositional(atm_interface, eps=1e-05)

Bases: Compositional

Inverse Compositional (IC) Gauss-Newton algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (ParametricIterativeResult) – The parametric iterative fitting result.

property template

Returns the template of the ATM.

Type menpo.image.Image or subclass

property transform

Returns the model driven differential transform object of the AAM, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.

Type subclass of DL and DX

2.1.4 menpofit.clm

Constrained Local Model

Deformable model that consists of a generative parametric shape model and discriminatively trained experts per part.

CLM

Bases: object

Class for training a multi-scale holistic Constrained Local Model. Please see the references for a basic list of relevant papers.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the CLM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- **reference_shape** (*menpo.shape.PointCloud* or None, optional) The reference shape that will be used for building the CLM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- patch_normalisation (*callable*, optional) The normalisation function to be applied on the extracted patches.
- **context_shape** ((*int*, *int*) or *list* of (*int*, *int*), optional) The context shape for the convolution. If a *list* is provided, then it defines a context shape per scale.
- **cosine_mask** (*bool*, optional) If True, then a cosine mask (Hanning function) will be applied on the extracted patches.
- sample_offsets ((n_offsets, n_dims) ndarray or None, optional) The offsets to sample from within a patch. So (0, 0) is the centre of the patch (no offset) and (1, 0) would be sampling the patch from 1 pixel up the first axis away from the centre. If None, then no offsets are applied.
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.

- **expert_ensemble_cls** (*subclass* of ExpertEnsemble, optional) The class to be used for training the ensemble of experts. The most common choice is *CorrelationFilterExpertEnsemble*.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **verbose** (bool, optional) If True, then the progress of building the CLM will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

References

increment (*images*, *group=None*, *shape_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*) Method to increment the trained CLM with a new set of training images.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the CLM. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- shape_forgetting_factor([0.0, 1.0] float, optional) Forgetting factor that weights the relative contribution of new samples vs old samples for the shape model. If 1.0, all samples are weighted equally and, hence, the result is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples.
- **verbose** (*bool*, optional) If True, then the progress of building the CLM will be printed.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

shape_instance (shape_weights=None, scale_index=- 1)

Generates a novel shape instance given a set of shape weights. If no weights are provided, the mean shape is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- scale_index (int, optional) The scale to be used.

Returns instance (*menpo.shape.PointCloud*) – The shape instance.

property n_scales

Returns the number of scales.

Type int

Fitter

GradientDescentCLMFitter

Bases: CLMFitter

Class for defining an CLM fitter using gradient descent optimization.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided *max_iters*.

Parameters

- clm (CLM or subclass) The trained CLM model.
- gd_algorithm_cls (class, optional) The gradient descent optimisation algorithm that will get applied. The possible options are RegularisedLandmarkMeanShift and ActiveShapeModel.
- n_shape (int or float or list of those or None, optional) The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_shape_components during training.
- **fit_from_bb** (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

- image (menpo.image.Image or subclass) The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

property clm

The trained CLM model.

Type CLM or subclass

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *closure*

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

Gradient Descent Optimisation Algorithms

ActiveShapeModel

Bases: GradientDescentCLMAlgorithm

Active Shape Model (ASM) algorithm.

Parameters

- **expert_ensemble** (*subclass* of ExpertEnsemble) The ensemble of experts object, e.g. *CorrelationFilterExpertEnsemble*.
- **shape_model** (*subclass* of *PDM*, optional) The shape model object, e.g. OrthoPDM.
- gaussian covariance (int or float, optional) The covariance of the Gaussian kernel.
- **eps** (*float*, optional) Value for checking the convergence of the optimization.

References

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (ParametricIterativeResult) - The parametric iterative fitting result.

RegularisedLandmarkMeanShift

Bases: GradientDescentCLMAlgorithm

Regularized Landmark Mean-Shift (RLMS) algorithm.

Parameters

- **expert_ensemble** (*subclass* of ExpertEnsemble) The ensemble of experts object, e.g. *CorrelationFilterExpertEnsemble*.
- **shape_model** (*subclass* of *PDM*, optional) The shape model object, e.g. OrthoPDM.
- **kernel_covariance** (*int* or *float*, optional) The covariance of the kernel.
- **eps** (*float*, optional) Value for checking the convergence of the optimization.

References

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *map_inference=False*) Execute the optimization algorithm.

Parameters

- **image** (*menpo.image.Image*) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.
- map_inference (*bool*, optional) If True, then the solution will be given after performing MAP inference.

Returns fitting_result (ParametricIterativeResult) - The parametric iterative fitting result.

Experts Ensemble

Algorithms for learning an ensemble of discriminative experts.

CorrelationFilterExpertEnsemble

```
class menpofit.clm.CorrelationFilterExpertEnsemble(images,
                                                                                            shapes,
                                                                    icf_cls=<class
                                                                                              'men-
                                                                    pofit.clm.expert.base.IncrementalCorrelationFilterThinWra
                                                                    patch\_shape=(17,
                                                                                                17),
                                                                                               34),
                                                                    context shape=(34,
                                                                    response_covariance=3,
                                                                    patch_normalisation=functools.partial(<function
                                                                    normalize_norm>,
                                                                    mode='per_channel',
                                                                                                er-
                                                                    ror on divide by zero=False),
                                                                    cosine mask=True,
                                                                    ple offsets=None,
                                                                                          prefix="
```

verbose=False)

151

Bases: ConvolutionBasedExpertEnsemble

Class for defining an ensemble of correlation filter experts.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of training shapes that correspond to the images.
- icf_cls (class, optional) The incremental correlation filter class. For example IncrementalCorrelationFilterThinWrapper.
- patch_shape ((int, int), optional) The shape of the patches that will be extracted around the landmarks. Those patches are used to train the experts.
- context_shape ((int, int), optional) The context shape for the convolution.
- **response_covariance** (*int*, optional) The covariance of the generated Gaussian response.
- patch_normalisation (*callable*, optional) A normalisation function that will be applied on the extracted patches.
- **cosine_mask** (*bool*, optional) If True, then a cosine mask (Hanning function) will be applied on the extracted patches.
- **sample_offsets** ((n_offsets, n_dims) *ndarray* or None, optional) The offsets to sample from within a patch. So (0, 0) is the centre of the patch (no offset) and (1, 0) would be sampling the patch from 1 pixel up the first axis away from the centre. If None, then no offsets are applied.
- **prefix** (*str*, optional) The prefix of the printed progress information.
- **verbose** (*bool*, optional) If True, then information will be printed regarding the training progress.

increment (images, shapes, prefix=", verbose=False)

Increments the learned ensemble of convolution-based experts given a new set of training data.

Parameters

2.1. Deformable Models

- images (list of menpo.image.Image) The list of training images.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of training shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix of the printed training progress.
- verbose (bool, optional) If True, then information about the training progress will be printed.

predict_probability(image, shape)

Method for predicting the probability map of the response experts on a given image. Note that the provided shape must have the same number of points as the number of experts.

Parameters

- image (menpo.image.Image or subclass) The test image.
- **shape** (*menpo.shape.PointCloud*) The shape that corresponds to the image from which the patches will be extracted.

Returns probability_map ((n_experts, 1, height, width) *ndarray*) – The probability map of the response of each expert.

predict_response (image, shape)

Method for predicting the response of the experts on a given image. Note that the provided shape must have the same number of points as the number of experts.

Parameters

- image (menpo.image.Image or subclass) The test image.
- **shape** (*menpo.shape.PointCloud*) The shape that corresponds to the image from which the patches will be extracted.

Returns response ((n_experts, 1, height, width) *ndarray*) – The response of each expert.

property frequency_filter_images

Returns a *list* of *n_experts* filter images on the frequency domain.

Type *list* of *menpo.image.Image*

property n_experts

Returns the number of experts.

Type int

property n_sample_offsets

Returns the number of offsets that are sampled within a patch.

Type int

property padded_size

Returns the convolution pad size, i.e. floor (1.5 * patch_shape - 1).

Type (int, int)

property search_shape

Returns the search shape (*patch_shape*).

Type (int, int)

property spatial_filter_images

Returns a *list* of *n_experts* filter images on the spatial domain.

Type list of menpo.image.Image

Experts

Discriminative experts

IncrementalCorrelationFilterThinWrapper

Bases: object

Wrapper class for defining an Incremental Correlation Filter.

Parameters

• cf_callable (callable, optional) – The correlation filter function. Possible options are:

Class	Method	
mccf	Multi-Channel Correlation Filter	
mosse	Minimum Output Sum of Squared Errors Filter	

• icf_callable (callable, optional) – The incremental correlation filter function. Possible options are:

Class	Method	
imccf	Incremental Multi-Channel Correlation Filter	
imosse	Incremental Minimum Output Sum of Squared Errors Filter	

$increment(A, B, n_x, Z, t)$

Method that trains the correlation filter.

Parameters

- A ((N,) ndarray) The current auto-correlation array, where N = (patch_h+response_h-1) * (patch_w+response_w-1) * n channels
- B ((N, N) ndarray) The current cross-correlation array, where N = (patch_h+response_h-1) * (patch_w+response_w-1) * n_channels
- **n_x** (*int*) The current number of images.
- **Z** (*list* or (n_images, n_channels, patch_h, patch_w) *ndarray*) The training images (patches). If *list*, then it consists of *n_images* (n_channels, patch_h, patch_w) *ndarray* members.
- t((1, response_h, response_w) *ndarray*) The desired response.

Returns

• correlation_filter((n_channels, response_h, response_w) ndarray) - The learned correlation filter.

- auto_correlation ((N,) ndarray) The auto-correlation array, where N = (patch_h+response_h-1) * (patch_w+response_w-1) * n_channels
- cross_correlation ((N, N) ndarray) The cross-correlation array, where N = (patch_h+response_h-1) * (patch_w+response_w-1) * n_channels

train(X, t)

Method that trains the correlation filter.

Parameters

- **X** (*list* or (n_images, n_channels, patch_h, patch_w) *ndarray*) The training images (patches). If *list*, then it consists of *n_images* (n_channels, patch_h, patch_w) *ndarray* members.
- t((1, response_h, response_w) *ndarray*) The desired response.

Returns

- correlation_filter((n_channels, response_h, response_w) ndarray) The learned correlation filter.
- auto_correlation ((N,) ndarray) The auto-correlation array, where N = (patch_h+response_h-1) * (patch_w+response_w-1) * n_channels
- cross_correlation ((N, N) ndarray) The cross-correlation array, where N = (patch_h+response_h-1) * (patch_w+response_w-1) * n channels

2.1.5 menpofit.unified_aam_clm

Unified Active Appearance Model and Constrained Local Model

This method combines a holistic AAM with a part-based CLM under a unified optimisation problem.

UnifiedAAMCLM

```
class menpofit.unified_aam_clm.base.UnifiedAAMCLM(images,
                                                                                              holis-
                                                                              group=None,
                                                                   tic features=<function
                                                                                           no op>,
                                                                   reference_shape=None,
                                                                                              diag-
                                                                   onal=None,
                                                                                scales=(0.5,
                                                                                               1.0),
                                                                   expert_ensemble_cls=<class 'men-
                                                                  pofit.clm.expert.ensemble.CorrelationFilterExpertEnsemble'
                                                                   patch shape=(17,
                                                                                                17),
                                                                   context shape=(34,
                                                                                                34).
                                                                   sample_offsets=None,
                                                                   transform=<class
                                                                                              'men-
                                                                   pofit.transform.piecewiseaffine.DifferentiablePiecewiseAffine
                                                                   shape model cls=<class
                                                                                              'men-
                                                                  pofit.modelinstance.OrthoPDM'>,
                                                                   max_shape_components=None,
                                                                   max_appearance_components=None,
                                                                   sigma=None,
                                                                                       boundary=3,
                                                                   response_covariance=2,
                                                                   patch_normalisation=<function
                                                                   no_op>, cosine_mask=True, ver-
```

bose = False)

Bases: object

Class for training a multi-scale unified holistic AAM and CLM as presented in [1]. Please see the references for AAMs and CLMs in their respective base classes.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that will be used to train the model. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for building the AAM. The purpose of the reference shape is to normalise the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- expert_ensemble_cls (subclass of ExpertEnsemble, optional) The class to be used for training the ensemble of experts. The most common choice is

CorrelationFilterExpertEnsemble.

- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- **context_shape** ((*int*, *int*) or *list* of (*int*, *int*), optional) The context shape for the convolution. If a *list* is provided, then it defines a context shape per scale.
- sample_offsets ((n_offsets, n_dims) ndarray or None, optional) The sample_offsets to sample from within a patch. So (0, 0) is the centre of the patch (no offset) and (1, 0) would be sampling the patch from 1 pixel up the first axis away from the centre. If None, then no sample_offsets are applied.
- transform (subclass of DL and DX, optional) A differential warp transform object, e.g. DifferentiablePiecewiseAffine or DifferentiableThinPlateSplines.
- **shape_model_cls** (*subclass* of *OrthoPDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- max_shape_components (int, float, list of those or None, optional) The number of shape components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- max_appearance_components (int, float, list of those or None, optional) The number of appearance components to keep. If int, then it sets the exact number of components. If float, then it defines the variance percentage that will be kept. If list, then it should define a value per scale. If a single number, then this will be applied to all scales. If None, then all the components are kept. Note that the unused components will be permanently trimmed.
- **sigma** (*float* or None, optional) If not None, the input images are smoothed with an isotropic Gaussian filter with the specified standard deviation.
- **boundary** (*int*, optional) The number of pixels to be left as a safe margin on the boundaries of the reference frame (has potential effects on the gradient computation).
- **response_covariance** (*int*, optional) The covariance of the generated Gaussian response.
- patch_normalisation (*callable*, optional) The normalisation function to be applied on the extracted patches.
- **cosine_mask** (*bool*, optional) If True, then a cosine mask (Hanning function) will be applied on the extracted patches.
- **verbose** (*bool*, optional) If True, then the progress of building the model will be printed.

References

appearance_reconstructions (appearance_parameters, n_iters_per_scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a *UnifiedAAMCLMResult* object, in order to generate the appearance reconstructions of a fitting procedure.

- appearance_parameters (list of (n_params,) ndarray) A set of appearance parameters per fitting iteration. It can be retrieved as a property of an UnifiedAAMCLMResult object.
- n_iters_per_scale (*list* of *int*) The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a UnifiedAAMCLMResult object.

Returns appearance_reconstructions (*list* of *menpo.image.Image*) – *List* of the appearance reconstructions that correspond to the provided parameters.

build_fitter_interfaces (sampling)

Method that builds the correct fitting interface for a UnifiedAAMCLMFitter.

Parameters sampling (*list* of *int* or *ndarray* or None) – It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

Returns fitter_interfaces (*list*) – The *list* of fitting interfaces per scale.

instance(shape weights=None, appearance weights=None, scale index=-1)

Generates a novel instance of the AAM part of the model given a set of shape and appearance weights. If no weights are provided, then the mean AAM instance is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- appearance_weights ((n_weights,) ndarray or list or None, optional) The weights of the appearance model that will be used to create a novel appearance instance. If None, the weights are assumed to be zero, thus the mean appearance is used.
- scale_index (int, optional) The scale to be used.

Returns image (menpo.image.Image) – The AAM instance.

random_instance (scale_index=- 1)

Generates a random instance of the AAM part of the model.

Parameters scale_index (int, optional) – The scale to be used.

Returns image (*menpo.image.Image*) – The AAM instance.

```
shape_instance (shape_weights=None, scale_index=- 1)
```

Generates a novel shape instance given a set of shape weights. If no weights are provided, the mean shape is returned.

Parameters

- **shape_weights** ((n_weights,) *ndarray* or *list* or None, optional) The weights of the shape model that will be used to create a novel shape instance. If None, the weights are assumed to be zero, thus the mean shape is used.
- scale_index (int, optional) The scale to be used.

Returns instance (*menpo.shape.PointCloud*) – The shape instance.

property n_scales

Returns the number of scales.

Type int

2.1. Deformable Models

Fitter

The model optimised with a combination of Lucas-Kanade and Regularised Landmark Mean Shift algorithms.

UnifiedAAMCLMFitter

Bases: MultiScaleParametricFitter

Class defining a Unified AAM - CLM fitter.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided *max_iters*.

- unified_aam_clm (UnifiedAAMCLM or subclass) The trained unified AAM-CLM model.
- **algorithm_cls** (*class*, optional) The unified optimisation algorithm that will get applied. The possible algorithms are:

Class	Method
ProjectOutRegularisedLandmarkMeanShift	Project-Out IC + RLMS
AlternatingRegularisedLandmarkMeanShift	Alternating IC + RLMS

- n_shape (int or float or list of those or None, optional) The number of shape components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max shape components during training.
- n_appearance (int or float or list of those or None, optional) The number of appearance components that will be used. If int, then it defines the exact number of active components. If float, then it defines the percentage of variance to keep. If int or float, then the provided value will be applied for all scales. If list, then it defines a value per scale. If None, then all the available components will be used. Note that this simply sets the active components without trimming the unused ones. Also, the available components may have already been trimmed to max_appearance_components during training.
- **sampling** (*list* of *int* or *ndarray* or None) It defines a sampling mask per scale. If *int*, then it defines the sub-sampling step of the sampling mask. If *ndarray*, then it explicitly defines the sampling mask. If None, then no sub-sampling is applied.

appearance_reconstructions (appearance_parameters, n_iters_per_scale)

Method that generates the appearance reconstructions given a set of appearance parameters. This is to be combined with a <code>UnifiedAAMCLMResult</code> object, in order to generate the appearance reconstructions of a fitting procedure.

Parameters

- appearance_parameters (list of (n_params,) ndarray) A set of appearance parameters per fitting iteration. It can be retrieved as a property of an UnifiedAAMCLMResult object.
- n_iters_per_scale (*list* of *int*) The number of iterations per scale. This is necessary in order to figure out which appearance parameters correspond to the model of each scale. It can be retrieved as a property of a UnifiedAAMCLMResult object.

Returns appearance_reconstructions (*list* of *menpo.image.Image*) – *List* of the appearance reconstructions that correspond to the provided parameters.

fit_from_bb (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- bounding_box (menpo.shape.PointDirectedGraph) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Fits the multi-scale fitter to an image given an initial shape.

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.

- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within an <code>UnifiedAAMCLMResult</code>.

Parameters

- image (menpo.image.Image or subclass) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type list of closure

property n scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property response_covariance

Returns the covariance value of the desired Gaussian response used to train the ensemble of experts.

Type int

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type list of int or float

property unified_aam_clm

The trained unified AAM-CLM model.

Type UnifiedAAMCLM or subclass

Optimisation Algorithms

AlternatingRegularisedLandmarkMeanShift

Bases: UnifiedAlgorithm

Alternating Inverse Compositional + Regularized Landmark Mean Shift

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *prior=False*, *a=0.5*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **prior** (*bool*, optional) If True, use a Gaussian priors over the latent shape and appearance spaces. see the reference [1] section 3.1.1 for details.
- a (*float*, optional) Ratio of the image noise variance and the shape noise variance. See [1] section 5 equations (25) & (26) and footnote 6.

Returns fitting_result (UnifiedAAMCLMAlgorithmResult) – The parametric iterative fitting result.

References

2.1. Deformable Models 161

ProjectOutRegularisedLandmarkMeanShift

Bases: UnifiedAlgorithm

Project-Out Inverse Compositional + Regularized Landmark Mean Shift

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*, *prior=False*, *a=0.5*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **prior** (*bool*, optional) If True, use a Gaussian priors over the latent shape and appearance spaces. see the reference [1] section 3.1.1 for details.
- **a** (*float*, optional) Ratio of the image noise variance and the shape noise variance. See [1] section 5 equations (25) & (26) and footnote 6.

Returns fitting_result (UnifiedAAMCLMAlgorithmResult) – The parametric iterative fitting result.

References

Fitting Result

UnifiedAAMCLMResult

Bases: MultiScaleParametricIterativeResult

Class for storing the multi-scale iterative fitting result of a Unified AAM-CLM. It holds the shapes, shape parameters, appearance parameters and costs per iteration.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

Parameters

- results (*list* of *UnifiedAAMCLMAlgorithmResult*) The *list* of optimization results per scale.
- **scales** (*list* or *tuple*) The *list* of scale values per scale (low to high).
- **affine_transforms** (*list* of *menpo.transform.Affine*) The list of affine transforms per scale that transform the shapes into the original image space.
- scale_transforms (list of menpo.shape.Scale) The list of scaling transforms per scale.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

```
displacements_stats (stat_type='mean')
```

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError - type must be 'mean', 'median', 'min' or 'max'

errors (compute error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

2.1. Deformable Models 163

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='line width=2, render markers=True, marker style='o', marker size=4, marker_face_colour='b', marker_edge_colour='k', $marker_edge_width=1.0,$ render_axes=True, axes_font_name='sans-serif', $axes_font_size=10$, axes_font_weight='normal', axes_font_style='normal', $axes_x_limits=0.0$, axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5) Plot of the cost function evolution at each fitting iteration.

- **figure_id** (*object*, optional) The id of the figure to be used.
- **new_figure** (*bool*, optional) If True, a new figure is created.
- render lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line style ({'-', '--', '-.', ':'}, optional) The style of the lines.
- line_width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.

• marker_style (marker, optional) – The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers.If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (*int*, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.

- grid_line_style({'-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -:}
```

- line width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker edge width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes font name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes font size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_errors (compute_error=None, figure_id=None,
                                                            new_figure=False,
                                                                                 render_lines=True,
                line colour='b',
                                      line_style='-',
                                                         line\_width=2,
                                                                             render_markers=True,
                marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
                marker\_edge\_width=1.0,
                                               render axes=True,
                                                                       axes_font_name='sans-serif',
                                         axes_font_style='normal',
                                                                        axes_font_weight='normal',
                axes\_font\_size=10,
                axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None,
                figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the error evolution at each fitting iteration.
```

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (float, optional) The width of the lines.
- render markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed_initial_error (compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed_initial_shapes[0]*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view (figure id=None, new figure=False, render image=True, render final shape=True, render_initial_shape=False, subplots enabled=True, $render_gt_shape=False,$ channels=None, interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt marker face colour='y', gt marker edge colour='k', gt line colour='y', render lines=True, line style='-', line width=2, render markers=True. marker style='o', marker size=4, render numbering=False, numbers horizontal align='center', $marker\ edge\ width=1.0,$ numbers_vertical_align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title="', legend_font_name='sans-serif', legend_font_style='normal', legend_font_weight='normal', legend font size=10, legend_marker_scale=None, end location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, *legend_border=True*, legend_border_padding=None, *legend_shadow=False*, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (*bool*, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.

• **final_marker_face_colour** (See Below, optional) - The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) – The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional)—The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **line_style** (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- legend_font_size (int, optional) The font size of the legend.
- legend_font_weight (See Below, optional)—The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.

- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- $figure_size$ ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view iterations (figure id=None, new figure=False, iters=None, render image=True, subplots enabled=False, channels=None. interpolation='bilinear', render lines=True, cmap name=None, alpha=1.0, masked=True. line_style='-', line width=2, line colour=None, render_markers=True, marker edge colour=None, marker face colour=None, marker style='o', $marker\ size=4$, marker_edge_width=1.0, render numbering=False, numbers horizontal align='center'. numbers vertical align='bottom'. numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, leg*end_border=True*, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=None,axes y limits=None, $axes \ x \ ticks=None,$ axes y ticks=None, figure size=(7,7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (str (See Below), optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:}
```

- line_width (*float* or *list* of *float*, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_colour** (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_width (*float* or *list* of *float*, optional) – The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different

value per iteration shape.

- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (*str* (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour (See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend title (str, optional) The title of the legend.
- **legend_font_name** (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- **legend_bbox_to_anchor** ((*float*, *float*) *tuple*, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- **render_axes** (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property appearance_parameters

Returns the *list* of appearance parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type *list* of (n_params,) *ndarray*

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property n_iters_per_scale

Returns the number of iterations per scale of the fitting process.

Type list of int

property n_scales

Returns the number of scales used during the fitting process.

2.1. Deformable Models

Type int

property reconstructed_initial_shapes

Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

Type list of menpo.shape.PointCloud

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type list of (n_params,) ndarray

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists) and *final_shape*.

Type list of menpo.shape.PointCloud

UnifiedAAMCLMAlgorithmResult

class menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult (shapes,

shape_parameters, appearance_parameters, initial_shape=None, image=None, gt_shape=None, costs=None)

Bases: ParametricIterativeResult

Class for storing the iterative result of a Unified AAM-CLM optimisation algorithm.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of shapes per iteration. The first and last members correspond to the initial and final shapes, respectively.
- **shape_parameters** (*list* of (n_shape_parameters,) *ndarray*) The *list* of shape parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.
- appearance_parameters (list of (n_appearance_parameters,) ndarray) The list of appearance parameters per iteration. The first and last members correspond to the initial and final shapes, respectively.
- initial_shape (*menpo.shape.PointCloud* or None, optional) The initial shape from which the fitting process started. If None, then no initial shape is assigned.

- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- **gt_shape** (*menpo.shape.PointCloud* or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.
- **costs** (*list* of *float* or None, optional) The *list* of cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

displacements_stats (stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be 'mean', 'median', 'min' or 'max'

errors (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (callable, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

• ValueError - Initial shape has not been set, so the initial error cannot be computed

ValueError – Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-
                                                              marker_style='o',
                    line width=2,
                                   render_markers=True,
                                                                                   marker\ size=4,
               marker_face_colour='b',
                                            marker_edge_colour='k',
                                                                          marker\ edge\ width=1.0,
               render \ axes=True,
                                          axes font name='sans-serif',
                                                                               axes font size=10,
               axes font style='normal',
                                              axes_font_weight='normal',
                                                                                axes x limits=0.0,
               axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(10, 6),
               render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the cost function evolution at each fitting iteration.
```

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'}, optional) The style of the lines.
- line width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (*colour* or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers.If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (float, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes y ticks (list or tuple or None, optional) The ticks of the y axis.
- figure size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_errors (compute_error=None, figure_id=None,
                                                          new_figure=False,
                                                                                render_lines=True,
                line colour='b',
                                      line style='-',
                                                         line width=2,
                                                                             render markers=True,
                marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
                marker\ edge\ width=1.0,
                                              render axes=True,
                                                                      axes font name='sans-serif',
                axes\_font\_size=10,
                                         axes_font_style='normal',
                                                                       axes_font_weight='normal',
                axes x limits=0.0, axes y limits=None, axes x ticks=None, axes y ticks=None,
                figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the error evolution at each fitting iteration.
```

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• marker_size (int, optional) – The size of the markers in points.

• marker_face_colour (*colour* or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (*float* or (*float*, *float*) or None, optional) The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves' width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style({'-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed initial error (compute error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed_initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view (figure id=None, new figure=False, render image=True, render final shape=True, render_gt_shape=False, render initial shape=False, *subplots enabled=True*, nels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial_marker_face_colour='b', initial_marker_edge_colour='k', initial_line_colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line_style='-', line_width=2, marker_style='o', marker_size=4, render_markers=True, $marker_edge_width=1.0$, render_numbering=False, numbers_horizontal_align='center', numbers_font_name='sans-serif', numbers_vertical_align='bottom', numbers_font_size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', legend font size=10, legend font weight='normal', legend marker scale=None, end location=2, legend bbox to anchor=(1.05,1.0), legend border axes pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend border=True, legend border padding=None, legend shadow=False, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=None, axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(7,7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape,

Parameters

• **figure_id** (*object*, optional) – The id of the figure to be used.

ground truth shape and the image, id they were provided.

- new_figure (bool, optional) If True, a new figure is created.
- render_image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.

- **render_initial_shape** (*bool*, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- **final_marker_face_colour** (See Below, optional) The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) - The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **gt_line_colour** (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **line_style** (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (*float* or *list* of *float*, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (*bool*, optional) If True, the legend will be rendered.
- legend_title (str, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- **legend_font_size** (*int*, optional) The font size of the legend.
- legend_font_weight (See Below, optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **legend_marker_scale** (*float*, optional) – The relative size of the legend markers with respect to the original

• **legend_location** (*int*, optional) – The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes y ticks (list or tuple or None, optional) The ticks of the y axis.
- figure size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view_iterations (figure_id=None, new_figure=False, iters=None, render_image=True, *subplots enabled=False*, channels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, render_lines=True, line style='-', line width=2, line colour=None, render markers=True, marker_edge_colour=None, marker_face_colour=None, marker_style='o', $marker\ size=4$, $marker_edge_width=1.0$, render_numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers font colour='k', render legend=True, legend title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend bbox to anchor=(1.05, 1.0), legend border axes pad=None, $legend \ n \ columns=1$, legend horizontal spacing=None, legend vertical spacing=None, legend border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes x limits=None,axes y limits=None, axes x ticks=None, $axes_v_ticks=None, figure_size=(7, 7)$ Visualize the iterations of the fitting process.

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the

image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• interpolation (str (See Below), optional) — The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- **alpha** (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_style (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -., :}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_colour (colour or list of colour (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_markers (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (*int* or *list* of *int*, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (str (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (*str* (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (*bool*, optional) If True, the legend will be rendered.
- **legend title** (*str*, optional) The title of the legend.
- **legend_font_name** (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend** n **columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

2.1. Deformable Models

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (*float* or (*float*, *float*) or None, optional) The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property appearance_parameters

Returns the *list* of appearance parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type *list* of (n_params,) *ndarray*

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property reconstructed_initial_shape

Returns the initial shape's reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type menpo.shape.PointCloud

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *reconstructed_initial_shape* and *final_shape*.

Type *list* of (n_params,) *ndarray*

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists), *reconstructed_initial_shape* and *final_shape*.

Type list of menpo.shape.PointCloud

2.1.6 menpofit.dlib

Ensemble of Regression Trees (provided by DLib)

Method that employs gradient boosting for learning an ensemble of regression trees to estimate the landmark positions directly from a sparse subset of pixel intensities.

DlibERT

```
class menpofit.dlib.DlibERT (images,
                                                                     bounding_box_group_glob=None,
                                                   group=None,
                                      reference shape=None,
                                                                   diagonal=None,
                                                                                         scales=(0.5,
                                                                              n_dlib_perturbations=1.
                                      1.0).
                                                  n_perturbations=30,
                                      perturb_from_gt_bounding_box=<function
                                      noisy_shape_from_bounding_box>,
                                                                            n\_iterations=10,
                                                                                                 fea-
                                      ture padding=0, n pixel pairs=400, distance prior weighting=0.1,
                                      regularisation_weight=0.1,
                                                                   n_{split_tests}=20,
                                                                                        n trees=500,
                                      n tree levels=5, verbose=False)
```

 $\textbf{Bases: } \textit{MultiScaleNonParametric} \overline{\textbf{F}itter}$

Class for training a multi-scale Ensemble of Regression Trees model. This class uses the implementation provided by the official DLib package (http://dlib.net/) and makes it multi-scale.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

- bounding_box_group_glob (glob or None, optional) Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- **reference_shape** (*menpo.shape.PointCloud* or None, optional) The reference shape that will be used for normalising the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- n_perturbations (*int* or None, optional) The number of perturbations to be generated from each of the bounding boxes using *perturb_from_gt_bounding_box*. Note that the total number of perturbations is *n_perturbations* * *n_dlib_perturbations*.
- **perturb_from_gt_bounding_box** (*function*, optional) The function that will be used to generate the perturbations.
- n_dlib_perturbations (int or None or list of those, optional) The number of perturbations to be generated from the part of DLib. DLib calls this "oversampling amount". If list, it must specify a value per scale. Note that the total number of perturbations is n_perturbations * n_dlib_perturbations.
- n_iterations (*int* or *list* of *int*, optional) The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.
- **feature_padding** (*float* or *list* of *float*, optional) When we randomly sample the pixels for the feature pool we do so in a box fit around the provided training landmarks. By default, this box is the tightest box that contains the landmarks. However, you can expand or shrink the size of the pixel sampling region by setting a different value of padding. To explain this precisely, for a padding of 0 we say that the pixels are sampled from a box of size 1x1. The padding value is added to each side of the box. So a padding of 0.5 would cause the algorithm to sample pixels from a box that was 2x2, effectively multiplying the area pixels are sampled from by 4. Similarly, setting the padding to -0.2 would cause it to sample from a box 0.6x0.6 in size. If *list*, it must specify a value per scale.
- n_pixel_pairs (int or list of int, optional) P parameter from [1]. At each level of the cascade we randomly sample pixels from the image. These pixels are used to generate features for the random trees. So in general larger settings of this parameter give better accuracy but make the algorithm run slower. If list, it must specify a value per scale.
- distance_prior_weighting (float or list of float, optional) To decide how to split nodes in the regression trees the algorithm looks at pairs of pixels in the image. These pixel pairs are sampled randomly but with a preference for selecting pixels that are near each other. This parameter controls this "nearness" preference. In particular, smaller values will make the algorithm prefer to select pixels close together and larger values will make it care less about picking nearby pixel pairs. Note that this is the inverse of how it is defined in [1]. For this object, you should think of distance_prior_weighting as "the fraction of the bounding box will we traverse to find a neighboring pixel". Nominally, this is normalized

between 0 and 1. So reasonable settings are values in the range (0, 1). If *list*, it must specify a value per scale.

- regularisation_weight (*float* or *list* of *float*, optional) Boosting regularization parameter *nu* from [1]. Larger values may cause overfitting but improve performance on training data. If *list*, it must specify a value per scale.
- n_split_tests (int or list of int, optional) When generating the random trees we randomly sample n_split_tests possible split features at each node and pick the one that gives the best split. Larger values of this parameter will usually give more accurate outputs but take longer to train. It is equivalent of S from [1]. If list, it must specify a value per scale.
- n_trees (*int* or *list* of *int*, optional) Number of trees created for each cascade. The total number of trees in the learned model is equal n_trees * n_tree_levels. Equivalent to *K* from [1]. If *list*, it must specify a value per scale.
- n_tree_levels (int or list of int, optional) The number of levels in the tree (depth of tree). In particular, there are pow(2, n_tree_levels) leaves in each tree. Equivalent to F from [1]. If list, it must specify a value per scale.
- verbose (bool, optional) If True, then the progress of building ERT will be printed.

References

fit_from_bb (image, bounding_box, gt_shape=None)

Fits the model to an image given an initial bounding box.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.

Returns fitting_result (MultiScaleNonParametricIterativeResult) - The result of the fitting procedure.

fit_from_shape (image, initial_shape, gt_shape=None)

Fits the model to an image. Note that it is not possible to initialise the fitting process from a shape. Thus, this method raises a warning and calls *fit_from_bb* with the bounding box of the provided *initial_shape*.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start. Note that the shape won't actually be used, only its bounding box.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.

Returns fitting_result (MultiScaleNonParametricIterativeResult) - The result of the fitting procedure.

2.1. Deformable Models

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *closure*

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

DlibWrapper

class menpofit.dlib.DlibWrapper(model)

Bases: object

Wrapper class for fitting a pre-trained ERT model. Pre-trained models are provided by the official DLib package (http://dlib.net/).

Parameters model (*Path* or *str*) – Path to the pre-trained model.

fit_from_bb (image, bounding_box, gt_shape=None)

Fits the model to an image given an initial bounding box.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- bounding_box (menpo.shape.PointDirectedGraph) The initial bounding box.
- gt_shape (menpo.shape.PointCloud) The ground truth shape associated to the image.

Returns fitting_result (Result) – The result of the fitting procedure.

fit_from_shape (image, initial_shape, gt_shape=None)

Fits the model to an image. Note that it is not possible to initialise the fitting process from a shape. Thus, this method raises a warning and calls *fit from bb* with the bounding box of the provided *initial shape*.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start. Note that the shape won't actually be used, only its bounding box.
- gt_shape (menpo.shape.PointCloud) The ground truth shape associated to the image.

Returns fitting_result (Result) – The result of the fitting procedure.

2.1.7 menpofit.1k

Lucas-Kanade Alignment

LK performs alignment (or optical flow estimation) between a template image and a test image with respect to an affine transformation.

LucasKanadeFitter

Bases: MultiScaleNonParametricFitter

Class for defining a multi-scale Lucas-Kanade fitter that performs alignment with respect to a homogeneous transform. Please see the references for a basic list of relevant papers.

Parameters

- template (menpo.image.Image) The template image.
- **group** (*str* or None, optional) The landmark group of the *template* that will be used as reference shape. If None and the *template* only has a single landmark group, then that is the one that will be used.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo, feature for a list of potential features.
- **diagonal** (*int* or None, optional) This parameter is used to rescale the reference shape (specified by *group*) so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- **scales** (*tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale.
- transform (*subclass* of *DP* and *DX*, optional) A differential homogeneous transform object, e.g. *DifferentiableAlignmentAffine*.
- **algorithm_cls** (*class*, optional) The Lukas-Kanade optimisation algorithm that will get applied. The possible algorithms in *menpofit.lk.algorithm* are:

Class	Warp Direction	Warp Update
ForwardAdditive	Forward	Additive
ForwardCompositional	Forward	Compositional
InverseCompositional	Inverse	

 residual_cls (class subclass, optional) – The residual that will get applied. All possible residuals are:

Class	Description
SSD	Sum of Squared Differences
FourierSSD	Sum of Squared Differences on Fourier domain
ECC	Enhanced Correlation Coefficient
GradientImages	Image Gradient
GradientCorrelation	Gradient Correlation

References

fit_from_bb (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Fits the multi-scale fitter to an image given an initial shape.

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (*bool*, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned

fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

 kwargs (dict, optional) – Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a LucasKanadeResult.

Parameters

- image (menpo.image.Image or subclass) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *closure*

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

Optimisation Algorithms

ForwardAdditive

class menpofit.lk.ForwardAdditive (template, transform, residual, eps=1e-10)

Bases: LucasKanade

Forward Additive (FA) Lucas-Kanade algorithm.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*) Execute the optimization algorithm.

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.

- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

Returns fitting_result (LucasKanadeAlgorithmResult) – The parametric iterative fitting result.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a LucasKanadeResult.

Parameters

- **image** (*menpo.image.Image* or *subclass*) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped images (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

ForwardCompositional

 $\textbf{class} \ \texttt{menpofit.lk.ForwardCompositional} \ (\textit{template}, \textit{transform}, \textit{residual}, \textit{eps=1e-10})$

Bases: LucasKanade

Forward Compositional (FC) Lucas-Kanade algorithm

Parameters

- **template** (*menpo.image.Image* or subclass) The image template.
- transform (*subclass* of *DP* and *DX*, optional) A differential affine transform object, e.g. *DifferentiableAlignmentAffine*.
- residual (class subclass, optional) The residual that will get applied. All possible residuals are:

Class	Description
SSD	Sum of Squared Differences
FourierSSD	Sum of Squared Differences on Fourier domain
ECC	Enhanced Correlation Coefficient
GradientImages	Image Gradient
GradientCorrelation	Gradient Correlation

• **eps** (*float*, optional) – Value for checking the convergence of the optimization.

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*) Execute the optimization algorithm.

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

Returns fitting_result (LucasKanadeAlgorithmResult) - The parametric iterative fitting result.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a LucasKanadeResult.

Parameters

- image (menpo.image.Image or subclass) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

InverseCompositional

class menpofit.lk.InverseCompositional (template, transform, residual, eps=1e-10)
 Bases: LucasKanade

Inverse Compositional (IC) Lucas-Kanade algorithm

Parameters

- template (menpo.image.Image or subclass) The image template.
- **transform** (*subclass* of *DP* and *DX*, optional) A differential affine transform object, e.g. *DifferentiableAlignmentAffine*.
- residual (class subclass, optional) The residual that will get applied. All possible residuals are:

Class	Description
SSD	Sum of Squared Differences
FourierSSD	Sum of Squared Differences on Fourier domain
ECC	Enhanced Correlation Coefficient
GradientImages	Image Gradient
GradientCorrelation	Gradient Correlation

• eps (float, optional) – Value for checking the convergence of the optimization.

2.1. Deformable Models

run (*image*, *initial_shape*, *gt_shape=None*, *max_iters=20*, *return_costs=False*) Execute the optimization algorithm.

Parameters

- image (menpo.image.Image) The input test image.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the optimization will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape of the image. It is only needed in order to get passed in the optimization result object, which has the ability to compute the fitting error.
- max_iters (*int*, optional) The maximum number of iterations. Note that the algorithm may converge, and thus stop, earlier.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.

Returns fitting_result (LucasKanadeAlgorithmResult) – The parametric iterative fitting result.

warped_images (image, shapes)

Given an input test image and a list of shapes, it warps the image into the shapes. This is useful for generating the warped images of a fitting procedure stored within a LucasKanadeResult.

Parameters

- image (menpo.image.Image or subclass) The input image to be warped.
- **shapes** (*list* of *menpo.shape.PointCloud*) The list of shapes in which the image will be warped. The shapes are obtained during the iterations of a fitting procedure.

Returns warped_images (*list* of *menpo.image.MaskedImage* or *ndarray*) – The warped images.

Residuals

SSD

class menpofit.lk.SSD(kernel=None)

Bases: Residual

Class for Sum of Squared Differences residual.

References

cost closure()

Method to compute the optimization cost.

Returns cost (*float*) – The cost value.

classmethod gradient (image, forward=None)

Calculates the gradients of the given method.

If forward is provided, then the gradients are warped (as required in the forward additive algorithm)

Parameters

- **image** (*menpo.image.Image*) The image to calculate the gradients for
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

hessian (sdi, sdi2=None)

Calculates the Gauss-Newton approximation to the Hessian.

This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

$$JJ^{T}$$

Parameters

- **sdi** ((N, n_params) *ndarray*) The steepest descent images.
- sdi2 ((N, n_params) ndarray or None, optional) The steepest descent images.

Returns H ((n_params, n_params) *ndarray*) – The approximation to the Hessian

steepest_descent_images (image, dW_dp, forward=None)

Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

$$\nabla I \frac{\partial W}{\partial p}$$

The input image is vectorised (N-pixels) so that masked images can be handled.

Parameters

- **image** (*menpo.image.Image*) The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
- **dW_dp** (*ndarray*) The Jacobian of the warp.
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

Returns VT_dW_dp ((N, n_params) ndarray) - The steepest descent images

steepest_descent_update (sdi, image, template)

Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

$$\sum_{x} \left[\nabla I \frac{\partial W}{\partial p}\right]^{T} \left[T(x) - I(W(x; p))\right]$$

- **sdi** ((N, n_params) *ndarray*) The steepest descent images.
- **image** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)

• **template** (*menpo.image.Image*) – Either the warped image or the template (depending on the framework)

Returns sd_delta_p ((n_params,) ndarray) - The steepest descent parameter updates.

FourierSSD

class menpofit.lk.FourierSSD(kernel=None)

Bases: Residual

Class for Sum of Squared Differences on the Fourier domain residual.

References

cost_closure()

Method to compute the optimization cost.

Returns cost (*float*) – The cost value.

classmethod gradient(image, forward=None)

Calculates the gradients of the given method.

If forward is provided, then the gradients are warped (as required in the forward additive algorithm)

Parameters

- image (menpo.image.Image) The image to calculate the gradients for
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

hessian (sdi, sdi2=None)

Calculates the Gauss-Newton approximation to the Hessian.

This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

$$JJ^{T}$$

Parameters

- sdi ((N, n_params) ndarray) The steepest descent images.
- sdi2 ((N, n_params) ndarray or None, optional) The steepest descent images.

Returns H ((n_params, n_params) *ndarray*) – The approximation to the Hessian

steepest_descent_images (image, dW_dp, forward=None)

Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

$$\nabla I \frac{\partial W}{\partial p}$$

The input image is vectorised (*N*-pixels) so that masked images can be handled.

- **image** (*menpo.image.Image*) The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
- **dW_dp** (*ndarray*) The Jacobian of the warp.
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

Returns VT_dW_dp ((N, n_params) ndarray) - The steepest descent images

steepest_descent_update (sdi, image, template)

Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

$$\sum_{x} \left[\nabla I \frac{\partial W}{\partial p}\right]^{T} \left[T(x) - I(W(x;p))\right]$$

Parameters

- **sdi** ((N, n_params) *ndarray*) The steepest descent images.
- **image** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)
- **template** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)

Returns sd_delta_p ((n_params,) ndarray) – The steepest descent parameter updates.

ECC

class menpofit.lk.ECC

Bases: Residual

Class for Enhanced Correlation Coefficient residual.

References

cost closure()

Method to compute the optimization cost.

Returns cost (*float*) – The cost value.

classmethod gradient (image, forward=None)

Calculates the gradients of the given method.

If forward is provided, then the gradients are warped (as required in the forward additive algorithm)

- image (menpo.image.Image) The image to calculate the gradients for
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

hessian (sdi, sdi2=None)

Calculates the Gauss-Newton approximation to the Hessian.

This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

$$\mathbf{J}\mathbf{J}^{\mathbf{T}}$$

Parameters

- sdi ((N, n_params) ndarray) The steepest descent images.
- sdi2 ((N, n_params) ndarray or None, optional) The steepest descent images.

Returns H ((n_params, n_params) *ndarray*) – The approximation to the Hessian

steepest_descent_images (image, dW_dp, forward=None)

Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

$$\nabla I \frac{\partial W}{\partial p}$$

The input image is vectorised (N-pixels) so that masked images can be handled.

Parameters

- **image** (*menpo.image.Image*) The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
- dW_dp (ndarray) The Jacobian of the warp.
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

Returns VT_dW_dp ((N, n_params) ndarray) - The steepest descent images

steepest_descent_update (sdi, image, template)

Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

$$\sum_{x} \left[\nabla I \frac{\partial W}{\partial p}\right]^{T} \left[T(x) - I(W(x; p))\right]$$

Parameters

- **sdi** ((N, n_params) *ndarray*) The steepest descent images.
- **image** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)
- **template** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)

Returns sd_delta_p ((n_params,) ndarray) – The steepest descent parameter updates.

GradientImages

class menpofit.lk.GradientImages

Bases: Residual

Class for Gradient Images residual.

References

cost_closure()

Method to compute the optimization cost.

Returns cost (*float*) – The cost value.

classmethod gradient (image, forward=None)

Calculates the gradients of the given method.

If forward is provided, then the gradients are warped (as required in the forward additive algorithm)

Parameters

- image (menpo.image.Image) The image to calculate the gradients for
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

hessian (sdi, sdi2=None)

Calculates the Gauss-Newton approximation to the Hessian.

This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

$$JJ^{T}$$

Parameters

- **sdi** ((N, n_params) *ndarray*) The steepest descent images.
- sdi2 ((N, n_params) ndarray or None, optional) The steepest descent images.

Returns H ((n_params, n_params) *ndarray*) – The approximation to the Hessian

steepest_descent_images (image, dW_dp, forward=None)

Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

$$\nabla I \frac{\partial W}{\partial p}$$

The input image is vectorised (*N*-pixels) so that masked images can be handled.

- **image** (*menpo.image.Image*) The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
- **dW_dp** (*ndarray*) The Jacobian of the warp.

• **forward** (*tuple* or None, optional) – A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

Returns VT_dW_dp ((N, n_params) ndarray) - The steepest descent images

steepest_descent_update (sdi, image, template)

Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

$$\sum_{x} \left[\nabla I \frac{\partial W}{\partial p}\right]^{T} \left[T(x) - I(W(x;p))\right]$$

Parameters

- sdi ((N, n params) ndarray) The steepest descent images.
- **image** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)
- **template** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)

Returns sd_delta_p ((n_params,)) ndarray) - The steepest descent parameter updates.

GradientCorrelation

class menpofit.lk.GradientCorrelation

Bases: Residual

Class for Gradient Correlation residual.

References

cost_closure()

Method to compute the optimization cost.

Returns cost (*float*) – The cost value.

classmethod gradient (image, forward=None)

Calculates the gradients of the given method.

If forward is provided, then the gradients are warped (as required in the forward additive algorithm)

Parameters

- **image** (*menpo.image.Image*) The image to calculate the gradients for
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

hessian (sdi, sdi2=None)

Calculates the Gauss-Newton approximation to the Hessian.

This is abstracted because some residuals expect the Hessian to be pre-processed. The Gauss-Newton approximation to the Hessian is defined as:

$$\mathbf{J}\mathbf{J}^{\mathbf{T}}$$

Parameters

- sdi ((N, n params) ndarray) The steepest descent images.
- sdi2 ((N, n_params) ndarray or None, optional) The steepest descent images.

Returns H ((n_params, n_params) *ndarray*) – The approximation to the Hessian

steepest_descent_images (image, dW_dp, forward=None)

Calculates the standard steepest descent images.

Within the forward additive framework this is defined as

$$\nabla I \frac{\partial W}{\partial p}$$

The input image is vectorised (N-pixels) so that masked images can be handled.

Parameters

- **image** (*menpo.image.Image*) The image to calculate the steepest descent images from, could be either the template or input image depending on which framework is used.
- **dW_dp** (*ndarray*) The Jacobian of the warp.
- **forward** (*tuple* or None, optional) A *tuple* containing the extra weights required for the function *warp* (which should be passed as a function handle), i.e. (`menpo.image. Image`, `menpo.transform.AlignableTransform>`). If None, then the optimization algorithm is assumed to be inverse.

Returns VT_dW_dp ((N, n_params) ndarray) - The steepest descent images

steepest_descent_update (sdi, image, template)

Calculates the steepest descent parameter updates.

These are defined, for the forward additive algorithm, as:

$$\sum_{x} \left[\nabla I \frac{\partial W}{\partial p}\right]^{T} \left[T(x) - I(W(x;p))\right]$$

Parameters

- **sdi** ((N, n_params) *ndarray*) The steepest descent images.
- **image** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)
- **template** (*menpo.image.Image*) Either the warped image or the template (depending on the framework)

Returns sd delta p ((n params,) *ndarray*) – The steepest descent parameter updates.

2.1. Deformable Models

Fitting Result

LucasKanadeResult

```
class menpofit.lk.result.LucasKanadeResult(results, scales, scale_transforms, image=None, gt shape=None)
```

Bases: MultiScaleParametricIterativeResult

Class for storing the multi-scale iterative fitting result of an ATM. It holds the shapes, shape parameters and costs per iteration.

Parameters

- results (list of ATMAlgorithmResult) The list of optimization results per scale.
- scales (list or tuple) The list of scale values per scale (low to high).
- **affine_transforms** (*list* of *menpo.transform*. *Affine*) The list of affine transforms per scale that transform the shapes into the original image space.
- scale_transforms (list of menpo.shape.Scale) The list of scaling transforms per scale.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

displacements_stats (stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be 'mean', 'median', 'min' or 'max'

errors (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final error(compute error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

```
initial_error(compute_error=None)
```

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- ValueError Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-
', line_width=2, render_markers=True, marker_style='o', marker_size=4,
marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0,
render_axes=True, axes_font_name='sans-serif', axes_font_size=10,
axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0,
axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6),
render_grid=True, grid_line_style='--', grid_line_width=0.5)
Plot of the cost function evolution at each fitting iteration.
```

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'}, optional) The style of the lines.
- line_width (*float*, optional) The width of the lines.
- render markers (bool, optional) If True, the markers will be rendered.
- marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (*int*, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- **figure_size** ((*float*, *float*) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker edge width (*float*, optional) – The width of the markers' edge.

- render axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines
- grid line width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

plot_errors (compute_error=None, figure_id=None, new_figure=False, render lines=True, line width=2, line colour='b', line style='-', render markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', $marker_edge_width=1.0$, render_axes=True, axes_font_name='sans-serif', $axes_font_size=10$, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5) Plot of the error evolution at each fitting iteration.

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (*bool*, optional) If True, the line will be rendered.

• line_colour (*colour* or None (See below), optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -:}
```

- line_width (float, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) - The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• **axes_x_limits** (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves'

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style({'-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed_initial_error (compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed_initial_shapes[0]*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result(pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view (figure id=None, new figure=False, render image=True, render final shape=True, render_initial_shape=False, subplots enabled=True, $render_gt_shape=False,$ channels=None, interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt marker face colour='y', gt marker edge colour='k', gt line colour='y', render lines=True, line style='-', line width=2, render markers=True. marker style='o', marker size=4, render numbering=False, numbers horizontal align='center', $marker\ edge\ width=1.0,$ numbers_vertical_align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title="', legend_font_name='sans-serif', legend_font_style='normal', legend_font_weight='normal', legend font size=10, legend_marker_scale=None, end location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, *legend_border=True*, legend_border_padding=None, *legend_shadow=False*, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape,

ground truth shape and the image, id they were provided.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (bool, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (bool, optional) If True and the ground truth shape exists, then it gets rendered.
- subplots_enabled (bool, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- channels (int or list of int or all or None) If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- cmap_name (str, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (float, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.

• **final_marker_face_colour** (See Below, optional) - The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) - The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional)—The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour
of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- **line_style** (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (*bool*, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- **legend_font_size** (*int*, optional) The font size of the legend.
- legend_font_weight (See Below, optional)—The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- legend_border_axes_pad (*float*, optional) The pad between the axes and legend border
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.

- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view iterations (figure id=None, new figure=False, iters=None, render image=True, subplots enabled=False, channels=None. interpolation='bilinear', cmap name=None, render lines=True, alpha=1.0, masked=True. line_style='-', line width=2, line colour=None, render_markers=True, marker edge colour=None, marker face colour=None, marker style='o', $marker\ size=4$, marker_edge_width=1.0, render numbering=False, numbers horizontal align='center'. numbers vertical align='bottom'. numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=None,axes y limits=None, $axes \ x \ ticks=None,$ axes y ticks=None, figure size=(7,7)) Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (str (See Below), optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_colour** (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_markers (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_width (*float* or *list* of *float*, optional) – The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different

value per iteration shape.

- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (str (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend title (str, optional) The title of the legend.
- **legend_font_name** (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- **render_axes** (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property homogeneous_parameters

Returns the *list* of parameters of the homogeneous transform obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type *list* of (n_params,) *ndarray*

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type *menpo.shape.Image* or *subclass* or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property n_iters_per_scale

Returns the number of iterations per scale of the fitting process.

Type list of int

property n_scales

Returns the number of scales used during the fitting process.

Type int

property reconstructed_initial_shapes

Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

Type list of menpo.shape.PointCloud

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type list of (n_params,) ndarray

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists) and *final_shape*.

Type *list* of *menpo.shape.PointCloud*

LucasKanadeAlgorithmResult

Bases: ParametricIterativeResult

Class for storing the iterative result of a Lucas-Kanade Image Alignment optimization algorithm.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of shapes per iteration. The first and last members correspond to the initial and final shapes, respectively.
- homogeneous_parameters (*list* of (n_parameters,) *ndarray*) The *list* of parameters of the homogeneous transform per iteration. The first and last members correspond to the initial and final shapes, respectively.
- initial_shape (*menpo.shape.PointCloud* or None, optional) The initial shape from which the fitting process started. If None, then no initial shape is assigned.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.
- **costs** (*list* of *float* or None, optional) The *list* of cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

displacements_stats (stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError - type must be 'mean', 'median', 'min' or 'max'

errors (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error(compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-
                    line width=2,
                                                               marker style='o',
                                     render markers=True,
                                                                                   marker size=4,
               marker face colour='b',
                                            marker edge colour='k',
                                                                          marker\ edge\ width=1.0,
               render axes=True,
                                           axes font name='sans-serif',
                                                                                axes font size=10,
               axes_font_style='normal',
                                               axes_font_weight='normal',
                                                                                axes\_x\_limits=0.0,
               axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6),
               render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the cost function evolution at each fitting iteration.
```

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- **new_figure** (*bool*, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.

• **line_colour** (*colour* or None, optional) – The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'}, optional) The style of the lines.
- line_width (float, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker edge width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (*int*, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
'extra bold', 'black'}
```

• axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes y ticks (list or tuple or None, optional) The ticks of the y axis.
- figure size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -:}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• marker_size (*int*, optional) – The size of the markers in points.

 marker_face_colour (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (*float* or (*float*, *float*) or None, optional) The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves' width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

plot errors (compute error=None, figure id=None, new figure=False, render lines=True, render markers=True, line_style='-', line width=2, line colour='b', marker style='o', marker size=4, marker face colour='b', marker edge colour='k', $marker_edge_width=1.0$, render_axes=True, axes_font_name='sans-serif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=0.0, axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(10, 6), render grid=True, grid line style='--', grid line width=0.5) Plot of the error evolution at each fitting iteration.

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -:, :}
```

- line width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker edge width (float, optional) The width of the markers' edge.
- **render_axes** (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (*float* or (*float*, *float*) or None, optional) The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves' width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- **figure_size** ((*float*, *float*) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed_initial_error (compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed_initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a Result instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).

• pass_gt_shape (bool, optional) – If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view(figure id=None, new figure=False, render image=True, render final shape=True, render initial shape=False, render_gt_shape=False, *subplots enabled=True*, nels=None, interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, final marker face colour='r', final marker edge colour='k', final line colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, *line_style='-', line_width=2,* render_markers=True, marker_style='o', marker size=4, render_numbering=False, numbers_horizontal_align='center', $marker_edge_width=1.0$, numbers_vertical_align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title="', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, end_location=2, legend_bbox_to_anchor=(1.05, legend_border_axes_pad=None, 1.0), legend horizontal spacing=None, $legend \ n \ columns=1$, legend vertical spacing=None, *legend border=True*, legend_border_padding=None, legend shadow=False, end rounded corners=False, render axes=False, axes font name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(7,7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- **new figure** (*bool*, optional) If True, a new figure is created.
- **render_image** (*bool*, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (*bool*, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- ullet masked (bool, optional) If True, then the image is rendered as masked.
- **final_marker_face_colour** (See Below, optional) The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) – The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- line_style (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (str or list of str, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers font size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.

• numbers_font_weight (See Below, optional) - The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- legend_font_size (int, optional) The font size of the legend.
- legend_font_weight (See Below, optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- legend_n_columns (int, optional) The number of the legend's columns.

- legend_horizontal_spacing (float, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style
 of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view iterations (figure id=None, new figure=False, iters=None, render image=True, subplots enabled=False, channels=None. interpolation='bilinear', render lines=True, cmap name=None, alpha=1.0, masked=True. line_style='-', line width=2, line colour=None, render_markers=True, marker edge colour=None, marker face colour=None, marker style='o', $marker\ size=4$, marker_edge_width=1.0, render numbering=False, numbers horizontal align='center'. numbers vertical align='bottom'. numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=None,axes y limits=None, $axes \ x \ ticks=None,$ axes y ticks=None, figure size=(7,7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (str (See Below), optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_colour (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_markers (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_width (*float* or *list* of *float*, optional) – The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different

value per iteration shape.

- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (*str* (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend title (str, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- **render_axes** (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property homogeneous_parameters

Returns the *list* of parameters of the homogeneous transform obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

Type list of (n_params,) ndarray

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property reconstructed_initial_shape

Returns the initial shape's reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type menpo.shape.PointCloud

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *reconstructed initial shape* and *final shape*.

```
Type list of (n_params,) ndarray
```

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists), *reconstructed_initial_shape* and *final_shape*.

Type list of menpo.shape.PointCloud

2.1.8 menpofit.sdm

Supervised Descent Method

SDM is a cascaded-regression deformable model that learns average descent directions that minimise a given cost function.

SupervisedDescentFitter

```
bound-
class menpofit.sdm.SupervisedDescentFitter(images,
                                                                      group=None,
                                                        ing_box_group_glob=None,
                                                        sd_algorithm_cls=None,
                                                                                           refer-
                                                                                  diagonal=None,
                                                        ence_shape=None,
                                                        holistic_features=<function
                                                                                         no\_op>,
                                                        patch_features=<function
                                                                                         no\_op>,
                                                        patch\_shape=(17, 17), scales=(0.5, 1.0),
                                                        n iterations=3, n perturbations=30,
                                                                                            per-
                                                        turb_from_gt_bounding_box=<function
                                                        noisy_shape_from_bounding_box>,
                                                        batch_size=None, verbose=False)
```

Bases: MultiScaleNonParametricFitter

Class for training a multi-scale Supervised Descent model.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- bounding_box_group_glob (glob or None, optional) Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- **sd_algorithm_cls** (*class*, optional) The Supervised Descent algorithm to be used. The possible algorithms are are separated in the following four categories:

Non-parametric:

Class	Regression	
NonParametricNewton	IRLRegression	
NonParametricGaussNewton	IIRLRegression	
NonParametricPCRRegression	PCRRegression	
NonParametricOptimalRegression	OptimalLinearRegression	
NonParametricOPPRegression	OPPRegression	

Parametric shape:

Class	Regression
ParametricShapeNewton	IRLRegression
ParametricShapeGaussNewton	IIRLRegression
ParametricShapePCRRegression	PCRRegression
ParametricShapeOptimalRegressio	n OptimalLinearRegression
ParametricShapeOPPRegression	ParametricShapeOPPRegression

Parametric appearance:

Class	Regression
ParametricAppearanceProjectOutNewton	IRLRegression
ParametricAppearanceProjectOutGuassNewton	IIRLRegression
ParametricAppearanceMeanTemplateNewton	IRLRegression
ParametricAppearanceMeanTemplateGuassNewton	IIRLRegression
ParametricAppearanceWeightsNewton	IRLRegression
ParametricAppearanceWeightsGuassNewton	IIRLRegression

Parametric shape and appearance:

Class	Regression
FullyParametricProjectOutNewton	IRLRegression
FullyParametricProjectOutGaussNewton	IIRLRegression
FullyParametricMeanTemplateNewton	IRLRegression
FullyParametricWeightsNewton	IRLRegression
FullyParametricProjectOutOPP	OPPRegression

- **reference_shape** (*menpo.shape.PointCloud* or None, optional) The reference shape that will be used for normalising the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the *diagonal* before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape
 so that the diagonal of its bounding box matches the provided value. In other words, this
 parameter controls the size of the model at the highest scale. If None, then the reference
 shape does not get rescaled.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo, feature for a list of potential features.

- patch_features (closure or list of closure, optional) The features that will be extracted from the patches of the training images. Note that, as opposed to holistic_features, these features are extracted after extracting the patches. If list, then it must define a feature function per scale. Please refer to menpo.feature and menpofit.feature for a list of potential features.
- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- n_iterations (*int* or *list* of *int*, optional) The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.
- n_perturbations (*int*, optional) The number of perturbations to be generated from each of the bounding boxes using *perturb_from_gt_bounding_box*.
- **perturb_from_gt_bounding_box** (*callable*, optional) The function that will be used to generate the perturbations from each of the bounding boxes.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.
- **verbose** (*bool*, optional) If True, then the progress of the training will be printed.

References

fit_from_bb (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- bounding_box (menpo.shape.PointDirectedGraph) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

fit_from_shape (image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Method to increment the trained SDM with a new set of training images.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- bounding_box_group_glob (glob or None, optional) Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- **verbose** (*bool*, optional) If True, then the progress of training will be printed.
- **batch_size** (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type list of closure

property n_scales

Returns the number of scales.

Type int

property reference shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

Pre-defined Models

Models with pre-defined algorithms that are commonly-used in literature.

SDM

menpofit.sdm.SDM (images, group=None, bounding_box_group_glob=None, reference_shape=None, diagonal=None, holistic_features=<function no_op>, patch_features=<function no_op>, patch_shape=(17, 17), scales=(0.5, 1.0), n_iterations=3, n_perturbations=30, perturb_from_gt_bounding_box=<function noisy_shape_from_bounding_box>, batch_size=None, verbose=False)

Class for training a non-parametric multi-scale Supervised Descent model using NonParametricNewton.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- bounding_box_group_glob (glob or None, optional) Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for normalising the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- patch_features (closure or list of closure, optional) The features that will be extracted from the patches of the training images. Note that, as opposed to holistic_features, these features are extracted after extracting the patches. If list, then it must define a feature

function per scale. Please refer to *menpo.feature* and *menpofit.feature* for a list of potential features.

- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- n_iterations (*int* or *list* of *int*, optional) The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.
- n_perturbations (*int*, optional) The number of perturbations to be generated from each of the bounding boxes using *perturb_from_gt_bounding_box*.
- **perturb_from_gt_bounding_box** (*callable*, optional) The function that will be used to generate the perturbations from each of the bounding boxes.
- batch_size (*int* or None, optional) If an *int* is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.
- **verbose** (*bool*, optional) If True, then the progress of the training will be printed.

References

RegularizedSDM

class menpofit.sdm.RegularizedSDM (images, group=None, bounding_box_group_glob=None, alpha=0.0001, reference_shape=None, diagonal=None, holistic_features=<function no_op>, patch_features=<function no_op>, patch_shape=(17, 17), scales=(0.5, 1.0), n_iterations=6, n_perturbations=30, perturb_from_gt_bounding_box=<function noisy_shape_from_bounding_box>, batch_size=None, verbose=False)

Bases: SupervisedDescentFitter

Class for training a non-parametric multi-scale Supervised Descent model using NonParametricNewton with regularization.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.
- bounding_box_group_glob (glob or None, optional) Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- **alpha** (*float*, optional) The regression regularization parameter.

- reference_shape (menpo.shape.PointCloud or None, optional) The reference shape that will be used for normalising the size of the training images. The normalization is performed by rescaling all the training images so that the scale of their ground truth shapes matches the scale of the reference shape. Note that the reference shape is rescaled with respect to the diagonal before performing the normalisation. If None, then the mean shape will be used.
- diagonal (int or None, optional) This parameter is used to rescale the reference shape so that the diagonal of its bounding box matches the provided value. In other words, this parameter controls the size of the model at the highest scale. If None, then the reference shape does not get rescaled.
- holistic_features (closure or list of closure, optional) The features that will be extracted from the training images. Note that the features are extracted before warping the images to the reference shape. If list, then it must define a feature function per scale. Please refer to menpo.feature for a list of potential features.
- patch_features (closure or list of closure, optional) The features that will be extracted from the patches of the training images. Note that, as opposed to holistic_features, these features are extracted after extracting the patches. If list, then it must define a feature function per scale. Please refer to menpo.feature and menpofit.feature for a list of potential features.
- patch_shape ((int, int) or list of (int, int), optional) The shape of the patches to be extracted. If a list is provided, then it defines a patch shape per scale.
- **scales** (*float* or *tuple* of *float*, optional) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale. If *float*, then a single scale is assumed.
- n_iterations (*int* or *list* of *int*, optional) The number of iterations (cascades) of each level. If *list*, it must specify a value per scale. If *int*, then it defines the total number of iterations (cascades) over all scales.
- n_perturbations (*int*, optional) The number of perturbations to be generated from each of the bounding boxes using *perturb_from_gt_bounding_box*.
- **perturb_from_gt_bounding_box** (*callable*, optional) The function that will be used to generate the perturbations from each of the bounding boxes.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.
- verbose (bool, optional) If True, then the progress of the training will be printed.

References

fit_from_bb (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

- image (menpo.image.Image or subclass) The image to be fitted.
- bounding_box (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.

- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Method to increment the trained SDM with a new set of training images.

- images (list of menpo.image.Image) The list of training images.
- **group** (*str* or None, optional) The landmark group that corresponds to the ground truth shape of each image. If None and the images only have a single landmark group, then that is the one that will be used. Note that all the training images need to have the specified landmark group.

- bounding_box_group_glob (glob or None, optional) Glob that defines the bounding boxes to be used for training. If None, then the bounding boxes of the ground truth shapes are used.
- **verbose** (*bool*, optional) If True, then the progress of training will be printed.
- batch_size (int or None, optional) If an int is provided, then the training is performed in an incremental fashion on image batches of size equal to the provided value. If None, then the training is performed directly on the all the images.

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *closure*

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

Non-Parametric Algorithms

The cascaded regression of these algorithms is performed between landmark coordinates and image-based features.

NonParametricNewton

```
class menpofit.sdm.NonParametricNewton (patch\_features = < function no\_op >, patch\_shape = (17, 17), n\_iterations = 3, compute\_error = < function euclidean\_bb\_normalised\_error >, alpha = 0, bias = True)
```

Bases: NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Incremental Regularized Linear Regression (IRLRegression).

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (*int*, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.

• bias (bool, optional) – Flag that controls whether to use a bias term.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)
Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- **gt_shape** (class: *menpo.shape.PointCloud* or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (list of menpo.shape.PointCloud) The list of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

NonParametricGaussNewton

```
 \begin{array}{c} \textbf{class} \ \texttt{menpofit.sdm.NonParametricGaussNewton} \ (patch\_features = < function \\ patch\_shape = (17, 17), \quad n\_iterations = 3, \\ compute\_error = < function \\ clidean\_bb\_normalised\_error >, \quad alpha = 0, \\ bias = True, \ alpha 2 = 0) \end{array}
```

Bases: NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Indirect Incremental Regularized Linear Regression (*IIRLRegression*).

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (int, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- bias (*bool*, optional) Flag that controls whether to use a bias term.
- alpha2 (*float*, optional) The regularization parameter of the Hessian matrix.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

- image (menpo.image.Image or subclass) The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (class: menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (*bool*, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned

fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

NonParametricPCRRegression

Bases: NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Principal Component Regression (PCRRegression).

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (*int*, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- variance (*float* or None, optional) The SVD variance.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.

- current_shapes (list of menpo.shape.PointCloud) The list of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (class: menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) — The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

NonParametricOptimalRegression

Bases: NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Multivariate Linear Regression with optimal reconstructions (OptimalLinearRegression).

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (int, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- variance (float or None, optional) The SVD variance.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (class: menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

NonParametricOPPRegression

Bases: NonParametricSDAlgorithm

Class for training a non-parametric cascaded-regression algorithm using Multivariate Linear Regression with Orthogonal Procrustes Problem reconstructions (*OPPRegression*).

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (*int*, optional) The number of iterations (cascades).
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (class: menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

Parametric Shape Algorithms

The cascaded regression of these algorithms is performed between the parameters of a statistical shape model and image-based features.

ParametricShapeNewton

Bases: ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Incremental Regularized Linear Regression (IRLRegression).

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- **n_iterations** (*int*, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)
Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

• images (list of menpo.image.Image) – The list of training images.

- gt_shapes (list of menpo.shape.PointCloud) The list of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

ParametricShapeGaussNewton

```
 \textbf{class} \ \texttt{menpofit.sdm.ParametricShapeGaussNewton} \ (patch\_features = < function \\ patch\_shape = (17, 17), \ n\_iterations = 3, \\ shape\_model\_cls = < class \\ pofit.modelinstance.OrthoPDM'>, \\ compute\_error = < function \\ euclidean\_bb\_normalised\_error>, alpha = 0, \\ bias = True, alpha = 0)
```

Bases: ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Indirect Incremental Regularized Linear Regression (*IIRLRegression*).

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (int, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- bias (bool, optional) Flag that controls whether to use a bias term.
- **alpha2** (*float*, optional) The regularization parameter of the Hessian matrix.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) - The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

ParametricShapePCRRegression

Bases: ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Principal Component Regression (*PCRRegression*).

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (*int*, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- variance (*float* or None, optional) The SVD variance.
- bias (bool, optional) Flag that controls whether to use a bias term.

Raises ValueError – variance must be set to a number between 0 and 1

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- **images** (*list* of *menpo.image.Image*) The *list* of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

ParametricShapeOptimalRegression

class menpofit.sdm.ParametricShapeOptimalRegression(patch_features=<function</pre>

no_op>, patch_shape=(17, 17), n_iterations=3, shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>, compute_error=<function euclidean_bb_normalised_error>, variance=None, bias=True)

Bases: ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Multivariate Linear Regression with optimal reconstructions (OptimalLinearRegression).

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (int, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- variance (*float* or None, optional) The SVD variance.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)

Method to increment the model with the set of current shapes.

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.

• **verbose** (*bool*, optional) – If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

ParametricShapeOPPRegression

Bases: ParametricShapeSDAlgorithm

Class for training a cascaded-regression algorithm that employs a parametric shape model using Multivariate Linear Regression with Orthogonal Procrustes Problem reconstructions (*OPPRegression*).

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (int, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- whiten (*bool*, optional) Whether to use a whitened PCA model.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)
Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

• images (list of menpo.image.Image) – The list of training images.

- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Parametric Appearance Algorithms

The cascaded regression of these algorithms is performed between landmark coordinates and features that are based on a statistical parametric appearance model.

ParametricAppearanceProjectOutNewton

Bases: ParametricAppearanceNewton

Class for training a cascaded-regression Newton algorithm that employs a parametric appearance model using Incremental Regularized Linear Regression (IRLRegression). The algorithm uses the projected-out appearance vectors as features in the regression.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- current_shapes (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

- **image** (*menpo.image.Image* or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- **gt_shape** (*menpo.shape.PointCloud* or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) - The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

ParametricAppearanceProjectOutGuassNewton

class menpofit.sdm.ParametricAppearanceProjectOutGuassNewton(patch_features=<function

```
no_op>,
patch_shape=(17,
17),
n_iterations=3,
appear-
ance_model_cls=<class
'menpo.model.pca.PCAVectorModel'>,
com-
pute_error=<function
eu-
clidean_bb_normalised_error>,
alpha=0,
bias=True,
pha2=0)
```

Bases: ParametricAppearanceGaussNewton

Class for training a cascaded-regression Gauss-Newton algorithm that employs a parametric appearance model using Indirect Incremental Regularized Linear Regression (*IIRLRegression*). The algorithm uses the projected-out appearance vectors as features in the regression.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)
 Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) - The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

ParametricAppearanceMeanTemplateNewton

Bases: ParametricAppearanceNewton

Class for training a cascaded-regression Newton algorithm that employs a parametric appearance model using Incremental Regularized Linear Regression (IRLRegression). The algorithm uses the centered appearance vectors as features in the regression.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

alpha=0, bias=True)

train (*images*, *gt_shapes*, *current_shapes*, *prefix=", verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

Parametric Appearance Mean Template Guass Newton

class menpofit.sdm.ParametricAppearanceMeanTemplateGuassNewton (patch_features=<function

```
no_op>,
patch_shape=(17,
17),
n_iterations=3,
appear-
ance_model_cls=<class
'menpo.model.pca.PCAVectorModel'>,
com-
pute_error=<function
eu-
clidean_bb_normalised_error>,
alpha=0,
bias=True,
alpha2=0)
```

Bases: ParametricAppearanceGaussNewton

Class for training a cascaded-regression Gauss-Newton algorithm that employs a parametric appearance model using Indirect Incremental Regularized Linear Regression (*IIRLRegression*). The algorithm uses the centered appearance vectors as features in the regression.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (list of menpo.shape.PointCloud) The list of ground truth shapes that correspond to the images.
- current_shapes (list of menpo.shape.PointCloud) The list of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

ParametricAppearanceWeightsNewton

Bases: ParametricAppearanceNewton

Class for training a cascaded-regression Newton algorithm that employs a parametric appearance model using Incremental Regularized Linear Regression (IRLRegression). The algorithm uses the projection weights of the appearance vectors as features in the regression.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)
 Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (list of menpo.shape.PointCloud) The list of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

Parametric Appearance Weights Guass Newton

```
 \textbf{class} \ \texttt{menpofit.sdm.ParametricAppearanceWeightsGuassNewton} \ (patch\_features = < function \\ no\_op>, \\ patch\_shape = (17, 17), \\ n\_iterations = 3, appear-\\ ance\_model\_cls = < class \\ 'menpo.model.pca.PCAVectorModel'>, \\ com-\\ pute\_error = < function \\ eu-\\ clidean\_bb\_normalised\_error>, \\ alpha = 0, bias = True,
```

alpha2=0)

Bases: ParametricAppearanceGaussNewton

Class for training a cascaded-regression Gauss-Newton algorithm that employs a parametric appearance model using Indirect Incremental Regularized Linear Regression (*IIRLRegression*). The algorithm uses the projection weights of the appearance vectors as features in the regression.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- **images** (*list* of *menpo.image.Image*) The *list* of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- current_shapes (list of menpo.shape.PointCloud) The list of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (NonParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=", verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (list of menpo.shape.PointCloud) The list of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

Fully Parametric Algorithms

The cascaded regression is performed between the parameters of a statistical shape model and features that are based on a statistical parametric appearance model.

FullyParametricProjectOutNewton

Bases: ParametricAppearanceProjectOut

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Incremental Regularized Linear Regression (IRLRegression). The algorithm uses the projected-out appearance vectors as features in the regression.

Parameters

• patch_features (callable, optional) – The features to be extracted from the patches of an image.

alpha=0, bias=True)

- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (int, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- appearance_model_cls (menpo.model.PCAVectorModel or subclass) The class to be used for building the appearance model.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.

- **alpha** (*float*, optional) The regularization parameter.
- **bias** (*bool*, optional) Flag that controls whether to use a bias term.

 $\verb|increment| (images, gt_shapes, current_shapes, prefix=", verbose=False)|$

Method to increment the model with the set of current shapes.

Parameters

- **images** (*list* of *menpo.image.Image*) The *list* of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

 $\begin{tabular}{ll} \textbf{Returns fitting_result} & (\textit{ParametricIterativeResult}) - \textbf{The result of the fitting procedure}. \\ \end{tabular}$

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

FullyParametricProjectOutGaussNewton

class menpofit.sdm.FullyParametricProjectOutGaussNewton (patch_features=<function

no_op>,
patch_shape=(17,
17), n_iterations=3,
shape_model_cls=<class
'menpofit.modelinstance.OrthoPDM'>,
appearance_model_cls=<class
'menpo.model.pca.PCAVectorModel'>,
compute_error=<function euclidean_bb_normalised_error>,
alpha=0, bias=True, alpha2=0)

Bases: ParametricAppearanceProjectOut

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Indirect Incremental Regularized Linear Regression (*IIRLRegression*). The algorithm uses the projected-out appearance vectors as features in the regression.

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (int, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- appearance_model_cls (menpo.model.PCAVectorModel or subclass) The class to be used for building the appearance model.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- bias (*bool*, optional) Flag that controls whether to use a bias term.
- alpha2 (*float*, optional) The regularization parameter of the Hessian matrix.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- current_shapes (list of menpo.shape.PointCloud) The list of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

FullyParametricMeanTemplateNewton

class menpofit.sdm.FullyParametricMeanTemplateNewton(patch features=<function</pre>

no_op>, patch_shape=(17, 17), n_iterations=3, shape_model_cls=<class 'men-pofit.modelinstance.OrthoPDM'>, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True)

Bases: ParametricAppearanceMeanTemplate

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Incremental Regularized Linear Regression (*IRLRegression*). The algorithm uses the centered appearance vectors as features in the regression.

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (int, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- appearance_model_cls (menpo.model.PCAVectorModel or subclass) The class to be used for building the appearance model.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- bias (bool, optional) Flag that controls whether to use a bias term.

increment (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

- image (menpo.image.Image or subclass) The image to be fitted.
- **initial_shape** (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) – The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix="*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- gt_shapes (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

FullyParametricWeightsNewton

 $Bases: \verb"ParametricAppearanceWeights"$

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Incremental Regularized Linear Regression (IRLRegression). The algorithm uses the projection weights of the appearance vectors as features in the regression.

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- n_iterations (*int*, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- appearance_model_cls (menpo.model.PCAVectorModel or subclass) The class to be used for building the appearance model.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **alpha** (*float*, optional) The regularization parameter.
- bias (*bool*, optional) Flag that controls whether to use a bias term.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)
 Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) — The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

285

2.1. Deformable Models

FullyParametricProjectOutOPP

Bases: ParametricAppearanceProjectOut

Class for training a cascaded-regression algorithm that employs parametric shape and appearance models using Multivariate Linear Regression with Orthogonal Procrustes Problem reconstructions (OPPRegression).

Parameters

- patch_features (callable, optional) The features to be extracted from the patches of an image.
- patch_shape ((int, int), optional) The shape of the extracted patches.
- **n_iterations** (*int*, optional) The number of iterations (cascades).
- **shape_model_cls** (*subclass* of *PDM*, optional) The class to be used for building the shape model. The most common choice is *OrthoPDM*.
- appearance_model_cls (menpo.model.PCAVectorModel or subclass) The class to be used for building the appearance model.
- **compute_error** (*callable*, optional) The function to be used for computing the fitting error when training each cascade.
- **bias** (*bool*, optional) Flag that controls whether to use a bias term.

increment (images, gt_shapes, current_shapes, prefix=", verbose=False)
Method to increment the model with the set of current shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- current_shapes (list of menpo.shape.PointCloud) The list of current shapes that correspond to the images.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

run (*image*, *initial_shape*, *gt_shape=None*, *return_costs=False*, **kwargs)
Run the algorithm to an image given an initial shape.

Parameters

• image (menpo.image.Image or subclass) – The image to be fitted.

- initial_shape (menpo.shape.PointCloud) The initial shape from which the fitting procedure will start.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that this argument currently has no effect and will raise a warning if set to ``True``. This is because it is not possible to evaluate the cost function of this algorithm.

Returns fitting_result (ParametricIterativeResult) — The result of the fitting procedure.

train (*images*, *gt_shapes*, *current_shapes*, *prefix=''*, *verbose=False*) Method to train the model given a set of initial shapes.

Parameters

- images (list of menpo.image.Image) The list of training images.
- **gt_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of ground truth shapes that correspond to the images.
- **current_shapes** (*list* of *menpo.shape.PointCloud*) The *list* of current shapes that correspond to the images, which will be used as initial shapes.
- **prefix** (*str*, optional) The prefix to use when printing information.
- **verbose** (*bool*, optional) If True, then information is printed during training.

Returns current_shapes (*list* of *menpo.shape.PointCloud*) – The *list* of current shapes that correspond to the images.

2.2 Internal API

2.2.1 menpofit.builder

Building Functions

Collection of functions that are commonly-used by most deformable model builders.

align_shapes

```
menpofit.builder.align_shapes(shapes)
```

Function that aligns a set of shapes by applying Generalized Procrustes Analysis.

Parameters shapes (*list* of *menpo.shape.PointCloud*) – The input shapes.

Returns aligned_shapes (*list* of *menpo.shape.PointCloud*) – The list of aligned shapes.

build_patch_reference_frame

menpofit.builder.build_patch_reference_frame(landmarks, boundary=3, group='source', patch_shape=(17, 17))

Builds a patch-based reference frame from a particular set of landmarks.

Parameters

- landmarks (menpo.shape.PointCloud) The landmarks that will be used to build the reference frame.
- **boundary** (*int*, optional) The number of pixels to be left as a safe margin on the boundaries of the reference frame (has potential effects on the gradient computation).
- group (str, optional) Group that will be assigned to the provided set of landmarks on the reference frame.
- patch shape ((int, int), optional) The shape of the patches.

Returns patch_based_reference_frame (menpo.image.MaskedImage) – The patch-based reference frame.

build_reference_frame

menpofit.builder.build_reference_frame (landmarks, boundary=3, group='source')
Builds a reference frame from a particular set of landmarks.

Parameters

- landmarks (menpo.shape.PointCloud) The landmarks that will be used to build the reference frame.
- **boundary** (*int*, optional) The number of pixels to be left as a safe margin on the boundaries of the reference frame (has potential effects on the gradient computation).
- **group** (*str*, optional) Group that will be assigned to the provided set of landmarks on the reference frame.

Returns reference_frame (manpo.image.MaskedImage) – The reference frame.

compute_features

menpofit.builder.compute_features (images, features, prefix=", verbose=False)
Function that extracts features from a list of images.

Parameters

- images (list of menpo.image.Image) The set of images.
- **features** (*callable*) The features extraction function. Please refer to *menpo.feature* and *menpofit.feature*.
- **prefix** (*str*) The prefix of the printed information.
- **verbose** (*bool*, Optional) Flag that controls information and progress printing.

Returns feature_images (*list* of *menpo.image.Image*) – The list of feature images.

compute reference shape

menpofit.builder.compute_reference_shape (shapes, diagonal, verbose=False)
Function that computes the reference shape as the mean shape of the provided shapes.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) The set of shapes from which to build the reference shape.
- diagonal (int or None) If int, it ensures that the mean shape is scaled so that the diagonal of the bounding box containing it matches the provided value. If None, then the mean shape is not rescaled.
- **verbose** (*bool*, optional) If True, then progress information is printed.

Returns reference_shape (menpo.shape.PointCloud) – The reference shape.

densify_shapes

menpofit.builder.densify_shapes (shapes, reference_frame, transform)
Function that densifies a set of sparse shapes given a reference frame.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) The input shapes.
- reference_frame (menpo.image.BooleanImage) The reference frame, the mask of which will be used.
- transform (menpo.transform.Transform) The transform to use for mapping the dense points.

Returns dense_shapes (*list* of *menpo.shape.PointCloud*) – The list of dense shapes.

extract patches

menpofit.builder.extract_patches (images, shapes, patch_shape, normalise_function=<function no_op>, prefix=", verbose=False")

Function that extracts patches around the landmarks of the provided images.

Parameters

- images (list of menpo.image.Image) The set of images to warp.
- **shapes** (*list* of *menpo.shape.PointCloud*) The set of shapes that correspond to the images.
- patch_shape ((int, int)) The shape of the patches.
- **normalise_function** (*callable*) A normalisation function to apply on the values of the patches.
- **prefix** (*str*) The prefix of the printed information.
- **verbose** (*bool*, Optional) Flag that controls information and progress printing.

Returns patch_images (*list* of *menpo.image.Image*) — The list of images with the patches per image. Each output image has shape (n_center, n_offset, n_channels, patch_shape).

normalization wrt reference shape

```
menpofit.builder.normalization_wrt_reference_shape(images, group, diagonal, ver-
bose=False)
```

Function that normalizes the images' sizes with respect to the size of the mean shape. This step is essential before building a deformable model.

The normalization includes: 1) Computation of the reference shape as the mean shape of the images' landmarks. 2) Scaling of the reference shape using the diagonal. 3) Rescaling of all the images so that their shape's scale is in correspondence with the reference shape's scale.

Parameters

- images (list of menpo.image.Image) The set of images to normalize.
- **group** (*str*) If *str*, then it specifies the group of the images's shapes. If None, then the images must have only one landmark group.
- diagonal (int or None) If int, it ensures that the mean shape is scaled so that the diagonal of the bounding box containing it matches the provided value. If None, then the mean shape is not rescaled.
- verbose (bool, Optional) Flag that controls information and progress printing.

Returns

- **reference_shape** (*menpo.shape.PointCloud*) The reference shape that was used to resize all training images to a consistent object size.
- **normalized_images** (*list* of *menpo.image.Image*) The images with normalized size.

rescale_images_to_reference_shape

Function that normalizes the images' sizes with respect to the size of the provided reference shape. In other words, the function rescales the provided images so that the size of the bounding box of their attached shape is the same as the size of the bounding box of the provided reference shape.

Parameters

- images (list of menpo.image.Image) The set of images that will be rescaled.
- **group** (*str* or None) If *str*, then it specifies the group of the images's shapes. If None, then the images must have only one landmark group.
- $reference_shape (menpo.shape.PointCloud)$ The reference shape.
- **verbose** (*bool*, optional) If True, then progress information is printed.

Returns normalized_images (*list* of *menpo.image.Image*) – The rescaled images.

scale images

menpofit.builder.scale_images(images, scale, prefix=", return_transforms=False, verbose=False)

Function that rescales a list of images and optionally returns the scale transforms.

Parameters

- images (list of menpo.image.Image) The set of images to scale.
- **scale** (*float* or *tuple* of *floats*) The scale factor. If a tuple, the scale to apply to each dimension. If a single *float*, the scale will be applied uniformly across each dimension.
- **prefix** (*str*, optional) The prefix of the printed information.
- return_transforms (bool, optional) If True, then a list with the menpo.transform.Scale objects that were used to perform the rescale for each image is also returned.
- **verbose** (*bool*, optional) Flag that controls information and progress printing.

Returns

- scaled_images (list of menpo.image.Image) The list of rescaled images.
- **scale_transforms** (*list* of *menpo.transform.Scale*) The list of scale transforms that were used. It is returned only if *return_transforms* is True.

warp_images

menpofit.builder.warp_images(images, shapes, reference_frame, transform, prefix=", ver-bose=None)

Function that warps a list of images into the provided reference frame.

Parameters

- images (list of menpo.image.Image) The set of images to warp.
- **shapes** (*list* of *menpo.shape.PointCloud*) The set of shapes that correspond to the images.
- reference_frame (menpo.image.BooleanImage) The reference frame to warp to.
- transform (menpo.transform.Transform) Transform from the reference frame back to the image. Defines, for each pixel location on the reference frame, which pixel location should be sampled from on the image.
- **prefix** (*str*) The prefix of the printed information.
- **verbose** (*bool*, Optional) Flag that controls information and progress printing.

Returns warped_images (*list* of *menpo.image.MaskedImage*) – The list of warped images.

Warnings

MenpoFitBuilderWarning

```
class menpofit.builder.MenpoFitBuilderWarning
    Bases: Warning
```

A warning that some part of building the model may cause issues.

```
with_traceback()
```

Exception.with_traceback(tb) - set self.__traceback__ to tb and return self.

MenpoFitModelBuilderWarning

```
class menpofit.builder.MenpoFitModelBuilderWarning
    Bases: Warning
```

A warning that the parameters chosen to build a given model may cause unexpected behaviour.

```
with traceback()
```

Exception.with_traceback(tb) - set self.__traceback__ to tb and return self.

2.2.2 menpofit.checks

Functions for checking the parameters' values that are passed in MenpoFit's classes.

Parameters Check Functions

check diagonal

```
menpofit.checks.check_diagonal(diagonal)
```

Checks that the diagonal length used to normalize the images' size is ≥ 20 .

Parameters diagonal (*int*) – The value to check.

Returns diagonal (*int*) – The value if it's correct.

Raises ValueError – diagonal must be >= 20 or None

check_landmark_trilist

```
menpofit.checks.check_landmark_trilist(image, transform, group=None)
```

Checks that the provided image has a triangulated shape (thus an isntance of *menpo.shape.TriMesh*) and the transform is *menpo.transform.PiecewiseAffine*

Parameters

- **image** (*menpo.image.Image* or subclass) The input image.
- transform (menpo.transform.PiecewiseAffine) The transform object.
- group (str or None, optional) The group of the shape to check.

Raises Warning – The given images do not have an explicit triangulation applied. A Delaunay Triangulation will be computed and used for warping. This may be suboptimal and cause warping artifacts.

check trilist

menpofit.checks.check_trilist(shape, transform)

Checks that the provided shape is triangulated (thus an isntance of *menpo.shape.TriMesh*) and the transform is *menpo.transform.PiecewiseAffine*

Parameters

- **shape** (*menpo.shape.TriMesh*) The input shape (usually the reference/mean shape of a model).
- transform (menpo.transform.PiecewiseAffine) The transform object.

Raises Warning – The given images do not have an explicit triangulation applied. A Delaunay Triangulation will be computed and used for warping. This may be suboptimal and cause warping artifacts.

check model

menpofit.checks.check_model(model, cls)

Function that checks whether the provided class object is a subclass of the provided base class.

Parameters

- model (class) The object.
- cls (class) The required base class.

Raises ValueError – Model must be a {cls} instance.

Multi-Scale Parameters Check Functions

check scales

```
menpofit.checks.check_scales(scales)
```

Checks that the provided *scales* argument is either *int* or *float* or an iterable of those. It makes sure that it returns a *list* of *scales*.

Parameters scales (int or float or list/tuple of those) – The value to check.

Returns scales (*list* of *int* or *float*) – The scales in a list.

Raises ValueError – scales must be an int/float or a list/tuple of int/float

check multi scale param

menpofit.checks.check_multi_scale_param(n_scales, types, param_name, param)

General function for checking a parameter defined for multiple scales. It raises an error if the parameter is not an iterable with the correct size and correct types.

Parameters

- n_scales (int) The number of scales.
- **types** (*tuple*) The *tuple* of variable types that the parameter is allowed to have.
- param_name (str) The name of the parameter.
- param (*types*) The parameter value.

Returns param (*list* of *types*) – The list of values per scale.

Raises ValueError – {param_name} must be in {types} or a list/tuple of {types} with the same length as the number of scales

check callable

menpofit.checks.check_callable(callables, n_scales)

Checks the callable type per level.

Parameters

- callables (callable or list of callables) The callable to be used per scale.
- n_scales (int) The number of scales.

Returns callable_list (*list*) – A *list* of callables.

Raises ValueError – callables must be a callable or a list/tuple of callables with the same length as the number of scales

check_patch_shape

menpofit.checks.check_patch_shape(patch_shape, n_scales)

Function for checking a multi-scale *patch_shape* parameter value.

Parameters

- patch_shape (list/tuple of int/float or list of those) The patch shape per scale
- n_scales (int) The number of scales.

Returns patch_shape (*list* of *list/tuple* of *int/float*) – The list of patch shape per scale.

Raises ValueError – patch_shape must be a list/tuple of int or a list/tuple of lit/tuple of int/float with the same length as the number of scales

check_max_iters

```
menpofit.checks.check_max_iters (max_iters, n_scales)
```

Function that checks the value of a max_iters parameter defined for multiple scales. It must be int or list of int.

Parameters

- max iters (int or list of int) The value to check.
- n_scales (*int*) The number of scales.

Returns max_iters (*list* of *int*) – The list of values per scale.

Raises ValueError – max_iters can be integer, integer list containing 1 or {n_scales} elements or None

check max components

menpofit.checks.check_max_components (max_components, n_scales, var_name)

Checks the maximum number of components per scale. It must be None or *int* or *float* or a *list* of those containing 1 or {n_scales} elements.

Parameters

- max_components (None or int or float or a list of those) The value to check.
- n scales (int) The number of scales.
- **var_name** (*str*) The name of the variable.

Returns max_components (*list* of None or *int* or *float*) – The list of max components per scale.

Raises ValueError – {var_name} must be None or an int > 0 or a $0 \le 1$ or a list of those containing 1 or {n_scales} elements

set models components

menpofit.checks.set_models_components (models, n_components)

Function that sets the number of active components to a list of models.

Parameters

- models (*list* or *class*) The list of models per scale.
- n_components (int or float or None or list of those) The number of components per model.

Raises ValueError – n_components can be an integer or a float or None or a list containing 1 or n_scales of those

check algorithm cls

menpofit.checks.check_algorithm_cls (algorithm_cls, n_scales, base_algorithm_cls)

Function that checks whether the list of class objects defined per scale are subclasses of the provided base class.

Parameters

- algorithm_cls (class or list of class) The list of objects per scale.
- n_scales (int) The number of scales.
- base_algorithm_cls (class) The required base class.

Raises ValueError – algorithm_cls must be a subclass of {base_algorithm_cls} or a list/tuple of {base_algorithm_cls} subclasses with the same length as the number of scales {n_scales}

check_sampling

menpofit.checks.check_sampling(sampling, n_scales)

Function that checks the value of a *sampling* parameter defined for multiple scales. It must be *int* or *ndarray* or *list* of those.

Parameters

- **sampling** (*int* or *ndarray* or *list* of those) The value to check.
- n scales (int) The number of scales.

Returns sampling (*list* of *int* or *ndarray*) – The list of values per scale.

Raises

- ValueError A sampling list can only contain 1 element or {n_scales} elements
- **ValueError** sampling can be an integer or ndarray, a integer or ndarray list containing 1 or {n_scales} elements or None

check_graph

menpofit.checks.check_graph(graph, graph_types, param_name, n_scales)

Checks the provided graph per pyramidal level. The graph must be a subclass of graph_types or a list of those.

Parameters

- **graph** (*graph* or *list* of *graph* types) The graph argument to check.
- **graph_types** (*graph* or *tuple* of *graphs*) The *tuple* of allowed graph types.
- param_name (str) The name of the graph parameter.
- **n_scales** (*int*) The number of pyramidal levels.

Returns graph (*list* of graph types) – The graph per scale in a *list*.

Raises

- **ValueError** {param_name} must be a list of length equal to the number of scales.
- **ValueError** {param_name} must be a list of {graph_types_str}. {} given instead.

2.2.3 menpofit.differentiable

Differentiable Abstract Classes

Objects that are able to compute their own derivatives.

DL

```
class menpofit.differentiable.DL
```

Bases: object

Object that is able to take its own derivative with respect to landmark changes.

abstract d_dl (points)

The derivative of this spatial object with respect to spatial changes in anchor landmark points or centres, evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

d_dl ((n_points, n_centres, n_dims) *ndarray*) - The Jacobian wrt landmark changes.

d_dl[i, k, m] is the scalar differential change that the any dimension of the i'th point experiences due to a first order change in the m'th dimension of the k'th landmark point.

Note that at present this assumes that the change in every dimension is equal.

DP

```
class menpofit.differentiable.DP
```

Bases: object

Object that is able to take its own derivative with respect to the parametrisation.

The parametrisation of objects is typically defined by the *menpo.base.Vectorizable* interface. As a result, DP is a mix-in that should be inherited along with *menpo.base.Vectorizable*.

abstract d_dp (points)

The derivative of this spatial object with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

 $\mathbf{d}_{\mathbf{d}}\mathbf{p}$ ((n_points, n_parameters, n_dims) ndarray) — The Jacobian with respect to the parametrisation.

d_dp[i, j, k] is the scalar differential change that the k'th dimension of the i'th point experiences due to a first order change in the j'th scalar in the parametrisation vector.

DX

```
class menpofit.differentiable.DX
```

Bases: object

Object that is able to take its own derivative with respect to spatial changes.

```
abstract d_dx (points)
```

The first order derivative of this spatial object with respect to spatial changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

d_dx ((n_points, n_dims, n_dims) *ndarray*) - The Jacobian wrt spatial changes.

 $d_dx[i, j, k]$ is the scalar differential change that the j'th dimension of the i'th point experiences due to a first order change in the k'th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

2.2.4 menpofit.error

Normalisers

Functions that compute a metric which can be used to normalise the error between two shapes.

Bounding Box Normalisers

bb area

menpofit.error.bb_area(shape)

Computes the area of the bounding box of the provided shape, i.e.

hw

where h and w are the height and width of the bounding box.

Parameters shape (menpo.shape.PointCloud or subclass) – The input shape.

Returns bb_area (*float*) – The area of the bounding box.

bb perimeter

menpofit.error.bb perimeter(shape)

Computes the perimeter of the bounding box of the provided shape, i.e.

$$2(h+w)$$

where h and w are the height and width of the bounding box.

Parameters shape (menpo.shape.PointCloud or subclass) – The input shape.

Returns bb_perimeter (*float*) – The perimeter of the bounding box.

bb_avg_edge_length

menpofit.error.bb_avg_edge_length(shape)

Computes the average edge length of the bounding box of the provided shape, i.e.

$$\frac{h+w}{2} = \frac{2h+2w}{4}$$

where h and w are the height and width of the bounding box.

Parameters shape (menpo.shape.PointCloud or subclass) – The input shape.

Returns bb_avg_edge_length (*float*) – The average edge length of the bounding box.

bb_diagonal

menpofit.error.bb_diagonal(shape)

Computes the diagonal of the bounding box of the provided shape, i.e.

$$\sqrt{h^2+w^2}$$

where h and w are the height and width of the bounding box.

Parameters shape (*menpo.shape.PointCloud* or *subclass*) – The input shape.

Returns bb_diagonal (*float*) – The diagonal of the bounding box.

Distance Normalisers

distance two indices

menpofit.error.distance_two_indices(index1, index2, shape)

Computes the Euclidean distance between two points of a shape, i.e.

$$\sqrt{(s_{i,x}-s_{j,x})^2+(s_{i,y}-s_{j,y})^2}$$

where $s_{i,x}$, $s_{i,y}$ are the x and y coordinates of the i'th point (index1) and $s_{j,x}$, $s_{j,y}$ are the x and y coordinates of the j'th point (index2).

Parameters

- index1 (int) The index of the first point.
- index2 (int) The index of the second point.
- **shape** (*menpo.shape.PointCloud*) The input shape.

Returns distance_two_indices (*float*) – The Euclidean distance between the points.

Errors

Functions that compute the error between two shapes.

Root Mean Square Error

root_mean_square_error

menpofit.error.root_mean_square_error(shape, gt_shape)

Computes the root mean square error between two shapes, i.e.

$$\sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - s_i^*)^2}$$

where s_i and s_i^* are the coordinates of the *i*'th point of the final and ground truth shapes, and N is the total number of points.

Parameters

- shape (menpo.shape.PointCloud) The input shape (e.g. the final shape of a fitting procedure).
- gt_shape (menpo.shape.PointCloud) The ground truth shape.

Returns root mean square error (*float*) – The root mean square error.

root mean square bb normalised error

Computes the root mean square error between two shapes normalised by a measure based on the ground truth shape's bounding box, i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - s_i^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. s_i and s_i^* are the coordinates of the i'th point of the final and ground truth shapes, and N is the total number of points. Finally, $\mathcal{N}(s^*)$ is a normalising function that returns a measure based on the ground truth shape's bounding box.

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure).
- gt_shape (menpo.shape.PointCloud) The ground truth shape.
- norm_shape (menpo.shape.PointCloud or None, optional) The shape to be used to compute the normaliser. If None, then the ground truth shape is used.
- norm_type ({'area', 'perimeter', 'avg_edge_length', 'diagonal'}, optional) The type of the normaliser. Possible options are:

Method	Description
bb_area	Area of <i>norm_shape</i> 's bounding box
bb_perimeter	Perimeter of <i>norm_shape</i> 's bounding box
bb_avg_edge_length	Average edge length of <i>norm_shape</i> 's bbox
bb_diagonal	Diagonal of <i>norm_shape</i> 's bounding box

Returns error (*float*) – The computed root mean square normalised error.

root_mean_square_distance_normalised_error

menpofit.error.root_mean_square_distance_normalised_error(shape, gt_shape, distance_norm f)

Computes the root mean square error between two shapes normalised by a distance measure between two shapes, i.e.

$$\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s, s^*)}$$

where

$$\mathcal{F}(s, s^*) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - s_i^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. s_i and s_i^* are the coordinates of the i'th point of the final and ground truth shapes, and N is the total number of points. Finally, $\mathcal{N}(s, s^*)$ is a normalising function based on a distance metric between the two shapes.

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure).
- gt_shape (menpo.shape.PointCloud) The ground truth shape.
- distance_norm_f (callable) The function to be used for computing the normalisation distance metric.

Returns error (*float*) – The computed root mean square normalised error.

root mean square distance indexed normalised error

Computes the root mean square error between two shapes normalised by the distance measure between two points of the ground truth shape, i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (s_i - s_i^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. s_i and s_i^* are the coordinates of the i'th point of the final and ground truth shapes, and N is the total number of points. Finally, $\mathcal{N}(s^*)$ is a normalising function that returns the distance between two points of the ground truth shape.

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure).
- gt_shape (menpo.shape.PointCloud) The ground truth shape.

- index1 (int) The index of the first point.
- index2 (int) The index of the second point.

Returns error (*float*) – The computed root mean square normalised error.

Euclidean Distance Error

euclidean error

menpofit.error.euclidean_error(shape, gt_shape)

Computes the Euclidean error between two shapes, i.e.

$$\frac{1}{N} \sum_{i=1}^{N} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape and N is the total number of points.

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure).
- gt_shape (menpo.shape.PointCloud) The ground truth shape.

Returns root_mean_square_error (*float*) – The Euclidean error.

euclidean_bb_normalised_error

Computes the Euclidean error between two shapes normalised by a measure based on the ground truth shape's bounding box, i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape and N is the total number of points. Finally, $\mathcal{N}(s^*)$ is a normalising function that returns a measure based on the ground truth shape's bounding box.

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure)
- gt_shape (menpo.shape.PointCloud) The ground truth shape.
- norm_shape (menpo.shape.PointCloud or None, optional) The shape to be used to compute the normaliser. If None, then the ground truth shape is used.

• norm_type ({'area', 'perimeter', 'avg_edge_length', 'diagonal'}, optional) - The type of the normaliser. Possible options are:

Method	Description
bb_area	Area of <i>norm_shape</i> 's bounding box
bb_perimeter	Perimeter of <i>norm_shape</i> 's bounding box
bb_avg_edge_length	Average edge length of <i>norm_shape</i> 's bbox
bb_diagonal	Diagonal of <i>norm_shape</i> 's bounding box

Returns error (*float*) – The computed Euclidean normalised error.

euclidean_distance_normalised_error

menpofit.error.euclidean_distance_normalised_error(shape, gt_shape, distance_norm f)

Computes the Euclidean error between two shapes normalised by a distance measure between two shapes, i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s,s^*)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x},s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*,s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape and N is the total number of points. Finally, $\mathcal{N}(s,s^*)$ is a normalising function based on a distance metric between the two shapes.

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure).
- gt_shape (menpo.shape.PointCloud) The ground truth shape.
- distance_norm_f (callable) The function to be used for computing the normalisation distance metric.

Returns error (*float*) – The computed Euclidean normalised error.

euclidean distance indexed normalised error

Computes the Euclidean error between two shapes normalised by the distance measure between two points of the ground truth shape, i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape and N is the total number of points. Finally, $\mathcal{N}(s^*)$ is a normalising function that returns the distance between two points of the ground truth shape.

Parameters

- shape (menpo.shape.PointCloud) The input shape (e.g. the final shape of a fitting procedure).
- gt_shape (menpo.shape.PointCloud) The ground truth shape.
- index1 (int) The index of the first point.
- index2 (int) The index of the second point.

Returns error (*float*) – The computed Euclidean normalised error.

Statistical Measures

Functions that compute statistical measures given a set of errors for multiple images.

compute cumulative error

```
menpofit.error.compute_cumulative_error(errors, bins)
```

Computes the values of the Cumulative Error Distribution (CED).

Parameters

- **errors** (*list* of *float*) The *list* of errors per image.
- **bins** (*list* of *float*) The values of the error bins centers at which the CED is evaluated.

Returns ced (*list* of *float*) – The computed CED.

area_under_curve_and_failure_rate

```
menpofit.error.area_under_curve_and_failure_rate(errors, step_error, max_error, min_error=0.0)
```

Computes the Area Under the Curve (AUC) and Failure Rate (FR) of a given Cumulative Distribution Error (CED).

Parameters

- **errors** (*list* of *float*) The *list* of errors per image.
- **step_error** (*float*) The sampling step of the error bins of the CED.
- max_error (*float*) The maximum error value of the CED.
- min_error (*float*) The minimum error value of the CED.

Returns

- auc (*float*) The Area Under the Curve value.
- **fr** (*float*) The Failure Rate value.

mad

```
menpofit.error.mad(errors)
```

Computes the Median Absolute Deviation of a set of errors.

Parameters errors (*list* of *float*) – The *list* of errors per image.

Returns mad (*float*) – The median absolute deviation value.

compute_statistical_measures

```
menpofit.error.compute_statistical_measures(errors, step_error, max_error, min error=0.0)
```

Computes various statistics given a set of errors that correspond to multiple images. It can also deal with multiple sets of errors that correspond to different methods.

Parameters

- **errors** (*list* of *float* or *list* of *list* of *float*) The *list* of errors per image. You can provide a *list* of *lists* for the errors of multiple methods.
- **step_error** (*float*) The sampling step of the error bins of the CED for computing the Area Under the Curve and the Failure Rate.
- max_error (float) The maximum error value of the CED for computing the Area Under the Curve and the Failure Rate.
- min_error (float) The minimum error value of the CED for computing the Area Under the Curve and the Failure Rate.

Returns

- **mean** (*float* or *list* of *float*) The mean value.
- **mean** (*float* or *list* of *float*) The standard deviation.
- **median** (*float* or *list* of *float*) The median value.
- mad (*float* or *list* of *float*) The mean absolute deviation value.
- max (*float* or *list* of *float*) The maximum value.
- auc (float or list of float) The area under the curve value.
- **fr** (*float* or *list* of *float*) The failure rate value.

Object-Specific Errors

Error functions for specific objects.

Face

bb_avg_edge_length_68_euclidean_error

menpofit.error.bb_avg_edge_length_68_euclidean_error(shape, gt_shape)

Computes the Euclidean error based on 68 points normalised by the average edge length of the 68-point ground truth shape's bounding box, i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{68} \sum_{i=1}^{68} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape. Finally, $\mathcal{N}(s^*)$ is a normalising function that returns the average edge length of the bounding box of the 68-point ground truth shape ($bb_avq_edge_length$).

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure). It must have 68 points.
- gt_shape (menpo.shape.PointCloud) The ground truth shape. It must have 68 points.

Returns normalised_error (*float*) – The computed Euclidean normalised error.

Raises

- ValueError Final shape must have 68 points
- ValueError Ground truth shape must have 68 points

bb avg edge length 49 euclidean error

menpofit.error.bb_avg_edge_length_49_euclidean_error(shape, gt_shape)

Computes the Euclidean error based on 49 points normalised by the average edge length of the 68-point ground truth shape's bounding box, i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{49} \sum_{i=1}^{49} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape. Finally, $\mathcal{N}(s^*)$ is a normalising function that returns the average edge length of the bounding box of the 68-point ground truth shape ($bb_avg_edge_length$).

Parameters

• **shape** (*menpo.shape.PointCloud*) – The input shape (e.g. the final shape of a fitting procedure). It must have 68 or 66 or 51 or 49 points.

• gt_shape (menpo.shape.PointCloud) - The ground truth shape. It must have 68 points.

Returns normalised_error (*float*) – The computed Euclidean normalised error.

Raises

- ValueError Final shape must have 68 or 51 or 49 points
- ValueError Ground truth shape must have 68 points

mean pupil 68 error

menpofit.error.mean pupil 68 error(shape, gt shape)

Computes the Euclidean error based on 68 points normalised with the distance between the mean eye points (pupils), i.e.

$$\frac{\mathcal{F}(s, s^*)}{\mathcal{N}(s)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{68} \sum_{i=1}^{68} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape. Finally, $\mathcal{N}(s)$ is the distance between the mean eye points (pupils).

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure). It must have 68 points.
- gt_shape (menpo.shape.PointCloud) The ground truth shape. It must have 68 points.

Returns normalised error (*float*) – The computed normalised Euclidean error.

Raises

- ValueError Final shape must have 68 points
- ValueError Ground truth shape must have 68 points

mean_pupil_49_error

menpofit.error.mean_pupil_49_error(shape, gt_shape)

Computes the euclidean error based on 49 points normalised with the distance between the mean eye points (pupils), i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{49} \sum_{i=1}^{49} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape. Finally, $\mathcal{N}(s)$ is the distance between the mean eye points (pupils).

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure). It must have either 68 or 66 or 51 or 49 points.
- gt_shape (menpo.shape.PointCloud) The ground truth shape. It must have either 68 or 66 or 51 or 49 points.

Returns normalised_error (*float*) – The computed normalised Euclidean error.

Raises

- ValueError Final shape must have 68 or 66 or 51 or 49 points
- ValueError Ground truth shape must have 68 or 66 or 51 or 49 points

outer_eye_corner_68_euclidean_error

menpofit.error.outer_eye_corner_68_euclidean_error(shape, gt_shape)

Computes the Euclidean error based on 68 points normalised with the distance between the mean eye points (pupils), i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{68} \sum_{i=1}^{68} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape. Finally, $\mathcal{N}(s^*)$ is the distance between the 36-th and 45-th points.

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure). It must have 68 points.
- gt_shape (menpo.shape.PointCloud) The ground truth shape. It must have 68 points.

Returns normalised_error (*float*) – The computed normalised Euclidean error.

Raises

- ValueError Final shape must have 68 points
- ValueError Ground truth shape must have 68 points

outer_eye_corner_51_euclidean_error

menpofit.error.outer_eye_corner_51_euclidean_error(shape, gt_shape)

Computes the Euclidean error based on 51 points normalised with the distance between the mean eye points (pupils), i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{51} \sum_{i=1}^{51} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape. Finally, $\mathcal{N}(s^*)$ is the distance between the 19-th and 28-th points.

Parameters

- shape (menpo.shape.PointCloud) The input shape (e.g. the final shape of a fitting procedure). It must 68 or 51 points.
- gt_shape (menpo.shape.PointCloud) The ground truth shape. It must have 68 or 51 points.

Returns normalised_error (*float*) – The computed normalised Euclidean error.

Raises

- ValueError Final shape must have 68 or 51 points
- ValueError Ground truth shape must have 68 or 51 points

outer_eye_corner_49_euclidean_error

menpofit.error.outer_eye_corner_49_euclidean_error(shape, gt_shape)

Computes the Euclidean error based on 49 points normalised with the distance between the mean eye points (pupils), i.e.

$$\frac{\mathcal{F}(s,s^*)}{\mathcal{N}(s^*)}$$

where

$$\mathcal{F}(s, s^*) = \frac{1}{49} \sum_{i=1}^{49} \sqrt{(s_{i,x} - s_{i,x}^*)^2 + (s_{i,y} - s_{i,y}^*)^2}$$

where s and s^* are the final and ground truth shapes, respectively. $(s_{i,x}, s_{i,y})$ are the x and y coordinates of the i'th point of the final shape, $(s_{i,x}^*, s_{i,y}^*)$ are the x and y coordinates of the i'th point of the ground truth shape. Finally, $\mathcal{N}(s^*)$ is the distance between the 19-th and 28-th points.

Parameters

- **shape** (*menpo.shape.PointCloud*) The input shape (e.g. the final shape of a fitting procedure). It must 68 or 66 or 51 or 49 points.
- gt_shape (menpo.shape.PointCloud) The ground truth shape. It must have 68 or 66 or 51 or 49 points.

Returns normalised_error (*float*) – The computed normalised Euclidean error.

Raises

- ValueError Final shape must have 68 or 66 or 51 or 49 points
- ValueError Ground truth shape must have 68 or 66 or 51 or 49 points

2.2.5 menpofit.fitter

Fitter Classes

MultiScaleNonParametricFitter

Bases: object

Class for defining a multi-scale fitter for a non-parametric fitting method, i.e. a method that does not optimise over a parametric shape model.

Parameters

- **scales** (*list* of *int* or *float*) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale.
- **reference_shape** (*menpo.shape.PointCloud*) The reference shape that will be used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of the reference shape.
- holistic_features (*list* of *closure*) The features that will be extracted from the input image at each scale. They must provided in ascending order, i.e. from lowest to highest scale.
- algorithms (list of class) The list of algorithm objects that will perform the fitting per scale.
- **fit_from_bb** (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- **gt_shape** (*menpo.shape.PointCloud*, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- kwargs (dict, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type list of closure

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type *list* of *int* or *float*

MultiScaleParametricFitter

Bases: MultiScaleNonParametricFitter

Class for defining a multi-scale fitter for a parametric fitting method, i.e. a method that optimises over the parameters of a statistical shape model.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimi-

sation. This step takes place at each scale and it is not considered as an iteration, thus it is not counted for the provided *max iters*.

Parameters

- scales (*list* of *int* or *float*) The scale value of each scale. They must provided in ascending order, i.e. from lowest to highest scale.
- **reference_shape** (*menpo.shape.PointCloud*) The reference shape that will be used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of the reference shape.
- holistic_features (*list* of *closure*) The features that will be extracted from the input image at each scale. They must provided in ascending order, i.e. from lowest to highest scale.
- algorithms (list of class) The list of algorithm objects that will perform the fitting per scale.
- **fit_from_bb** (*image*, *bounding_box*, *max_iters*=20, *gt_shape*=None, *return_costs*=False, **kwargs) Fits the multi-scale fitter to an image given an initial bounding box.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

fit_from_shape (image, initial_shape, max_iters=20, gt_shape=None, return_costs=False, **kwargs)

Fits the multi-scale fitter to an image given an initial shape.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (menpo.shape.PointCloud) The initial shape estimate from which the fitting procedure will start.

- max_iters (int or list of int, optional) The maximum number of iterations. If int, then it specifies the maximum number of iterations over all scales. If list of int, then specifies the maximum number of iterations per scale.
- gt_shape (menpo.shape.PointCloud, optional) The ground truth shape associated to the image.
- return_costs (bool, optional) If True, then the cost function values will be computed during the fitting procedure. Then these cost values will be assigned to the returned fitting_result. Note that the costs computation increases the computational cost of the fitting. The additional computation cost depends on the fitting method. Only use this option for research purposes.
- **kwargs** (*dict*, optional) Additional keyword arguments that can be passed to specific implementations.

Returns fitting_result (MultiScaleNonParametricIterativeResult or subclass) — The multi-scale fitting result containing the result of the fitting procedure.

property holistic_features

The features that are extracted from the input image at each scale in ascending order, i.e. from lowest to highest scale.

Type list of closure

property n_scales

Returns the number of scales.

Type int

property reference_shape

The reference shape that is used to normalise the size of an input image so that the scale of its initial fitting shape matches the scale of this reference shape.

Type menpo.shape.PointCloud

property scales

The scale value of each scale in ascending order, i.e. from lowest to highest scale.

Type list of int or float

Perturb Functions

Collection of functions that perform a kind of perturbation on a shape or bounding box.

align_shape_with_bounding_box

```
menpofit.fitter.align_shape_with_bounding_box (shape, bounding_box, align-
ment_transform_cls=<class
'menpo.transform.homogeneous.similarity.AlignmentSimilarity'>,
**kwargs)
```

Aligns the provided shape with the bounding box using a particular alignment transform.

Parameters

- **shape** (*menpo.shape.PointCloud*) The shape instance used in the alignment.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The bounding box instance used in the alignment.

• alignment_transform_cls (menpo.transform.Alignment, optional) — The class of the alignment transform used to perform the alignment.

Returns noisy_shape (menpo.shape.PointCloud) – The noisy shape

generate_perturbations_from_gt

```
menpofit.fitter.generate_perturbations_from_gt (images, n_perturbations, perturb_func, gt_group=None, bb_group_glob=None, verbose=False)
```

Function that returns a callable that generates perturbations of the bounding boxes of the provided images.

Parameters

- images (list of menpo.image.Image) The list of images.
- n_perturbations (int) The number of perturbed shapes to be generated per image.
- perturb_func (callable) The function that will be used for generating the perturbations.
- **gt_group** (*str*) The group of the ground truth shapes attached to the images.
- **bb_group_glob** (*str*) The group of the bounding boxes attached to the images.
- **verbose** (*bool*, optional) If True, then progress information is printed.

Returns generated_bb_func (*callable*) – The function that generates the perturbations.

noisy_alignment_similarity_transform

```
menpofit.fitter.noisy_alignment_similarity_transform(source, noise_type='uniform', noise_percentage=0.1, allow alignment rotation=False)
```

Constructs and perturbs the optimal similarity transform between the source and target shapes by adding noise to its parameters.

Parameters

- source (menpo.shape.PointCloud) The source pointcloud instance used in the alignment
- target (menpo.shape.PointCloud) The target pointcloud instance used in the alignment
- noise_type ({ 'uniform', 'gaussian'}, optional) The type of noise to be added.
- noise_percentage (float in (0, 1) or list of len 3, optional) The standard percentage of noise to be added. If float, then the same amount of noise is applied to the scale, rotation and translation parameters of the optimal similarity transform. If list of float it must have length 3, where the first, second and third elements denote the amount of noise to be applied to the scale, rotation and translation parameters, respectively.
- allow_alignment_rotation (*bool*, optional) If False, then the rotation is not considered when computing the optimal similarity transform between source and target.

Returns noisy_alignment_similarity_transform (*menpo.transform.Similarity*) – The noisy Similarity Transform between source and target.

noisy_shape_from_bounding_box

```
menpofit.fitter.noisy_shape_from_bounding_box(shape, bounding_box, noise_type='uniform', noise_percentage=0.05, allow alignment rotation=False)
```

Constructs and perturbs the optimal similarity transform between the bounding box of the source shape and the target bounding box, by adding noise to its parameters. It returns the noisy version of the provided shape.

Parameters

- **shape** (*menpo.shape.PointCloud*) The source pointcloud instance used in the alignment. Note that the bounding box of the shape will be used.
- bounding_box (menpo.shape.PointDirectedGraph) The target bounding box instance used in the alignment
- noise_type ({ 'uniform', 'gaussian'}, optional) The type of noise to be added.
- noise_percentage (float in (0, 1) or list of len 3, optional) The standard percentage of noise to be added. If float, then the same amount of noise is applied to the scale, rotation and translation parameters of the optimal similarity transform. If list of float it must have length 3, where the first, second and third elements denote the amount of noise to be applied to the scale, rotation and translation parameters, respectively.
- allow_alignment_rotation (*bool*, optional) If False, then the rotation is not considered when computing the optimal similarity transform between source and target.

Returns noisy shape (*menpo.shape.PointCloud*) – The noisy shape.

noisy_shape_from_shape

```
menpofit.fitter.noisy_shape_from_shape(reference_shape, shape, noise_type='uniform', noise_percentage=0.05, allow alignment rotation=False)
```

Constructs and perturbs the optimal similarity transform between the provided reference shape and the target shape, by adding noise to its parameters. It returns the noisy version of the reference shape.

Parameters

- **reference_shape** (*menpo.shape.PointCloud*) The source reference shape instance used in the alignment.
- shape (menpo.shape.PointDirectedGraph) The target shape instance used in the alignment
- noise_type ({ 'uniform', 'gaussian'}, optional) The type of noise to be added.
- noise_percentage (float in (0, 1) or list of len 3, optional) The standard percentage of noise to be added. If float, then the same amount of noise is applied to the scale, rotation and translation parameters of the optimal similarity transform. If list of float it must have length 3, where the first, second and third elements denote the amount of noise to be applied to the scale, rotation and translation parameters, respectively.
- allow_alignment_rotation (*bool*, optional) If False, then the rotation is not considered when computing the optimal similarity transform between source and target.

Returns noisy_reference_shape (*menpo.shape.PointCloud*) – The noisy reference shape.

noisy target alignment transform

Constructs the optimal alignment transform between the source and a noisy version of the target obtained by adding white noise to each of its points.

Parameters

- source (menpo.shape.PointCloud) The source pointcloud instance used in the alignment
- target (menpo.shape.PointCloud) The target pointcloud instance used in the alignment
- alignment_transform_cls (*menpo.transform.Alignment*, optional) The alignment transform class used to perform the alignment.
- noise_std (*float* or *list* of *float*, optional) The standard deviation of the white noise to be added to each one of the target points. If *float*, then the same standard deviation is used for all points. If *list*, then it must define a value per point.

Returns noisy_transform (menpo.transform.Alignment) – The noisy Similarity Transform

2.2.6 menpofit.io

Menpofit includes the ability to save and load pre-trained models for specific tasks. This module contains code for pickling down, downloading, and restoring fitters efficiently.

If you make use of one of menpofit's pre-trained models, you will find that the type that is provided to you is the PickleWrappedFitter. See it's documentation to understand it's purpose and how you can effectively use it.

PickleWrappedFitter

Wrapper around a menpofit fitter so that we can a) efficiently pickle it and b) parametrize over both the fitter construction and the fit methods (e.g. .fit_from_bb() and .fit_from_shape())

Pickling menpofit fitters is a little tricky for a two reasons. Firstly, on construction of a fitter from a deformable model some amount of pre-processing takes place which allocates potentially large arrays. To ship a compact model we would therefore rather delay the construction of the fitter until load time on the client.

If this was the only issue, we could achieve this by simply saving a partial over the fitter constructor with all the args and kwargs the fitter constructor takes - after loading the pickle, invoking the partial with no args (it's parameters being fully specified) would return the fitter and all would be well.

However, we also may want to choose **fit-time** parameters for the fitter for optimal usage, (for instance, a choice over the max_iters kwarg that we know to be efficient). This leaves us with a problem, as now we need to have some entity that can store state which we can pass to both the fitter and to the resulting fitters methods on the client at unpickle time.

This class is the solution to this problem. To use, you should **pickle down a partial over this class** specifying all arguments and kwargs needed for the fitter constructor and for the fit methods.

At load time, menpofit will invoke the partial, returning this object instantiated. This offers the same API as a menpofit fitter, and so can be used transparently to fit. If you wish to access the original fitter (without fit parameter customization) this can be accessed as the *wrapped_fitter* property.

Parameters

- **fitter_cls** (*Fitter*) A menpofit fitter class that will be constructed at unpickle time, e.g. LucasKanadeAAMFitter
- **fitter_args** (*tuple*) A tuple of all args that need to be passed to fitter_cls at construction time e.g. (aam,)
- **fitter_kwargs** (*dict*) A dictionary of kwargs that will to be passed to fitter_cls at construction time e.g. { 'lk_algorithm_cls': WibergInverseCompositional }
- fit_from_bb_kwargs (dict, e.g. { max_iters: [25, 5] }) A dictionary of kwargs that will to be passed to the wrapped fitter's fit_from_bb method at fit time. These in effect change the defaults that the original fitter offered, but can still be overridden at call time (e.g. self.fit_from_bb(image, bbox, max_iters=[50, 50]) would take precedence over the max_iters in the above example)
- fit_from_shape_kwargs (dict, e.g. { max_iters: [25, 5] }) A dictionary of kwargs that will to be passed to the wrapped fitter's fit_from_shape method at fit time. These in effect change the defaults that the original fitter offered, but can still be overridden at call time (e.g. self.fit_from_shape(image, shape, max_iters=[50, 50]) would take precedence over the max_iters in the above example)
- **image_preprocess** (*callable* or None, optional) A pre-processing function to apply on the test image before fitting. The default option converts the image to greyscale. The function needs to have the following signature:

```
new_image, transform = image_preprocess(image, pointcloud)
```

where *new_image* is the pre-processed image and *transform* is the *menpo.transform.Homogeneous* object that was applied on the image. If None, then no pre-processing is performed.

Examples

(continues on next page)

(continued from previous page)

fit from bb (*image*, *bounding box*, **kwargs)

Fits the fitter to an image given an initial bounding box, using the optimal parameters that we chosen for this pickled fitter.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- **bounding_box** (*menpo.shape.PointDirectedGraph*) The initial bounding box from which the fitting procedure will start. Note that the bounding box is used in order to align the model's reference shape.
- **kwargs** (*dict*, optional) Other kwargs to override the optimal defaults. See the documentation for .fit_from_bb() on the type of *self.wrapped_fitter* to see what can be provided here.

Returns fitting_result (FittingResult or subclass) – The fitting result containing the result of the fitting procedure.

fit_from_shape (image, initial_shape, **kwargs)

Fits the fitter to an image given an initial shape, using the optimal parameters that we chosen for this pickled fitter.

Parameters

- image (menpo.image.Image or subclass) The image to be fitted.
- initial_shape (*menpo.shape.PointCloud*) The initial shape estimate from which the fitting procedure will start.
- **kwargs** (dict) Other kwargs to override the optimal defaults. See the documentation for .fit_from_shape() on the type of *self.wrapped_fitter* to see what can be provided here

Returns fitting_result (FittingResult or subclass) — The fitting result containing the result of the fitting procedure.

2.2.7 menpofit.math

Regression

IRLRegression

```
class menpofit.math.IRLRegression (alpha=0, bias=True, incrementable=False)

Bases: object
```

Class for training and applying Incremental Regularized Linear Regression.

Parameters

- alpha (*float*, optional) The regularization parameter of the features.
- bias (bool, optional) If True, a bias term is used.
- incrementable (*bool*, optional) If True, then the regression model will have the ability to get incremented.

increment(X, Y)

Incrementally update the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) *ndarray*) The array of target vectors.

Raises ValueError - Model is not incrementable

predict (x)

Makes a prediction using the trained regression model.

```
Parameters x ((n_features,) ndarray) – The input feature vector.
```

Returns prediction ((n_dims,) *ndarray*) – The prediction vector.

train(X, Y)

Train the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) *ndarray*) The array of target vectors.

IIRLRegression

```
class menpofit.math.IIRLRegression(alpha=0, bias=False, alpha2=0)
    Bases: IRLRegression
```

Class for training and applying Indirect Incremental Regularized Linear Regression.

Parameters

- $alpha\ (\mathit{float}, optional)$ The regularization parameter.
- bias (bool, optional) If True, a bias term is used.
- alpha2 (*float*, optional) The regularization parameter of the Hessian.

increment(X, Y)

Incrementally update the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) *ndarray*) The array of target vectors.

Raises ValueError - Model is not incrementable

predict (x)

Makes a prediction using the trained regression model.

Parameters x ((n_features,) *ndarray*) – The input feature vector.

Returns prediction ((n_dims,) *ndarray*) – The prediction vector.

train(X, Y)

Train the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) *ndarray*) The array of target vectors.

PCRRegression

class menpofit.math.PCRRegression(variance=None, bias=True)

Bases: object

Class for training and applying Multivariate Linear Regression using Principal Component Regression.

Parameters

- variance (float or None, optional) The SVD variance.
- bias (bool, optional) If True, a bias term is used.

increment(X, Y)

Incrementally update the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) *ndarray*) The array of target vectors.

Raises ValueError - Model is not incrementable

predict (x)

Makes a prediction using the trained regression model.

Parameters \mathbf{x} ((n_features,) ndarray) – The input feature vector.

Returns prediction ((n_dims,) *ndarray*) – The prediction vector.

train(X, Y)

Train the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) *ndarray*) The array of target vectors.

OptimalLinearRegression

```
class menpofit.math.OptimalLinearRegression(variance=None, bias=True)
    Bases: object
```

Class for training and applying Multivariate Linear Regression using optimal reconstructions.

Parameters

- variance (float or None, optional) The SVD variance.
- bias (bool, optional) If True, a bias term is used.

increment(X, Y)

Incrementally update the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) ndarray) The array of target vectors.

Raises ValueError – Model is not incrementable

predict (x)

Makes a prediction using the trained regression model.

```
Parameters x ((n_features,) ndarray) – The input feature vector.
```

Returns prediction ((n_dims,) *ndarray*) – The prediction vector.

```
train(X, Y)
```

Train the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) *ndarray*) The array of target vectors.

OPPRegression

```
class menpofit.math.OPPRegression(bias=True, whiten=False)
    Bases: object
```

Class for training and applying Multivariate Linear Regression using Orthogonal Procrustes Problem reconstructions.

Parameters

- bias (bool, optional) If True, a bias term is used.
- whiten (*bool*, optional) Whether to use a whitened PCA model.

increment(X, Y)

Incrementally update the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- **Y** ((n_dims, n_samples) *ndarray*) The array of target vectors.

Raises ValueError - Model is not incrementable

predict(x)

Makes a prediction using the trained regression model.

Parameters x ((n_features,) *ndarray*) – The input feature vector.

Returns prediction ((n_dims,) *ndarray*) – The prediction vector.

train(X, Y)

Train the regression model.

Parameters

- **X**((n_features, n_samples) *ndarray*) The array of feature vectors.
- Y ((n_dims, n_samples) *ndarray*) The array of target vectors.

Correlation Filters

mccf

menpofit.math.mccf $(X, y, l=0.01, boundary='constant', crop_filter=True)$ Multi-Channel Correlation Filter (MCCF).

Parameters

- X((n_images, n_channels, image_h, image_w) ndarray) The training images.
- y((1, response_h, response_w) ndarray) The desired response.
- 1 (*float*, optional) Regularization parameter.
- boundary ({ 'constant', 'symmetric'}, optional) Determines how the image is padded.
- crop_filter (bool, optional) If True, the shape of the MOSSE filter is the same as the shape of the desired response. If False, the filter's shape is equal to: X[0].shape + y.shape 1

Returns

- **f** ((1, response_h, response_w) *ndarray*) Multi-Channel Correlation Filter (MCCF) filter associated to the training images.
- sXY ((N,) ndarray) The auto-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.
- sXX ((N, N) ndarray) The cross-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.

References

imccf

menpofit.math.imccf $(A, B, n_ab, X, y, l=0.01, boundary='constant', crop_filter=True, f=1.0)$ Incremental Multi-Channel Correlation Filter (MCCF)

Parameters

- **A** ((N, ndarray) The current auto-correlation array, where N = (patch h+response h-1) * (patch w+response w-1) * n channels.
- B ((N, N) ndarray) The current cross-correlation array, where N = (patch_h+response_h-1) * (patch_w+response_w-1) * n_channels.
- n_ab (int) The current number of images.
- **X** ((n_images, n_channels, image_h, image_w) *ndarray*) The training images (patches).
- y((1, response_h, response_w) ndarray) The desired response.
- 1 (*float*, optional) Regularization parameter.
- boundary ({ 'constant', 'symmetric'}, optional) Determines how the image is padded.
- crop_filter (bool, optional) If True, the shape of the MOSSE filter is the same as the shape of the desired response. If False, the filter's shape is equal to: X[0].shape + y.shape 1
- **f** ([0, 1] *float*, optional) Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally. If <1.0, more emphasis is put on the new samples.

Returns

- **f** ((1, response_h, response_w) *ndarray*) Multi-Channel Correlation Filter (MCCF) filter associated to the training images.
- sXY ((N, ndarray) The auto-correlation array, where $N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.$
- **sXX** ((N, N) *ndarray*) The cross-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.

References

mosse

menpofit.math.mosse $(X, y, l=0.01, boundary='constant', crop_filter=True)$ Minimum Output Sum of Squared Errors (MOSSE) filter.

Parameters

- **X**((n_images, n_channels, image_h, image_w) *ndarray*) The training images.
- y((1, response_h, response_w) ndarray) The desired response.
- 1 (*float*, optional) Regularization parameter.

- boundary ({ 'constant', 'symmetric'}, optional) Determines how the image is padded.
- crop_filter (bool, optional) If True, the shape of the MOSSE filter is the same as the shape of the desired response. If False, the filter's shape is equal to: X[0].shape + y.shape 1

Returns

- **f** ((1, response_h, response_w) *ndarray*) Minimum Output Sum od Squared Errors (MOSSE) filter associated to the training images.
- sXY ((N,) ndarray) The auto-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.
- sXX ((N, N) ndarray) The cross-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.

References

imosse

menpofit.math.imosse $(A, B, n_ab, X, y, l=0.01, boundary='constant', crop_filter=True, f=1.0)$ Incremental Minimum Output Sum of Squared Errors (iMOSSE) filter.

Parameters

- A ((N, ndarray) The current auto-correlation array, where $N = (patch_h + response_h 1) * (patch_w + response_w 1) * n_channels.$
- B ((N, N) ndarray) The current cross-correlation array, where N = (patch_h+response_h-1) * (patch_w+response_w-1) * n_channels.
- n_ab (int) The current number of images.
- X((n_images, n_channels, image_h, image_w) *ndarray*) The training images (patches).
- y((1, response_h, response_w) *ndarray*) The desired response.
- 1 (*float*, optional) Regularization parameter.
- boundary ({ 'constant', 'symmetric'}, optional) Determines how the image is padded.
- crop_filter (bool, optional) If True, the shape of the MOSSE filter is the same as the shape of the desired response. If False, the filter's shape is equal to: X[0].shape + y.shape 1
- **f** ([0, 1] *float*, optional) Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally. If <1.0, more emphasis is put on the new samples.

Returns

- **f** ((1, response_h, response_w) *ndarray*) Minimum Output Sum od Squared Errors (MOSSE) filter associated to the training images.
- **sXY** ((N,) *ndarray*) The auto-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.

```
• sXX ((N, N) ndarray) - The cross-correlation array, where N = (image_h+response_h-1) * (image_w+response_w-1) * n_channels.
```

References

2.2.8 menpofit.modelinstance

Abstract Classes

ModelInstance

```
class menpofit.modelinstance.ModelInstance(model)
    Bases: Targetable, Vectorizable, DP
```

Base class for creating a model that can produce a target *menpo.shape.PointCloud* and knows how to compute its own derivative with respect to its parametrisation.

Parameters model (*class*) – The trained model (e.g. *menpo.model.PCAModel*).

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

abstract d_dp (points)

The derivative of this spatial object with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

 $\mathbf{d}_{-}\mathbf{dp}$ ((n_points, n_parameters, n_dims) ndarray) – The Jacobian with respect to the parametrisation.

d_dp[i, j, k] is the scalar differential change that the k'th dimension of the i'th point experiences due to a first order change in the j'th scalar in the parametrisation vector.

from_vector (vector)

Build a new instance of the object from it's vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from vector inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns object (type (self)) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (*bool*) – If the vectorized object contains nan values.

set_target (new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and regenerate.

property n_dims

The number of dimensions of the target.

Type int

property n_parameters

The length of the vector that this object produces.

Type int

property n_points

The number of points on the target.

Type int

property n_weights

The number of parameters in the linear model.

Type int

property target

The current *menpo.shape.PointCloud* that this object produces.

Type menpo.shape.PointCloud

property weights

The weights of the model.

Type (n_weights,) ndarray

Similarity Model

similarity_2d_instance_model

```
menpofit.modelinstance.similarity 2d instance model(shape)
```

Creates a *menpo.model.MeanLinearModel* that encodes the 2D similarity transforms that can be applied on a 2D shape that consists of *n points*.

Parameters shape (*menpo.shape.PointCloud*) – The input 2D shape.

Returns model (*subclass* of *menpo.model.MeanLinearModel*) – Linear model with four components, the linear combinations of which represent the original shape under a similarity transform. The model is exhaustive (that is, all possible similarity transforms can be expressed with the model).

GlobalSimilarityModel

```
class menpofit.modelinstance.GlobalSimilarityModel(data, **kwargs)
    Bases: Targetable, Vectorizable
```

Class for creating a model that represents a global similarity transform (in-plane rotation, scaling, translation).

Parameters data (*list* of *menpo.shape.PointCloud*) – The *list* of shapes to use as training data.

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

$d_dp(_)$

Returns the Jacobian of the similarity model reshaped in order to have the standard Jacobian shape, i.e. $(n_points, n_weights, n_dims)$ which maps to $(n_features, n_components, n_dims)$ on the linear model.

Returns jacobian ((n_features, n_components, n_dims) *ndarray*) - The Jacobian of the model in the standard Jacobian shape.

from_vector(vector)

Build a new instance of the object from it's vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns object (type (self)) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (bool) – If the vectorized object contains nan values.

set_target (new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (*menpo.shape.PointCloud*) – The new target that this object should try and regenerate.

property n_dims

The number of dimensions of the spatial instance of the model.

Type int

property n_parameters

The length of the vector that this object produces.

Type int

property n_points

The number of points on the target.

Type int

property n_weights

The number of parameters in the linear model.

Type int

property target

The current *menpo.shape.PointCloud* that this object produces.

Type menpo.shape.PointCloud

property weights

The weights of the model.

Type (n_weights,) ndarray

Point Distribution Model

PDM

class menpofit.modelinstance.PDM (data, max_n_components=None)
 Bases: ModelInstance

Class for building a Point Distribution Model. It is a specialised version of ModelInstance for use with spatial data.

Parameters

- data (list of menpo.shape.PointCloud or menpo.model.PCAModel instance) If a list of menpo.shape.PointCloud, then a menpo.model.PCAModel will be trained from those training shapes. Otherwise, a trained menpo.model.PCAModel instance can be provided.
- max_n_components (*int* or None, optional) The maximum number of components that the model will keep. If None, then all the components will be kept.

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

d dp (points)

Returns the Jacobian of the similarity model reshaped in order to have the standard Jacobian shape, i.e. $(n_points, n_weights, n_dims)$ which maps to $(n_features, n_components, n_dims)$ on the linear model.

Returns jacobian ((n_features, n_components, n_dims) *ndarray*) - The Jacobian of the model in the standard Jacobian shape.

from vector(vector)

Build a new instance of the object from it's vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns object (type (self)) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has nan values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (*bool*) – If the vectorized object contains nan values.

increment (shapes, n_shapes=None, forgetting_factor=1.0, max_n_components=None, verbose=False)

Update the eigenvectors, eigenvalues and mean vector of this model by performing incremental PCA on the given samples.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) List of new shapes to update the model from.
- n_shapes (int or None, optional) If int, then shapes must be an iterator that yields n_shapes. If None, then shapes has to be a list (so we know how large the data matrix needs to be).
- **forgetting_factor** ([0.0, 1.0] *float*, optional) Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally and, hence, the results is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples. See [1] for details.
- max_n_components (*int* or None, optional) The maximum number of components that the model will keep. If None, then all the components will be kept.
- **verbose** (bool, optional) If True, then information about the progress will be printed.

References

set_target (new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and regenerate.

property n_active_components

The number of components currently in use on this model.

Type int

property n_dims

The number of dimensions of the spatial instance of the model

Type int

property n_parameters

The length of the vector that this object produces.

Type int

property n_points

The number of points on the target.

Type int

property n_weights

The number of parameters in the linear model.

Type int

property target

The current *menpo.shape.PointCloud* that this object produces.

Type menpo.shape.PointCloud

property weights

The weights of the model.

Type (n_weights,) ndarray

GlobalPDM

class menpofit.modelinstance.**GlobalPDM**(data,

global_transform_cls,

max_n_components=None)

Bases: PDM

Class for building a Point Distribution Model that also stores a Global Alignment transform. The final transform couples the Global Alignment transform to a statistical linear model, so that its weights are fully specified by both the weights of statistical model and the weights of the similarity transform.

Parameters

- data (list of menpo.shape.PointCloud or menpo.model.PCAModel instance) If a list of menpo.shape.PointCloud, then a menpo.model.PCAModel will be trained from those training shapes. Otherwise, a trained menpo.model.PCAModel instance can be provided.
- global_transform_cls (class) The Global Similarity transform class (e.g. DifferentiableAlignmentSimilarity).
- max_n_components (*int* or None, optional) The maximum number of components that the model will keep. If None, then all the components will be kept.

as vector(**kwargs)

Returns a flattened representation of the object as a single vector.

Returns vector $((N_i) ndarray)$ – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

d_dp (points)

The derivative with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns d_dp ((n_points, n_parameters, n_dims) *ndarray*) — The Jacobian with respect to the parametrisation.

from_vector (vector)

Build a new instance of the object from it's vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns object (type (self)) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (bool) – If the vectorized object contains nan values.

increment (shapes, n_shapes=None, forgetting_factor=1.0, max_n_components=None, verbose=False)

Update the eigenvectors, eigenvalues and mean vector of this model by performing incremental PCA on the given samples.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) List of new shapes to update the model from.
- n_shapes (int or None, optional) If int, then shapes must be an iterator that yields n_shapes. If None, then shapes has to be a list (so we know how large the data matrix needs to be).
- **forgetting_factor** ([0.0, 1.0] *float*, optional) Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally and, hence, the results is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples. See [1] for details.

- max_n_components (*int* or None, optional) The maximum number of components that the model will keep. If None, then all the components will be kept.
- verbose (bool, optional) If True, then information about the progress will be printed.

References

set target(new target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) - The new target that this object should try and regenerate.

property global_parameters

The parameters for the global transform.

Type ``(n_global_parameters,) *ndarray*

property n_active_components

The number of components currently in use on this model.

Type int

property n_dims

The number of dimensions of the spatial instance of the model

Type int

property n_global_parameters

The number of parameters in the *global_transform*

Type int

property n_parameters

The length of the vector that this object produces.

Type int

property n_points

The number of points on the target.

Type int

property n_weights

The number of parameters in the linear model.

Type int

property target

The current *menpo.shape.PointCloud* that this object produces.

Type menpo.shape.PointCloud

property weights

The weights of the model.

Type (n_weights,) *ndarray*

OrthoPDM

```
class menpofit.modelinstance.OrthoPDM(data, max_n_components=None)
    Bases: GlobalPDM
```

Class for building a Point Distribution Model that also stores a Global Alignment transform. The final transform couples the Global Alignment transform to a statistical linear model, so that its weights are fully specified by both the weights of statistical model and the weights of the similarity transform.

This transform (in contrast to the :map`GlobalPDM`) additionally orthonormalises both the global and the model basis against each other, ensuring that orthogonality and normalization is enforced across the unified bases.

Parameters

- data (list of menpo.shape.PointCloud or menpo.model.PCAModel instance) If a list of menpo.shape.PointCloud, then a menpo.model.PCAModel will be trained from those training shapes. Otherwise, a trained menpo.model.PCAModel instance can be provided.
- max_n_components (*int* or None, optional) The maximum number of components that the model will keep. If None, then all the components will be kept.

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

```
Returns type (self) - A copy of this object
```

d_dp (points)

The derivative with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns d_dp ((n_points, n_parameters, n_dims) *ndarray*) – The Jacobian with respect to the parametrisation.

from vector(vector)

Build a new instance of the object from it's vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns object (type (self)) – An new instance of this class.

from vector inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has nan values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (bool) – If the vectorized object contains nan values.

increment (shapes, n_shapes=None, forgetting_factor=1.0, max_n_components=None, verbose=False)

Update the eigenvectors, eigenvalues and mean vector of this model by performing incremental PCA on the given samples.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) List of new shapes to update the model from.
- n_shapes (int or None, optional) If int, then shapes must be an iterator that yields n_shapes. If None, then shapes has to be a list (so we know how large the data matrix needs to be).
- **forgetting_factor** ([0.0, 1.0] *float*, optional) Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally and, hence, the results is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples. See [1] for details.
- max_n_components (*int* or None, optional) The maximum number of components that the model will keep. If None, then all the components will be kept.
- **verbose** (*bool*, optional) If True, then information about the progress will be printed.

References

set_target (new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and regenerate.

property global_parameters

The parameters for the global transform.

Type (n_global_parameters,) ndarray

property n_active_components

The number of components currently in use on this model.

Type int

property n_dims

The number of dimensions of the spatial instance of the model

Type int

${\tt property} \ {\tt n_global_parameters}$

The number of parameters in the global_transform

Type int

property n_parameters

The length of the vector that this object produces.

Type int

property n_points

The number of points on the target.

Type int

property n_weights

The number of parameters in the linear model.

Type int

property target

The current menpo.shape.PointCloud that this object produces.

Type menpo.shape.PointCloud

property weights

The weights of the model.

Type (n_weights,) ndarray

2.2.9 menpofit.result

Basic Result

Class for defining a basic fitting result.

Result

class menpofit.result.Result (final_shape, image=None, initial_shape=None, gt_shape=None)
 Bases: object

Class for defining a basic fitting result. It holds the final shape of a fitting process and, optionally, the initial shape, ground truth shape and the image object.

Parameters

- **final_shape** (*menpo.shape.PointCloud*) The final shape of the fitting process.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- initial_shape (menpo.shape.PointCloud or None, optional) The initial shape that was provided to the fitting method to initialise the fitting process. If None, then no initial shape is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

final_error (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial error(compute error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

view (figure_id=None, render_image=True, render_final_shape=True, new_figure=False, render_initial_shape=False, render_gt_shape=False, *subplots_enabled=True*, *nels=None*, *interpolation='bilinear'*, *cmap_name=None*, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line style='-', line width=2, marker style='o', render markers=True, $marker\ size=4,$ $marker_edge_width=1.0$, render numbering=False, numbers_horizontal_align='center', numbers vertical align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render legend=True, legend title=", legend font name='sans-serif', legend font style='normal', legend font size=10, legend font weight='normal', legend marker scale=None, legend_bbox_to_anchor=(1.05. end location=2. legend border axes pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new figure (bool, optional) If True, a new figure is created.
- render_image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (*bool*, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• interpolation (See Below, optional) - The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- **final_marker_face_colour** (See Below, optional) The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) - The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour (See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **line_style** (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- render_markers (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.

• numbers_font_name (See Below, optional) - The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend_title (str, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- legend_font_size (int, optional) The font size of the legend.
- legend_font_weight (See Below, optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- **legend_bbox_to_anchor** ((*float*, *float*) *tuple*, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend horizontal spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the

Image's height. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- **figure_size** ((*float*, *float*) *tuple* or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

Iterative Result

Classes for defining an iterative fitting result.

NonParametricIterativeResult

```
class menpofit.result.NonParametricIterativeResult (shapes, initial\_shape=None, image=None, gt\_shape=None, costs=None)
```

Bases: Result

Class for defining a non-parametric iterative fitting result, i.e. the result of a method that does not optimize over a parametric shape model. It holds the shapes of all the iterations of the fitting procedure. It can optionally store the image on which the fitting was applied, as well as its ground truth shape.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of shapes per iteration. Note that the list does not include the initial shape. The last member of the list is the final shape.
- initial_shape (*menpo.shape.PointCloud* or None, optional) The initial shape from which the fitting process started. If None, then no initial shape is assigned.

- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- **gt_shape** (*menpo.shape.PointCloud* or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.
- **costs** (*list* of *float* or None, optional) The *list* of cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm. It must have the same length as *shapes*.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

displacements_stats (stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be 'mean', 'median', 'min' or 'max'

errors (compute_error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

• ValueError – Initial shape has not been set, so the initial error cannot be computed

ValueError – Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-
                                                              marker_style='o',
                    line width=2,
                                   render_markers=True,
                                                                                   marker size=4,
               marker_face_colour='b',
                                            marker_edge_colour='k',
                                                                         marker\ edge\ width=1.0,
               render axes=True,
                                          axes font name='sans-serif',
                                                                               axes font size=10,
               axes font style='normal',
                                              axes_font_weight='normal',
                                                                                axes x limits=0.0,
               axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(10, 6),
               render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the cost function evolution at each fitting iteration.
```

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'}, optional) The style of the lines.
- line width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (*colour* or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers.If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (float, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (*colour* or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (float or (float, float) or None, optional) – The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes y ticks (list or tuple or None, optional) The ticks of the y axis.
- figure size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_errors (compute_error=None,
                                       figure_id=None,
                                                          new_figure=False,
                                                                                render_lines=True,
                line colour='b',
                                      line style='-',
                                                         line width=2,
                                                                             render markers=True,
                marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
                marker\ edge\ width=1.0,
                                              render axes=True,
                                                                      axes font name='sans-serif',
                axes\_font\_size=10,
                                         axes_font_style='normal',
                                                                       axes_font_weight='normal',
                axes x limits=0.0, axes y limits=None, axes x ticks=None, axes y ticks=None,
                figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the error evolution at each fitting iteration.
```

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• marker_size (int, optional) – The size of the markers in points.

• marker_face_colour (colour or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style({'-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

render_final_shape=True, view (figure_id=None, new_figure=False, render_image=True, render_initial_shape=False, render_gt_shape=False, *subplots_enabled=True*, nels=None, interpolation='bilinear', cmap_name=None, alpha=1.0, masked=True, final_marker_face_colour='r', final_marker_edge_colour='k', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, line style='-', line width=2, marker style='o', render markers=True, $marker\ size=4,$ $marker_edge_width=1.0$, render numbering=False, numbers_horizontal_align='center', numbers vertical align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render legend=True, legend title=", legend font name='sans-serif', legend font style='normal', legend font size=10, legend font weight='normal', legend marker scale=None, end location=2. legend bbox to anchor=(1.05,legend border axes pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- **new_figure** (*bool*, optional) If True, a new figure is created.
- render_image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (*bool*, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• interpolation (See Below, optional) - The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.
- **final_marker_face_colour** (See Below, optional) The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) - The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional)—The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- line_style (str or list of str, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- render_markers (bool or list of bool, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.

• numbers_font_name (See Below, optional) - The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour (See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **legend_font_style** ({normal, italic, oblique}, optional) The font style of the legend.
- legend_font_size (int, optional) The font size of the legend.
- legend_font_weight (See Below, optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- **legend_bbox_to_anchor** ((*float*, *float*) *tuple*, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the

Image's height. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view_iterations (figure_id=None, render_image=True, new_figure=False, iters=None. *subplots_enabled=False*, channels=None, interpolation='bilinear', alpha=1.0, masked=True, render_lines=True, cmap_name=None, line style='-', line width=2, line colour=None, render markers=True, marker_edge_colour=None, *marker_face_colour=None*, marker_style='o', marker size=4, $marker_edge_width=1.0$, render numbering=False, numbers_horizontal_align='center', numbers_vertical_align='bottom', numbers_font_name='sans-serif', *numbers_font_size=10*, bers_font_style='normal', numbers_font_weight='normal', numbers font colour='k', render legend=True, legend title=". legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, *legend_location=2*, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=None, $legend_n_columns=1$, legend horizontal spacing=None, legend vertical spacing=None, legend border=True, legend_border_padding=None, legend shadow=False, legend rounded corners=False, render axes=False, axes font name='sansserif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, $axes_x_ticks=None,$ axes_y_ticks=None, figure_size=(7, 7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.shapes[1]	Iteration 1
i	self.shapes[i]	Iteration i
n_iters	self.final_shape	Final shape

- render_image (*bool*, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.

• **interpolation** (*str* (See Below), optional) – The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -., :}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_colour (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (str (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (int, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- legend_marker_scale (float, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• axes_font_size (int, optional) – The font size of the axes.

- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists) and *final_shape*.

Type list of menpo.shape.PointCloud

ParametricIterativeResult

Bases: NonParametricIterativeResult

Class for defining a parametric iterative fitting result, i.e. the result of a method that optimizes the parameters of a shape model. It holds the shapes and shape parameters of all the iterations of the fitting procedure. It can optionally store the image on which the fitting was applied, as well as its ground truth shape.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) The *list* of shapes per iteration. Note that the list does not include the initial shape. However, it includes the reconstruction of the initial shape. The last member of the list is the final shape.
- **shape_parameters** (*list* of *ndarray*) The *list* of shape parameters per iteration. Note that the list includes the parameters of the projection of the initial shape. The last member of the list corresponds to the final shape's parameters. It must have the same length as *shapes*.
- initial_shape (*menpo.shape.PointCloud* or None, optional) The initial shape from which the fitting process started. If None, then no initial shape is assigned.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.
- **costs** (*list* of *float* or None, optional) The *list* of cost per iteration. If None, then it is assumed that the cost function cannot be computed for the specific algorithm. It must have the same length as *shapes*.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

displacements_stats (stat_type='mean')

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError - type must be 'mean', 'median', 'min' or 'max'

errors (compute error=None)

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error(compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial error(compute error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

```
plot costs (figure id=None, new figure=False, render lines=True, line colour='b', line style='-
                    line_width=2, render_markers=True,
                                                              marker style='o',
                                                                                  marker size=4,
               marker face colour='b',
                                            marker edge colour='k',
                                                                       marker\ edge\ width=1.0,
               render_axes=True,
                                          axes_font_name='sans-serif',
                                                                               axes\_font\_size=10,
               axes font style='normal',
                                              axes font weight='normal',
                                                                               axes x limits=0.0,
               axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6),
               render grid=True, grid line style='--', grid line width=0.5)
     Plot of the cost function evolution at each fitting iteration.
```

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'}, optional) The style of the lines.
- line_width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker size (int, optional) The size of the markers in points.
- marker_face_colour (*colour* or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers.If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the

curves' height. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- **figure_size** ((*float*, *float*) or None, optional) The size of the figure in inches.
- render grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True,
                          line_colour='b', line_style='-', line_width=2, render_markers=True,
                          marker_style='o',
                                                                         marker_face_colour='b',
                                                  marker\_size=4,
                          marker edge colour='k',
                                                   marker\ edge\ width=1.0,
                                                                              render axes=True,
                          axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal',
                          axes font weight='normal',
                                                       axes x limits=0.0,
                                                                             axes y limits=None,
                          axes \ x \ ticks=None,
                                              axes_y_ticks=None, figure_size=(10,
                                                                                      6).
                          der grid=True, grid line style='--', grid line width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

• marker_size (int, optional) – The size of the markers in points.

• marker_face_colour (*colour* or None, optional) – The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (*float* or (*float*, *float*) or None, optional) The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves' width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

plot errors (compute error=None, figure id=None, new figure=False, render lines=True, line style='-', line width=2, render markers=True. line colour='b', marker style='o', marker size=4, marker face colour='b', marker edge colour='k', $marker_edge_width=1.0$, render_axes=True, axes_font_name='sans-serif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=0.0, axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(10, 6), render grid=True, grid line style='--', grid line width=0.5) Plot of the error evolution at each fitting iteration.

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -:, :}
```

- line width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (*colour* or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker edge width (float, optional) The width of the markers' edge.
- **render_axes** (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (*float* or (*float*, *float*) or None, optional) The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves' width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- **figure_size** ((*float*, *float*) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed_initial_error (compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed_initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a Result instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).

• pass_gt_shape (bool, optional) – If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view(figure id=None, new figure=False, render image=True, render final shape=True, render initial shape=False, render_gt_shape=False, subplots enabled=True, nels=None, interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, final marker face colour='r', final marker edge colour='k', final line colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt_marker_face_colour='y', gt_marker_edge_colour='k', gt_line_colour='y', render_lines=True, *line_style='-'*, *line_width=2*, render_markers=True, marker_style='o', marker size=4, render_numbering=False, numbers_horizontal_align='center', $marker_edge_width=1.0$, numbers_vertical_align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title="', legend_font_name='sans-serif', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=None, end_location=2, legend_bbox_to_anchor=(1.05, legend_border_axes_pad=None, legend horizontal spacing=None, $legend \ n \ columns=1$, legend vertical spacing=None, *legend border=True*, legend_border_padding=None, legend shadow=False, end rounded corners=False, render axes=False, axes font name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(7,7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

Parameters

- **figure** id (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_image (*bool*, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (*bool*, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- **final_marker_face_colour** (See Below, optional) The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) – The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- **line_style** (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **numbers font size** (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.

• numbers_font_weight (See Below, optional) - The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- **legend_font_name** (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- legend_font_size (int, optional) The font size of the legend.
- legend_font_weight (See Below, optional)—The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- **legend_bbox_to_anchor** ((*float*, *float*) *tuple*, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.

- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view iterations (figure id=None, new figure=False, iters=None, render image=True, subplots enabled=False, channels=None. interpolation='bilinear', cmap name=None, render lines=True, alpha=1.0, masked=True. line_style='-', line width=2, line colour=None, render_markers=True, marker edge colour=None, marker face colour=None, marker style='o', $marker\ size=4$, marker edge width=1.0, render numbering=False, numbers horizontal align='center'. numbers vertical align='bottom'. numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=None,axes y limits=None, $axes \ x \ ticks=None,$ axes y ticks=None, figure size=(7,7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (str (See Below), optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_colour** (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_width (*float* or *list* of *float*, optional) – The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different

value per iteration shape.

- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (str (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend title (str, optional) The title of the legend.
- **legend_font_name** (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property reconstructed_initial_shape

Returns the initial shape's reconstruction with the shape model that was used to initialise the iterative optimisation process.

Type menpo.shape.PointCloud

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *reconstructed_initial_shape* and *final_shape*.

Type list of (n_params,) ndarray

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists), *reconstructed initial shape* and *final shape*.

Type list of menpo.shape.PointCloud

Multi-Scale Iterative Result

Classes for defining a multi-scale iterative fitting result.

MultiScaleNonParametricIterativeResult

Bases: NonParametricIterativeResult

Class for defining a multi-scale non-parametric iterative fitting result, i.e. the result of a multi-scale method that does not optimise over a parametric shape model. It holds the shapes of all the iterations of the fitting procedure, as well as the scales. It can optionally store the image on which the fitting was applied, as well as its ground truth shape.

Parameters

- results (*list* of *NonParametricIterativeResult*) The *list* of non parametric iterative results per scale.
- **scales** (*list* of *float*) The scale values (normally small to high).
- **affine_transforms** (*list* of *menpo.transform.Affine*) The list of affine transforms per scale that transform the shapes into the original image space.
- scale_transforms (list of menpo.shape.Scale) The list of scaling transforms per scale.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- gt_shape (menpo.shape.PointCloud or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

```
displacements_stats (stat_type='mean')
```

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

Parameters stat_type ({ 'mean', 'median', 'min', 'max'}, optional) – Specifies a statistic metric to be extracted from the displacements.

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError - type must be 'mean', 'median', 'min' or 'max'

```
errors (compute error=None)
```

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

```
final_error (compute_error=None)
```

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

```
initial error(compute error=None)
```

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

```
plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-
', line_width=2, render_markers=True, marker_style='o', marker_size=4,
marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0,
render_axes=True, axes_font_name='sans-serif', axes_font_size=10,
axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0,
axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6),
render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of the cost function evolution at each fitting iteration.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- **new_figure** (*bool*, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• line_style ({ '-', '--', '-.', ':'}, optional) – The style of the lines.

- line width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (marker, optional) The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.

- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (*str* (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line_width (*float*, optional) The width of the lines.
- render markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- **figure_size** ((*float*, *float*) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

plot_errors (compute_error=None, figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', $line_width=2$, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', $marker_edge_width=1.0$, render_axes=True, axes_font_name='sans-serif', axes font style='normal', axes font size=10, axes font weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5) Plot of the error evolution at each fitting iteration.

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- **new_figure** (*bool*, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line width (*float*, optional) The width of the lines.
- render_markers (*bool*, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (*colour* or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (*float* or (*float*, *float*) or None, optional) The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

to_result(pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view (figure id=None, new figure=False, render image=True, render final shape=True, render_initial_shape=False, subplots enabled=True, chan $render_gt_shape=False,$ nels=None, interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, fifinal_marker_edge_colour='k', nal_marker_face_colour='r', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt marker face colour='y', gt marker edge colour='k', gt line colour='y', render lines=True, line style='-', line width=2, render markers=True, marker style='o', marker size=4, render numbering=False, numbers horizontal align='center', $marker\ edge\ width=1.0,$ numbers vertical align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title="', legend_font_name='sans-serif', legend_font_style='normal', legend_font_weight='normal', *legend_font_size=10*, legend_marker_scale=None, end location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, *legend_border=True*, legend_border_padding=None, *legend_shadow=False*, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape, ground truth shape and the image, id they were provided.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (*bool*, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (*bool*, optional) If True and the ground truth shape exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- **cmap_name** (*str*, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.

• **final_marker_face_colour** (See Below, optional) - The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) - The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional)—The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- **line_style** (*str* or *list* of *str*, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (*bool*, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- legend_font_size (int, optional) The font size of the legend.
- legend_font_weight (See Below, optional)—The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- **legend_border** (*bool*, optional) If True, a frame will be drawn around the legend.

- legend_border_padding (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- **figure_size** ((*float*, *float*) *tuple* or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

view iterations (figure id=None, new figure=False, iters=None, render image=True, subplots_enabled=False, channels=None. interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, render lines=True, line_style='-', $line_width=2$, line colour=None, render_markers=True, marker edge colour=None, marker face colour=None, marker style='o', $marker_size=4$. $marker_edge_width=1.0,$ render numbering=False, numbers horizontal align='center', numbers vertical align='bottom'. numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numrender_legend=True, bers_font_colour='k', legend_title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend_bbox_to_anchor=(1.05, legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes font size=10, axes font style='normal', axes font weight='normal', axes x limits=None, axes y limits=None, $axes \ x \ ticks=None,$ axes y ticks=None, figure size=(7,7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.shapes[1]	Iteration 1
i	self.shapes[i]	Iteration i
n_iters	self.final_shape	Final shape

- render_image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- **interpolation** (*str* (See Below), optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (*bool*, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_colour (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_width (*float* or *list* of *float*, optional) – The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different

value per iteration shape.

- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (*str* (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (*str* (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend title (str, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- **legend_bbox_to_anchor** ((*float*, *float*) *tuple*, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- **render_axes** (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property n_iters_per_scale

Returns the number of iterations per scale of the fitting process.

Type list of int

property n_scales

Returns the number of scales used during the fitting process.

Type int

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists) and *final_shape*.

Type *list* of *menpo.shape.PointCloud*

MultiScaleParametricIterativeResult

Bases: MultiScaleNonParametricIterativeResult

Class for defining a multi-scale parametric iterative fitting result, i.e. the result of a multi-scale method that optimizes over a parametric shape model. It holds the shapes of all the iterations of the fitting procedure, as well as the scales. It can optionally store the image on which the fitting was applied, as well as its ground truth shape.

Note: When using a method with a parametric shape model, the first step is to **reconstruct the initial shape** using the shape model. The generated reconstructed shape is then used as initialisation for the iterative optimisation. This step is not counted in the number of iterations.

Parameters

- results (*list* of *ParametricIterativeResult*) The *list* of parametric iterative results per scale.
- scales (*list* of *float*) The scale values (normally small to high).
- **affine_transforms** (*list* of *menpo.transform.Affine*) The list of affine transforms per scale that transform the shapes into the original image space.
- scale_transforms (list of menpo.shape.Scale) The list of scaling transforms per scale.
- **image** (*menpo.image.Image* or *subclass* or None, optional) The image on which the fitting process was applied. Note that a copy of the image will be assigned as an attribute. If None, then no image is assigned.
- **gt_shape** (*menpo.shape.PointCloud* or None, optional) The ground truth shape associated with the image. If None, then no ground truth shape is assigned.

displacements()

A list containing the displacement between the shape of each iteration and the shape of the previous one.

Type *list* of *ndarray*

```
displacements_stats (stat_type='mean')
```

A list containing a statistical metric on the displacements between the shape of each iteration and the shape of the previous one.

```
Parameters stat_type ({'mean', 'median', 'min', 'max'}, optional) - Specifies a statistic metric to be extracted from the displacements.
```

Returns displacements_stat (*list* of *float*) – The statistical metric on the points displacements for each iteration.

Raises ValueError – type must be 'mean', 'median', 'min' or 'max'

```
errors (compute_error=None)
```

Returns a list containing the error at each fitting iteration, if the ground truth shape exists.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the shape at each iteration and the ground truth shape.

Returns errors (*list* of *float*) – The error at each iteration of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

final_error (compute_error=None)

Returns the final error of the fitting process, if the ground truth shape exists. This is the error computed based on the *final shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the fitted and ground truth shapes.

Returns final_error (*float*) – The final error at the end of the fitting process.

Raises ValueError – Ground truth shape has not been set, so the final error cannot be computed

initial_error(compute_error=None)

Returns the initial error of the fitting process, if the ground truth shape and initial shape exist. This is the error computed based on the *initial_shape*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the initial and ground truth shapes.

Returns initial_error (*float*) – The initial error at the beginning of the fitting process.

Raises

- ValueError Initial shape has not been set, so the initial error cannot be computed
- **ValueError** Ground truth shape has not been set, so the initial error cannot be computed

plot_costs (figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='line width=2, render markers=True, marker style='o', marker size=4, marker_face_colour='b', marker_edge_colour='k', $marker_edge_width=1.0,$ render_axes=True, axes_font_name='sans-serif', $axes_font_size=10$, axes_font_weight='normal', axes_font_style='normal', $axes_x_limits=0.0$, axes y limits=None, axes x ticks=None, axes y ticks=None, figure size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5) Plot of the cost function evolution at each fitting iteration.

Parameters

- **figure id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None, optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line style ({'-', '--', '-.', ':'}, optional) The style of the lines.
- line_width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.

• marker_style (marker, optional) – The style of the markers. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers.If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float*, optional) The width of the markers' edge.
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- axes_font_size (*int*, optional) The font size of the axes.
- axes_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (*bool*, optional) If True, the grid will be rendered.

- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines
- grid_line_width (float, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_displacements (stat_type='mean', figure_id=None, new_figure=False, render_lines=True, line_colour='b', line_style='-', line_width=2, render_markers=True, marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k', marker_edge_width=1.0, render_axes=True, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
```

Plot of a statistical metric of the displacement between the shape of each iteration and the shape of the previous one.

Parameters

- **stat_type** ({mean, median, min, max}, optional) Specifies a statistic metric to be extracted from the displacements (see also *displacements_stats()* method).
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render_lines (bool, optional) If True, the line will be rendered.
- line_colour (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -., :}
```

- line width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (colour or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker edge width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes font name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes font size (int, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• **axes_font_weight** (*str* (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

```
plot_errors (compute_error=None, figure_id=None,
                                                            new_figure=False,
                                                                                render_lines=True,
                line colour='b',
                                      line_style='-',
                                                         line\_width=2,
                                                                             render_markers=True,
                marker_style='o', marker_size=4, marker_face_colour='b', marker_edge_colour='k',
                marker\_edge\_width=1.0,
                                               render axes=True,
                                                                       axes_font_name='sans-serif',
                                                                        axes_font_weight='normal',
                                         axes_font_style='normal',
                axes\_font\_size=10,
                axes_x_limits=0.0, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None,
                figure_size=(10, 6), render_grid=True, grid_line_style='--', grid_line_width=0.5)
     Plot of the error evolution at each fitting iteration.
```

Parameters

- **compute_error** (*callable*, optional) Callable that computes the error between the shape at each iteration and the ground truth shape.
- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render lines (*bool*, optional) If True, the line will be rendered.
- **line_colour** (*colour* or None (See below), optional) The colour of the line. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• line_style (str (See below), optional) – The style of the lines. Example options:

```
{-, --, -:}
```

- line_width (*float*, optional) The width of the lines.
- render_markers (bool, optional) If True, the markers will be rendered.
- marker_style (str (See below), optional) The style of the markers. Example marker options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int, optional) The size of the markers in points.
- marker_face_colour (colour or None, optional) The face (filling) colour of the markers. If None, the colour is sampled from the jet colormap. Example colour options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_colour (*colour* or None, optional) – The edge colour of the markers. If None, the colour is sampled from the jet colormap. Example *colour* options are

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- marker_edge_width (float, optional) The width of the markers' edge.
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- **axes_font_style** (*str* (See below), optional) The font style of the axes. Example options

```
{normal, italic, oblique}
```

• axes_font_weight (str (See below), optional) – The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the graph as a percentage of the curves' width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits (float or (float, float) or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes x ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Returns renderer (menpo.visualize.GraphPlotter) – The renderer object.

reconstructed_initial_error (compute_error=None)

Returns the error of the reconstructed initial shape of the fitting process, if the ground truth shape exists. This is the error computed based on the *reconstructed initial shapes[0]*.

Parameters compute_error (*callable*, optional) – Callable that computes the error between the reconstructed initial and ground truth shapes.

Returns reconstructed_initial_error (*float*) – The error that corresponds to the initial shape's reconstruction.

Raises ValueError – Ground truth shape has not been set, so the reconstructed initial error cannot be computed

to_result (pass_image=True, pass_initial_shape=True, pass_gt_shape=True)

Returns a *Result* instance of the object, i.e. a fitting result object that does not store the iterations. This can be useful for reducing the size of saved fitting results.

Parameters

- pass_image (bool, optional) If True, then the image will get passed (if it exists).
- pass_initial_shape (bool, optional) If True, then the initial shape will get passed (if it exists).
- pass_gt_shape (bool, optional) If True, then the ground truth shape will get passed (if it exists).

Returns result (Result) – The final "lightweight" fitting result.

view (figure id=None, new figure=False, render image=True, render final shape=True, subplots enabled=True, render_initial_shape=False, $render_gt_shape=False,$ channels=None, interpolation='bilinear', cmap name=None, alpha=1.0, masked=True, fifinal_marker_edge_colour='k', nal_marker_face_colour='r', final_line_colour='r', tial marker face colour='b', initial marker edge colour='k', initial line colour='b', gt marker face colour='y', gt marker edge colour='k', gt line colour='y', render lines=True, line style='-', line width=2, render markers=True, marker style='o', marker size=4, numbers horizontal align='center', $marker\ edge\ width=1.0,$ render numbering=False, numbers vertical align='bottom', numbers font name='sans-serif', numbers font size=10, numbers_font_style='normal', numbers_font_weight='normal', numbers_font_colour='k', render_legend=True, legend_title="', legend_font_name='sans-serif', legend_font_style='normal', legend font size=10, legend_font_weight='normal', legend_marker_scale=None, end location=2, legend bbox to anchor=(1.05,legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, *legend_shadow=False*, end_rounded_corners=False, render_axes=False, axes_font_name='sans-serif', axes_font_size=10, axes_font_style='normal', axes_font_weight='normal', axes_x_limits=None, axes_y_limits=None, axes_x_ticks=None, axes_y_ticks=None, figure_size=(7, 7)) Visualize the fitting result. The method renders the final fitted shape and optionally the initial shape,

ground truth shape and the image, id they were provided.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- render image (bool, optional) If True and the image exists, then it gets rendered.
- render_final_shape (bool, optional) If True, then the final fitting shape gets rendered.
- render_initial_shape (bool, optional) If True and the initial fitting shape exists, then it gets rendered.
- render_gt_shape (bool, optional) If True and the ground truth shape exists, then it gets rendered.
- subplots_enabled (bool, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- channels (int or list of int or all or None) If int or list of int, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (See Below, optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

- cmap_name (str, optional,) If None, single channel and three channel images default to greyscale and rgb colormaps respectively.
- alpha (float, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.

• **final_marker_face_colour** (See Below, optional) - The face (filling) colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_marker_edge_colour** (See Below, optional) - The edge colour of the markers of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• **final_line_colour** (See Below, optional) - The line colour of the final fitting shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_marker_edge_colour (See Below, optional) - The edge colour of the markers of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• initial_line_colour(See Below, optional) - The line colour of the initial shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_face_colour (See Below, optional) - The face (filling) colour of the markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_marker_edge_colour (See Below, optional) - The edge colour of the
markers of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• gt_line_colour (See Below, optional) - The line colour of the ground truth shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_lines** (*bool* or *list* of *bool*, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- line_style (str or list of str, optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order. Example options:

```
{'-', '--', '-.', ':'}
```

- line_width (float or list of float, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- marker_style (*str* or *list* of *str*, optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order. Example options:

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (final, initial, groundtruth) order.
- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different value per shape in (*final*, *initial*, *groundtruth*) order.
- render numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align ({center, right, left}, optional) The horizontal alignment of the numbers' texts.
- numbers_vertical_align ({center, top, bottom, baseline}, optional) The vertical alignment of the numbers' texts.
- numbers_font_name (See Below, optional) The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (See Below, optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (*bool*, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- legend_font_style ({normal, italic, oblique}, optional) The font style of the legend.
- **legend_font_size** (*int*, optional) The font size of the legend.
- legend_font_weight (See Below, optional)—The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- legend_border_axes_pad (*float*, optional) The pad between the axes and legend border
- legend_n_columns (int, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.

- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (See Below, optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- axes_font_weight (See Below, optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- axes_x_limits (float or (float, float) or None, optional) The limits of the x axis. If float, then it sets padding on the right and left of the Image as a percentage of the Image's width. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (class) – The renderer object.

view iterations (figure id=None, new figure=False, iters=None, render image=True, subplots enabled=False, channels=None. interpolation='bilinear', render lines=True, cmap name=None, alpha=1.0, masked=True. line_style='-', line width=2, line colour=None, render_markers=True, marker edge colour=None, marker face colour=None, marker style='o', $marker\ size=4$, marker_edge_width=1.0, render numbering=False, numbers horizontal align='center'. numbers vertical align='bottom'. numbers font name='sans-serif', numbers font size=10, bers font style='normal', numbers font weight='normal', numbers_font_colour='k', render_legend=True, legend_title=", legend_font_name='sans-serif', legend_font_style='normal', end_font_size=10, legend_font_weight='normal', legend_marker_scale=None, legend location=2, legend_bbox_to_anchor=(1.05, legend_border_axes_pad=None, $legend_n_columns=1$, legend_horizontal_spacing=None, legend_vertical_spacing=None, legend_border=True, legend_border_padding=None, legend_shadow=False, legend_rounded_corners=False, render_axes=False, axes_font_name='sansserif', axes font size=10, axes font style='normal', axes font weight='normal', $axes \ x \ limits=None,$ axes y limits=None, $axes \ x \ ticks=None,$ axes y ticks=None, figure size=(7,7))

Visualize the iterations of the fitting process.

Parameters

- **figure_id** (*object*, optional) The id of the figure to be used.
- new_figure (bool, optional) If True, a new figure is created.
- iters (int or list of int or None, optional) The iterations to be visualized. If None, then all the iterations are rendered.

No.	Visualised shape	Description
0	self.initial_shape	Initial shape
1	self.reconstructed_initial_shape	Reconstructed initial
2	self.shapes[2]	Iteration 1
i	self.shapes[i]	Iteration i-1
n_iters+1	self.final_shape	Final shape

- render image (bool, optional) If True and the image exists, then it gets rendered.
- **subplots_enabled** (*bool*, optional) If True, then the requested final, initial and ground truth shapes get rendered on separate subplots.
- **channels** (*int* or *list* of *int* or all or None) If *int* or *list* of *int*, the specified channel(s) will be rendered. If all, all the channels will be rendered in subplots. If None and the image is RGB, it will be rendered in RGB mode. If None and the image is not RGB, it is equivalent to all.
- interpolation (str (See Below), optional) The interpolation used to render the image. For example, if bilinear, the image will be smooth and if nearest, the image will be pixelated. Example options

```
{none, nearest, bilinear, bicubic, spline16, spline36, hanning,
hamming, hermite, kaiser, quadric, catrom, gaussian, bessel,
mitchell, sinc, lanczos}
```

• **cmap_name** (*str*, optional,) – If None, single channel and three channel images default to greyscale and rgb colormaps respectively.

- alpha (*float*, optional) The alpha blending value, between 0 (transparent) and 1 (opaque).
- masked (bool, optional) If True, then the image is rendered as masked.
- render_lines (bool or list of bool, optional) If True, the lines will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- **line_style** (*str* or *list* of *str* (See below), optional) The style of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options:

```
{-, --, -:}
```

- line_width (*float* or *list* of *float*, optional) The width of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- line_colour (*colour* or *list* of *colour* (See Below), optional) The colour of the lines. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_style (str or `list of str (See below), optional) The style of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{., ,, o, v, ^, <, >, +, x, D, d, s, p, *, h, H, 1, 2, 3, 4, 8}
```

- marker_size (int or list of int, optional) The size of the markers in points. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape.
- marker_edge_colour (colour or list of colour (See Below), optional) The edge colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_face_colour (colour or list of colour (See Below), optional) – The face (filling) colour of the markers. You can either provide a single value that will be used for all shapes or a list with a different value per iteration shape. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

• marker_edge_width (*float* or *list* of *float*, optional) – The width of the markers' edge. You can either provide a single value that will be used for all shapes or a list with a different

value per iteration shape.

- render_numbering (bool, optional) If True, the landmarks will be numbered.
- numbers_horizontal_align (*str* (See below), optional) The horizontal alignment of the numbers' texts. Example options

```
{center, right, left}
```

• numbers_vertical_align (str (See below), optional) – The vertical alignment of the numbers' texts. Example options

```
{center, top, bottom, baseline}
```

• numbers_font_name (str (See below), optional) – The font of the numbers. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- numbers_font_size (*int*, optional) The font size of the numbers.
- numbers_font_style ({normal, italic, oblique}, optional) The font style of the numbers.
- numbers_font_weight (str (See below), optional) The font weight of the numbers. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

• numbers_font_colour(See Below, optional) - The font colour of the numbers. Example options

```
{r, g, b, c, m, k, w}
or
(3, ) ndarray
```

- render_legend (bool, optional) If True, the legend will be rendered.
- legend title (str, optional) The title of the legend.
- **legend_font_name** (See below, optional) The font of the legend. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

• **legend_font_style** (*str* (See below), optional) – The font style of the legend. Example options

```
{normal, italic, oblique}
```

- **legend_font_size** (*int*, optional) The font size of the legend.
- **legend_font_weight** (*str* (See below), optional) The font weight of the legend. Example options

```
{ultralight, light, normal, regular, book, medium, roman,
semibold, demibold, demi, bold, heavy, extra bold, black}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- legend_location (int, optional) The location of the legend. The predefined values
 are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float) tuple, optional) The bbox that the legend will be anchored.
- **legend_border_axes_pad** (*float*, optional) The pad between the axes and legend border.
- **legend_n_columns** (*int*, optional) The number of the legend's columns.
- legend_horizontal_spacing (*float*, optional) The spacing between the columns.
- legend_vertical_spacing (float, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- **legend_border_padding** (*float*, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (*bool*, optional) If True, the axes will be rendered.
- axes_font_name (str (See below), optional) The font of the axes. Example options

```
{serif, sans-serif, cursive, fantasy, monospace}
```

- axes_font_size (int, optional) The font size of the axes.
- axes_font_style ({normal, italic, oblique}, optional) The font style of the axes.
- **axes_font_weight** (*str* (See below), optional) The font weight of the axes. Example options

```
{ultralight, light, normal, regular, book, medium, roman, semibold, demibold, demi, bold, heavy, extra bold, black}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the Image as a percentage of the Image's

width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set automatically.

- axes_y_limits ((float, float) tuple or None, optional) The limits of the y axis. If float, then it sets padding on the top and bottom of the Image as a percentage of the Image's height. If tuple or list, then it defines the axis limits. If None, then the limits are set automatically.
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) tuple or None optional) The size of the figure in inches.

Returns renderer (*class*) – The renderer object.

property costs

Returns a *list* with the cost per iteration. It returns None if the costs are not computed.

Type list of float or None

property final_shape

Returns the final shape of the fitting process.

Type menpo.shape.PointCloud

property gt_shape

Returns the ground truth shape associated with the image. In case there is not an attached ground truth shape, then None is returned.

Type menpo.shape.PointCloud or None

property image

Returns the image that the fitting was applied on, if it was provided. Otherwise, it returns None.

Type menpo.shape.Image or subclass or None

property initial_shape

Returns the initial shape that was provided to the fitting method to initialise the fitting process. In case the initial shape does not exist, then None is returned.

Type menpo.shape.PointCloud or None

property is_iterative

Flag whether the object is an iterative fitting result.

Type bool

property n_iters

Returns the total number of iterations of the fitting process.

Type int

property n_iters_per_scale

Returns the number of iterations per scale of the fitting process.

Type list of int

property n_scales

Returns the number of scales used during the fitting process.

Type int

property reconstructed_initial_shapes

Returns the result of the reconstruction step that takes place at each scale before applying the iterative optimisation.

Type *list* of *menpo.shape.PointCloud*

property shape_parameters

Returns the *list* of shape parameters obtained at each iteration of the fitting process. The *list* includes the parameters of the *initial_shape* (if it exists) and *final_shape*.

```
Type list of (n_params,) ndarray
```

property shapes

Returns the *list* of shapes obtained at each iteration of the fitting process. The *list* includes the *initial_shape* (if it exists) and *final_shape*.

Type list of menpo.shape.PointCloud

2.2.10 menpofit.transform

Model Driven Transforms

OrthoMDTransform

```
class menpofit.transform.OrthoMDTransform(model, transform_cls, source=None)
    Bases: GlobalMDTransform
```

A transform that couples an alignment transform to a statistical model together with a global similarity transform, such that the weights of the transform are fully specified by both the weights of statistical model and the weights of the similarity transform. The model is assumed to generate an instance which is then transformed by the similarity transform; the result defines the target landmarks of the transform. If no source is provided, the mean of the model is defined as the source landmarks of the transform.

This transform (in contrast to the GlobalMDTransform) additionally orthonormalises both the global and the model basis against each other, ensuring that orthogonality and normalization is enforced across the unified bases.

Parameters

- model (OrthoPDM or subclass) A linear statistical shape model (Point Distribution Model) that also has a global similarity transform that is orthonormalised with the shape bases.
- transform_cls (subclass of menpo.transform.Alignment) A class of menpo.transform.Alignment. The align constructor will be called on this with the source and target landmarks. The target is set to the points generated from the model using the provide weights the source is either given or set to the model's mean.
- **source** (*menpo.shape.PointCloud* or None, optional) The source landmarks of the transform. If None, the mean of the model is used.

Jp()

Compute the parameters' Jacobian, as shown in [1].

Returns Jp ((n_params, n_params) ndarray) - The parameters' Jacobian.

References

```
apply (x, batch_size=None, **kwargs) Applies this transform to x.
```

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply () method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (int, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- \mathbf{kwargs} (dict) Passed through to $_apply$ ().

Returns transformed (type (x)) – The transformed object or array

```
apply_inplace(*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

```
as vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

compose_after (transform)

Returns a TransformChain that represents this transform composed after the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

Parameters transform (Transform) – Transform to be applied before self

Returns transform (TransformChain) – The resulting transform chain.

compose_after_from_vector_inplace (delta)

Composes two transforms together based on the first order approximation proposed in [1].

Parameters delta ((N,) *ndarray*) – Vectorized *ModelDrivenTransform* to be applied before self.

Returns transform (*self*) – self, updated to the result of the composition

References

compose_before (transform)

Returns a **TransformChain** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

Parameters transform (Transform) - Transform to be applied after self

Returns transform (TransformChain) – The resulting transform chain.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) – A copy of this object

d_dp (points)

The derivative of this *ModelDrivenTransform* with respect to the parametrisation changes evaluated at points.

This is done by chaining the derivative of points wrt the source landmarks on the transform (dW/dL) together with the Jacobian of the linear model wrt its weights (dX/dp).

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns d_dp ((n_points, n_parameters, n_dims) *ndarray*) – The Jacobian with respect to the parametrisation.

from_vector(vector)

Build a new instance of the object from it's vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns object (type (self)) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (*bool*) – If the vectorized object contains nan values.

pseudoinverse()

The pseudoinverse of the transform - that is, the transform that results from swapping *source* and *target*, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

Type type (self)

pseudoinverse_vector(vector)

The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for self.from_vector(vector).pseudoinverse.as_vector(). On ModelDrivenTransform this is especially fast - we just negate the vector provided.

Parameters vector ((P,) *ndarray*) – A vectorized version of self

Returns pseudoinverse_vector ((N,) ndarray) - The pseudoinverse of the vector provided

```
set_target (new_target)
```

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and regenerate.

property has_true_inverse

Whether the transform has true inverse.

Type bool

property n_dims

The number of dimensions that the transform supports.

Type int

property n_dims_output

The output of the data from the transform.

None if the output of the transform is not dimension specific.

Type int or None

property n_parameters

The total number of parameters.

Type int

property n_points

The number of points on the target.

Type int

property target

The current *menpo.shape.PointCloud* that this object produces.

Type menpo.shape.PointCloud

LinearOrthoMDTransform

class menpofit.transform.LinearOrthoMDTransform(model, sparse_instance)

Bases: OrthoPDM, Transform

A transform that couples an alignment transform to a statistical model together with a global similarity transform, such that the weights of the transform are fully specified by both the weights of statistical model and the weights of the similarity transform. The model is assumed to generate an instance which is then transformed by the similarity transform; the result defines the target landmarks of the transform. If no source is provided, the mean of the model is defined as the source landmarks of the transform.

This transform (in contrast to the GlobalMDTransform) additionally orthonormalises both the global and the model basis against each other, ensuring that orthogonality and normalization is enforced across the unified bases.

This transform (in contrast to the OrthoMDTransform) should be used with linear statistical models of dense shapes.

Parameters

- model (menpo.model.LinearModel) A linear statistical shape model.
- sparse instance (menpo.shape.PointCloud) The source landmarks of the transform.

```
apply (x, batch_size=None, **kwargs)
```

Applies this transform to x.

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform apply () method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (int, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (*dict*) Passed through to _apply().

Returns transformed (type (x)) – The transformed object or array

```
apply_inplace(*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

compose_after (transform)

Returns a TransformChain that represents this transform composed after the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

Parameters transform (Transform) - Transform to be applied before self

Returns transform (TransformChain) – The resulting transform chain.

compose_before (transform)

Returns a **TransformChain** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

Parameters transform (Transform) – Transform to be applied after self

Returns transform (TransformChain) – The resulting transform chain.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other Copyable objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

d_dp (_)

The derivative with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns d_dp ((n_points, n_parameters, n_dims) *ndarray*) – The Jacobian with respect to the parametrisation.

from_vector (vector)

Build a new instance of the object from it's vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns object (type (self)) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has nan values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (bool) – If the vectorized object contains nan values.

increment (shapes, n_shapes=None, forgetting_factor=1.0, max_n_components=None, verbose=False)

Update the eigenvectors, eigenvalues and mean vector of this model by performing incremental PCA on the given samples.

Parameters

- **shapes** (*list* of *menpo.shape.PointCloud*) List of new shapes to update the model from.
- n_shapes (int or None, optional) If int, then shapes must be an iterator that yields n_shapes. If None, then shapes has to be a list (so we know how large the data matrix needs to be).
- **forgetting_factor** ([0.0, 1.0] *float*, optional) Forgetting factor that weights the relative contribution of new samples vs old samples. If 1.0, all samples are weighted equally and, hence, the results is the exact same as performing batch PCA on the concatenated list of old and new simples. If <1.0, more emphasis is put on the new samples. See [1] for details.
- max_n_components (*int* or None, optional) The maximum number of components that the model will keep. If None, then all the components will be kept.
- ullet verbose (bool, optional) If True, then information about the progress will be printed.

References

set_target (target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (*menpo.shape.PointCloud*) – The new target that this object should try and regenerate.

property dense_target

The current dense *menpo.shape.PointCloud* that this object produces.

Type menpo.shape.PointCloud

property global_parameters

The parameters for the global transform.

Type (n_global_parameters,) ndarray

property n_active_components

The number of components currently in use on this model.

Type int

property n_dims

The number of dimensions of the spatial instance of the model

Type int

property n_dims_output

The output of the data from the transform.

None if the output of the transform is not dimension specific.

Type int or None

property n_global_parameters

The number of parameters in the global_transform

Type int

property n_landmarks

The number of sparse landmarks.

Type int

property n_parameters

The length of the vector that this object produces.

Type int

property n_points

The number of points on the target.

Type int

property n_weights

The number of parameters in the linear model.

Type int

property sparse_target

The current sparse menpo.shape.PointCloud that this object produces.

Type menpo.shape.PointCloud

property target

The current *menpo.shape.PointCloud* that this object produces.

Type menpo.shape.PointCloud

property weights

The weights of the model.

Type (n_weights,) *ndarray*

Homogeneous Transforms

Differentiable Affine

```
class menpofit.transform.DifferentiableAffine (h\_matrix, copy=True, skip\_checks=False)
```

Bases: Affine, DP, DX

Base class for an affine transformation that can compute its own derivative with respect to spatial changes, as well as its parametrisation.

```
apply (x, batch_size=None, **kwargs)
```

Applies this transform to x.

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply () method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (*int*, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (*dict*) Passed through to _apply().

Returns transformed (type(x)) – The transformed object or array

```
apply_inplace(*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

```
compose after(transform)
```

A Transform that represents this transform composed after the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

An attempt is made to perform native composition, but will fall back to a **TransformChain** as a last resort. See <code>composes_with</code> for a description of how the mode of composition is decided.

Parameters transform (Transform) - Transform to be applied before self

Returns transform (**Transform** or **TransformChain**) – If the composition was native, a single new **Transform** will be returned. If not, a **TransformChain** is returned instead.

```
compose_after_from_vector_inplace(vector)
```

Specialised inplace composition with a vector. This should be overridden to provide specific cases of composition whereby the current state of the transform can be derived purely from the provided vector.

Parameters vector ((n_parameters,) *ndarray*) — Vector to update the transform state with.

compose_after_inplace(transform)

Update self so that it represents **this** transform composed **after** the given transform:

```
a_orig = a.copy()
a.compose_after_inplace(b)
a.apply(p) == a_orig.apply(b.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

Parameters transform (composes_inplace_with) - Transform to be applied before self

Raises ValueError - If transform isn't an instance of composes_inplace_with

compose_before (transform)

A **Transform** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

An attempt is made to perform native composition, but will fall back to a **TransformChain** as a last resort. See <code>composes_with</code> for a description of how the mode of composition is decided.

Parameters transform (Transform) - Transform to be applied after self

Returns transform (**Transform** or **TransformChain**) – If the composition was native, a single new **Transform** will be returned. If not, a **TransformChain** is returned instead.

compose_before_inplace(transform)

Update self so that it represents **this** transform composed **before** the given transform:

```
a_orig = a.copy()
a.compose_before_inplace(b)
a.apply(p) == b.apply(a_orig.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

Parameters transform (composes_inplace_with) - Transform to be applied after self

Raises ValueError - If transform isn't an instance of composes_inplace_with

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

d_dp (points)

The derivative with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

 $\mathbf{d}_{-}\mathbf{dp}$ ((n_points, n_parameters, n_dims) ndarray) — The Jacobian with respect to the parametrisation.

d_dp[i, j, k] is the scalar differential change that the k'th dimension of the i'th point experiences due to a first order change in the j'th scalar in the parametrisation vector.

d_dx (points)

The first order derivative with respect to spatial changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

 $\mathbf{d}_{\mathbf{d}}\mathbf{x}$ ((n_points, n_dims, n_dims) ndarray) – The Jacobian wrt spatial changes.

 $d_dx[i, j, k]$ is the scalar differential change that the j'th dimension of the i'th point experiences due to a first order change in the k'th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

decompose()

Decompose this transform into discrete Affine Transforms.

Useful for understanding the effect of a complex composite transform.

Returns

transforms (*list* of **DiscreteAffine**) – Equivalent to this affine transform, such that

```
reduce(lambda x, y: x.chain(y), self.decompose()) == self
```

from_vector (vector)

Build a new instance of the object from its vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) *ndarray*) – Flattened representation of the object.

Returns transform (Homogeneous) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (bool) – If the vectorized object contains nan values.

classmethod init_from_2d_shear(phi, psi, degrees=True)

Convenience constructor for 2D shear transformations about the origin.

Parameters

- phi (*float*) The angle of shearing in the X direction.
- **psi** (*float*) The angle of shearing in the Y direction.
- **degrees** (*bool*, optional) If True phi and psi are interpreted as degrees. If False, phi and psi are interpreted as radians.

Returns shear_transform (Affine) – A 2D shear transform.

classmethod init_identity(n_dims)

Creates an identity matrix Affine transform.

Parameters n_dims (*int*) – The number of dimensions.

Returns identity (Affine) – The identity matrix transform.

pseudoinverse()

The pseudoinverse of the transform - that is, the transform that results from swapping *source* and *target*, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

Type Homogeneous

pseudoinverse_vector(vector)

The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for:

```
self.from_vector(vector).pseudoinverse().as_vector()
```

Can be much faster than the explict call as object creation can be entirely avoided in some cases.

Parameters vector ((n_parameters,) ndarray) - A vectorized version of self

Returns pseudoinverse_vector ($(n_parameters,)$ *ndarray*) – The pseudoinverse of the vector provided

```
set_h_matrix(value, copy=True, skip_checks=False)
```

Deprecated Deprecated - do not use this method - you are better off just creating a new transform!

Updates h_matrix, optionally performing sanity checks.

Note that it won't always be possible to manually specify the h_matrix through this method, specifically if changing the h_matrix could change the nature of the transform. See h_matrix_is_mutable for how you can discover if the h_matrix is allowed to be set for a given class.

Parameters

• **value** (*ndarray*) – The new homogeneous matrix to set.

- copy (bool, optional) If False, do not copy the h_matrix. Useful for performance.
- **skip_checks** (*bool*, optional) If True, skip checking. Useful for performance.

Raises NotImplementedError - If h_matrix_is_mutable returns False.

property composes_inplace_with

Affine can swallow composition with any other Affine.

property composes_with

Any Homogeneous can compose with any other Homogeneous.

property h_matrix

The homogeneous matrix defining this transform.

property h_matrix_is_mutable

Deprecated True iff set_h_matrix() is permitted on this type of transform.

If this returns False calls to set_h_matrix() will raise a NotImplementedError.

Type bool

property has_true_inverse

The pseudoinverse is an exact inverse.

Type True

property linear_component

The linear component of this affine transform.

property n_dims

The dimensionality of the data the transform operates on.

Type int

property n_dims_output

The output of the data from the transform.

Type int

property n_parameters

n_dims * (n_dims + 1) parameters - every element of the matrix but the homogeneous part.

Type int

Examples

2D Affine: 6 parameters:

```
[p1, p3, p5]
[p2, p4, p6]
```

3D Affine: 12 parameters:

```
[p1, p4, p7, p10]
[p2, p5, p8, p11]
[p3, p6, p9, p12]
```

property translation_component

The translation component of this affine transform.

```
Type (n_dims,) ndarray
```

DifferentiableSimilarity

```
class menpofit.transform.DifferentiableSimilarity(h\_matrix, copy=True, skip\_checks=False)
```

Bases: Similarity, DP, DX

Base class for a similarity transformation that can compute its own derivative with respect to spatial changes, as well as its parametrisation.

```
apply (x, batch_size=None, **kwargs)
```

Applies this transform to x.

If x is Transformable, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform apply () method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (int, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (*dict*) Passed through to _apply().

Returns transformed (type(x)) – The transformed object or array

```
apply_inplace(*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

compose_after(transform)

A **Transform** that represents **this** transform composed **after** the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

An attempt is made to perform native composition, but will fall back to a **TransformChain** as a last resort. See <code>composes_with</code> for a description of how the mode of composition is decided.

Parameters transform (Transform) - Transform to be applied before self

Returns transform (Transform or TransformChain) – If the composition was native, a single new Transform will be returned. If not, a TransformChain is returned instead.

compose_after_from_vector_inplace (vector)

Specialised inplace composition with a vector. This should be overridden to provide specific cases of composition whereby the current state of the transform can be derived purely from the provided vector.

Parameters vector ((n_parameters,) ndarray) - Vector to update the transform state with

compose_after_inplace(transform)

Update self so that it represents **this** transform composed **after** the given transform:

```
a_orig = a.copy()
a.compose_after_inplace(b)
a.apply(p) == a_orig.apply(b.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

Parameters transform (composes_inplace_with) - Transform to be applied before
 self

Raises ValueError - If transform isn't an instance of composes_inplace_with

compose_before (transform)

A **Transform** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

An attempt is made to perform native composition, but will fall back to a **TransformChain** as a last resort. See *composes* with for a description of how the mode of composition is decided.

Parameters transform (Transform) - Transform to be applied after self

Returns transform (**Transform** or **TransformChain**) – If the composition was native, a single new **Transform** will be returned. If not, a **TransformChain** is returned instead.

compose_before_inplace(transform)

Update self so that it represents **this** transform composed **before** the given transform:

```
a_orig = a.copy()
a.compose_before_inplace(b)
a.apply(p) == b.apply(a_orig.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

Parameters transform (composes_inplace_with) - Transform to be applied after self

Raises ValueError - If transform isn't an instance of composes_inplace_with

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

d dp (points)

The derivative with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

 $\mathbf{d}_{-}\mathbf{dp}$ ((n_points, n_parameters, n_dims) ndarray) – The Jacobian with respect to the parametrisation.

d_dp[i, j, k] is the scalar differential change that the k'th dimension of the i'th point experiences due to a first order change in the j'th scalar in the parametrisation vector.

d_dx (points)

The first order derivative with respect to spatial changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

d_dx ((n_points, n_dims, n_dims) *ndarray*) – The Jacobian wrt spatial changes.

d_dx[i, j, k] is the scalar differential change that the j'th dimension of the i'th point experiences due to a first order change in the k'th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

decompose()

Decompose this transform into discrete Affine Transforms.

Useful for understanding the effect of a complex composite transform.

Returns

transforms (list of DiscreteAffine) – Equivalent to this affine transform, such that

```
reduce(lambda x, y: x.chain(y), self.decompose()) == self
```

from_vector(vector)

Build a new instance of the object from its vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns transform (Homogeneous) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (bool) – If the vectorized object contains nan values.

classmethod init_from_2d_shear(phi, psi, degrees=True)

Convenience constructor for 2D shear transformations about the origin.

Parameters

- phi (*float*) The angle of shearing in the X direction.
- **psi** (*float*) The angle of shearing in the Y direction.
- **degrees** (*bool*, optional) If True phi and psi are interpreted as degrees. If False, phi and psi are interpreted as radians.

Returns shear_transform (Affine) – A 2D shear transform.

classmethod init_identity(n_dims)

Creates an identity transform.

Parameters n dims (int) – The number of dimensions.

Returns identity (Similarity) – The identity matrix transform.

pseudoinverse()

The pseudoinverse of the transform - that is, the transform that results from swapping *source* and *target*, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

Type Homogeneous

pseudoinverse_vector(vector)

The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for:

```
self.from_vector(vector).pseudoinverse().as_vector()
```

Can be much faster than the explict call as object creation can be entirely avoided in some cases.

Parameters vector ((n_parameters,) ndarray) - A vectorized version of self

Returns pseudoinverse_vector ((n_parameters,) *ndarray*) — The pseudoinverse of the vector provided

```
set_h_matrix (value, copy=True, skip_checks=False)
```

Deprecated Deprecated - do not use this method - you are better off just creating a new transform!

Updates h_matrix, optionally performing sanity checks.

Note that it won't always be possible to manually specify the h_matrix through this method, specifically if changing the h_matrix could change the nature of the transform. See h_matrix_is_mutable for how you can discover if the h_matrix is allowed to be set for a given class.

Parameters

- **value** (*ndarray*) The new homogeneous matrix to set.
- copy (bool, optional) If False, do not copy the h_matrix. Useful for performance.
- **skip_checks** (*bool*, optional) If True, skip checking. Useful for performance.

Raises NotImplementedError - If h_matrix_is_mutable returns False.

property composes_inplace_with

Affine can swallow composition with any other Affine.

property composes_with

Any Homogeneous can compose with any other Homogeneous.

property h_matrix

The homogeneous matrix defining this transform.

property h_matrix_is_mutable

Deprecated True iff set_h_matrix() is permitted on this type of transform.

If this returns False calls to set_h_matrix() will raise a NotImplementedError.

Type bool

property has_true_inverse

The pseudoinverse is an exact inverse.

Type True

property linear_component

The linear component of this affine transform.

property n dims

The dimensionality of the data the transform operates on.

Type int

property n_dims_output

The output of the data from the transform.

Type int

property n_parameters

Number of parameters of Similarity

2D Similarity - 4 parameters

```
[(1 + a), -b, tx]
[b, (1 + a), ty]
```

3D Similarity: Currently not supported

Returns n_parameters (*int*) – The transform parameters

Raises DimensionalityError, NotImplementedError - Only 2D transforms are supported.

property translation_component

The translation component of this affine transform.

Type (n_dims,) ndarray

DifferentiableAlignmentSimilarity

```
class menpofit.transform.DifferentiableAlignmentSimilarity(source, target, rotation=True, allow\_mirror=False)
```

Bases: AlignmentSimilarity, DP, DX

Base class that constructs a similarity transformation that is the optimal transform to align the *source* to the *target*. It can compute its own derivative with respect to spatial changes, as well as its parametrisation.

aligned_source()

The result of applying self to source

Type PointCloud

alignment_error()

The Frobenius Norm of the difference between the target and the aligned source.

Type float

```
apply (x, batch_size=None, **kwargs)
```

Applies this transform to x.

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply () method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (*int*, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (*dict*) Passed through to _apply().

Returns transformed (type (x)) – The transformed object or array

```
apply_inplace (*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

as_non_alignment()

Returns the non-alignment version of the transform.

```
Type DifferentiableSimilarity
```

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

```
compose_after (transform)
```

A **Transform** that represents **this** transform composed **after** the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

An attempt is made to perform native composition, but will fall back to a **TransformChain** as a last resort. See <code>composes_with</code> for a description of how the mode of composition is decided.

Parameters transform (Transform) - Transform to be applied before self

Returns transform (**Transform** or **TransformChain**) – If the composition was native, a single new **Transform** will be returned. If not, a **TransformChain** is returned instead.

compose_after_from_vector_inplace(vector)

Specialised inplace composition with a vector. This should be overridden to provide specific cases of composition whereby the current state of the transform can be derived purely from the provided vector.

Parameters vector ((n_parameters,) *ndarray*) — Vector to update the transform state with.

compose_after_inplace(transform)

Update self so that it represents **this** transform composed **after** the given transform:

```
a_orig = a.copy()
a.compose_after_inplace(b)
a.apply(p) == a_orig.apply(b.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

Parameters transform (composes_inplace_with) - Transform to be applied before self

Raises ValueError - If transform isn't an instance of composes_inplace_with

compose_before (transform)

A **Transform** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

An attempt is made to perform native composition, but will fall back to a **TransformChain** as a last resort. See <code>composes_with</code> for a description of how the mode of composition is decided.

Parameters transform (Transform) - Transform to be applied after self

Returns transform (**Transform** or **TransformChain**) – If the composition was native, a single new **Transform** will be returned. If not, a **TransformChain** is returned instead.

compose before inplace(transform)

Update self so that it represents **this** transform composed **before** the given transform:

```
a_orig = a.copy()
a.compose_before_inplace(b)
a.apply(p) == b.apply(a_orig.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

Parameters transform (composes_inplace_with) - Transform to be applied after
 self

Raises ValueError - If transform isn't an instance of composes_inplace_with

copy()

Generate an efficient copy of this **HomogFamilyAlignment**.

Returns new_transform (type(self)) - A copy of this object

d dp (points)

The derivative with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) - The spatial points at which the derivative should be evaluated.

Returns

 $\mathbf{d}_{\mathbf{d}}\mathbf{p}$ ((n_points, n_parameters, n_dims) ndarray) – The Jacobian with respect to the parametrisation.

 $d_{p[i, j, k]}$ is the scalar differential change that the k'th dimension of the i'th point experiences due to a first order change in the j'th scalar in the parametrisation vector.

d_dx (points)

The first order derivative with respect to spatial changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

d_dx((n_points, n_dims, n_dims) *ndarray*) - The Jacobian wrt spatial changes.

d_dx[i, j, k] is the scalar differential change that the j'th dimension of the i'th point experiences due to a first order change in the k'th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

decompose()

Decompose this transform into discrete Affine Transforms.

Useful for understanding the effect of a complex composite transform.

Returns

transforms (list of DiscreteAffine) – Equivalent to this affine transform, such that

```
reduce(lambda x, y: x.chain(y), self.decompose()) == self
```

from vector(vector)

Build a new instance of the object from its vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns transform (Homogeneous) – An new instance of this class.

from vector inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has_nan_values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (bool) – If the vectorized object contains nan values.

classmethod init_from_2d_shear (phi, psi, degrees=True)

Convenience constructor for 2D shear transformations about the origin.

Parameters

• **phi** (*float*) – The angle of shearing in the X direction.

- psi (*float*) The angle of shearing in the Y direction.
- **degrees** (*bool*, optional) If True phi and psi are interpreted as degrees. If False, phi and psi are interpreted as radians.

Returns shear transform (Affine) – A 2D shear transform.

classmethod init_identity(n_dims)

Creates an identity transform.

Parameters n dims (int) – The number of dimensions.

Returns identity (Similarity) – The identity matrix transform.

pseudoinverse()

The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

Returns transform (type (self)) – The inverse of this transform.

pseudoinverse_vector(vector)

The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for:

```
self.from_vector(vector).pseudoinverse().as_vector()
```

Can be much faster than the explict call as object creation can be entirely avoided in some cases.

Parameters vector ((n_parameters,) ndarray) - A vectorized version of self

Returns pseudoinverse_vector ((n_parameters,) *ndarray*) — The pseudoinverse of the vector provided

set_h_matrix (value, copy=True, skip_checks=False)

Deprecated Deprecated - do not use this method - you are better off just creating a new transform!

Updates h_matrix, optionally performing sanity checks.

Note that it won't always be possible to manually specify the h_matrix through this method, specifically if changing the h_matrix could change the nature of the transform. See h_matrix_is_mutable for how you can discover if the h_matrix is allowed to be set for a given class.

Parameters

- **value** (*ndarray*) The new homogeneous matrix to set.
- copy (bool, optional) If False, do not copy the h_matrix. Useful for performance.
- **skip_checks** (*bool*, optional) If True, skip checking. Useful for performance.

Raises NotImplementedError - If h_matrix_is_mutable returns False.

set_target (new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) - The new target that this object should try and regenerate.

property composes_inplace_with

Affine can swallow composition with any other Affine.

property composes_with

Any Homogeneous can compose with any other Homogeneous.

property h_matrix

The homogeneous matrix defining this transform.

property h_matrix_is_mutable

Deprecated True iff set_h_matrix() is permitted on this type of transform.

If this returns False calls to set_h_matrix() will raise a NotImplementedError.

Type bool

property has_true_inverse

The pseudoinverse is an exact inverse.

Type True

property linear_component

The linear component of this affine transform.

property n_dims

The number of dimensions of the target.

Type int

property n_dims_output

The output of the data from the transform.

Type int

property n_parameters

Number of parameters of Similarity

2D Similarity - 4 parameters

3D Similarity: Currently not supported

Returns n_parameters (*int*) – The transform parameters

Raises DimensionalityError, NotImplementedError — Only 2D transforms are supported.

property n_points

The number of points on the target.

Type int

property source

The source **PointCloud** that is used in the alignment.

The source is not mutable.

Type PointCloud

property target

The current **PointCloud** that this object produces.

To change the target, use set_target().

Type PointCloud

property translation_component

The translation component of this affine transform.

Type (n_dims,) *ndarray*

DifferentiableAlignmentAffine

```
class menpofit.transform.DifferentiableAlignmentAffine (source, target)
    Bases: AlignmentAffine, DP, DX
```

Base class that constructs an affine transformation that is the optimal transform to align the *source* to the *target*. It can compute its own derivative with respect to spatial changes, as well as its parametrisation.

```
aligned source()
```

The result of applying self to source

```
Type PointCloud
```

```
alignment_error()
```

The Frobenius Norm of the difference between the target and the aligned source.

```
Type float
```

```
apply (x, batch_size=None, **kwargs)
```

Applies this transform to x.

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply () method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (int, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (*dict*) Passed through to _apply().

Returns transformed (type (x)) – The transformed object or array

```
apply_inplace (*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

```
as_non_alignment()
```

Returns the non-alignment version of the transform.

```
Type DifferentiableAffine
```

```
as_vector(**kwargs)
```

Returns a flattened representation of the object as a single vector.

Returns vector ((N,) ndarray) – The core representation of the object, flattened into a single vector. Note that this is always a view back on to the original object, but is not writable.

```
compose after(transform)
```

A **Transform** that represents **this** transform composed **after** the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

An attempt is made to perform native composition, but will fall back to a **TransformChain** as a last resort. See <code>composes_with</code> for a description of how the mode of composition is decided.

Parameters transform (Transform) - Transform to be applied before self

Returns transform (**Transform** or **TransformChain**) – If the composition was native, a single new **Transform** will be returned. If not, a **TransformChain** is returned instead.

compose_after_from_vector_inplace(vector)

Specialised inplace composition with a vector. This should be overridden to provide specific cases of composition whereby the current state of the transform can be derived purely from the provided vector.

Parameters vector ((n_parameters,) ndarray) — Vector to update the transform state with

compose_after_inplace(transform)

Update self so that it represents **this** transform composed **after** the given transform:

```
a_orig = a.copy()
a.compose_after_inplace(b)
a.apply(p) == a_orig.apply(b.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

Parameters transform (composes_inplace_with) - Transform to be applied before self

Raises ValueError - If transform isn't an instance of composes_inplace_with

compose_before (transform)

A **Transform** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

An attempt is made to perform native composition, but will fall back to a **TransformChain** as a last resort. See <code>composes_with</code> for a description of how the mode of composition is decided.

Parameters transform (Transform) - Transform to be applied after self

Returns transform (**Transform** or **TransformChain**) – If the composition was native, a single new **Transform** will be returned. If not, a **TransformChain** is returned instead.

compose_before_inplace (transform)

Update self so that it represents **this** transform composed **before** the given transform:

```
a_orig = a.copy()
a.compose_before_inplace(b)
a.apply(p) == b.apply(a_orig.apply(p))
```

a is permanently altered to be the result of the composition. b is left unchanged.

Parameters transform (composes_inplace_with) - Transform to be applied after self

Raises ValueError - If transform isn't an instance of composes_inplace_with

copy()

Generate an efficient copy of this **HomogFamilyAlignment**.

Returns new_transform (type(self)) - A copy of this object

d_dp (points)

The derivative with respect to the parametrisation changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

 $\mathbf{d}_{\mathbf{d}}\mathbf{p}$ ((n_points, n_parameters, n_dims) ndarray) – The Jacobian with respect to the parametrisation.

d_dp[i, j, k] is the scalar differential change that the k'th dimension of the i'th point experiences due to a first order change in the j'th scalar in the parametrisation vector.

d_dx (points)

The first order derivative with respect to spatial changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

d_dx ((n_points, n_dims, n_dims) *ndarray*) - The Jacobian wrt spatial changes.

d_dx[i, j, k] is the scalar differential change that the j'th dimension of the i'th point experiences due to a first order change in the k'th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

decompose()

Decompose this transform into discrete Affine Transforms.

Useful for understanding the effect of a complex composite transform.

Returns

transforms (*list* of **DiscreteAffine**) – Equivalent to this affine transform, such that

```
reduce(lambda x, y: x.chain(y), self.decompose()) == self
```

from vector(vector)

Build a new instance of the object from its vectorized state.

self is used to fill out the missing state required to rebuild a full object from it's standardized flattened state. This is the default implementation, which is a deepcopy of the object followed by a call to from_vector_inplace(). This method can be overridden for a performance benefit if desired.

Parameters vector ((n_parameters,) ndarray) - Flattened representation of the object.

Returns transform (Homogeneous) – An new instance of this class.

from_vector_inplace(vector)

Deprecated. Use the non-mutating API, from_vector.

For internal usage in performance-sensitive spots, see _from_vector_inplace()

Parameters vector ((n_parameters,) ndarray) - Flattened representation of this object

has nan values()

Tests if the vectorized form of the object contains nan values or not. This is particularly useful for objects with unknown values that have been mapped to nan values.

Returns has_nan_values (*bool*) – If the vectorized object contains nan values.

classmethod init_from_2d_shear(phi, psi, degrees=True)

Convenience constructor for 2D shear transformations about the origin.

Parameters

- **phi** (*float*) The angle of shearing in the X direction.
- **psi** (*float*) The angle of shearing in the Y direction.
- **degrees** (*bool*, optional) If True phi and psi are interpreted as degrees. If False, phi and psi are interpreted as radians.

Returns shear_transform (Affine) – A 2D shear transform.

classmethod init_identity(n_dims)

Creates an identity matrix Affine transform.

Parameters n_dims (*int*) – The number of dimensions.

Returns identity (Affine) – The identity matrix transform.

pseudoinverse()

The pseudoinverse of the transform - that is, the transform that results from swapping source and target, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

Returns transform (type (self)) – The inverse of this transform.

pseudoinverse_vector(vector)

The vectorized pseudoinverse of a provided vector instance. Syntactic sugar for:

```
self.from_vector(vector).pseudoinverse().as_vector()
```

Can be much faster than the explict call as object creation can be entirely avoided in some cases.

Parameters vector ((n_parameters,) ndarray) - A vectorized version of self

 $\begin{tabular}{ll} \textbf{Returns pseudoinverse_vector} & (\begin{tabular}{ll} \textbf{n.d.} & \textbf{p.s.} & \textbf{n.d.} &$

```
set h matrix(value, copy=True, skip checks=False)
```

Deprecated Deprecated - do not use this method - you are better off just creating a new transform!

Updates h_matrix, optionally performing sanity checks.

Note that it won't always be possible to manually specify the h_matrix through this method, specifically if changing the h_matrix could change the nature of the transform. See h_matrix_is_mutable for how you can discover if the h_matrix is allowed to be set for a given class.

Parameters

- **value** (*ndarray*) The new homogeneous matrix to set.
- copy (bool, optional) If False, do not copy the h_matrix. Useful for performance.
- **skip_checks** (*bool*, optional) If True, skip checking. Useful for performance.

Raises NotImplementedError – If h_matrix_is_mutable returns False.

set_target (new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and regenerate.

property composes_inplace_with

Affine can swallow composition with any other Affine.

property composes_with

Any Homogeneous can compose with any other Homogeneous.

property h_matrix

The homogeneous matrix defining this transform.

```
Type (n_dims + 1, n_dims + 1) ndarray
```

property h_matrix_is_mutable

Deprecated True iff set_h_matrix() is permitted on this type of transform.

If this returns False calls to set_h_matrix() will raise a NotImplementedError.

Type bool

property has_true_inverse

The pseudoinverse is an exact inverse.

Type True

property linear_component

The linear component of this affine transform.

property n_dims

The number of dimensions of the target.

Type int

property n_dims_output

The output of the data from the transform.

Type int

property n_parameters

n_dims * (n_dims + 1) parameters - every element of the matrix but the homogeneous part.

Type int

Examples

2D Affine: 6 parameters:

```
[p1, p3, p5]
[p2, p4, p6]
```

3D Affine: 12 parameters:

```
[p1, p4, p7, p10]
[p2, p5, p8, p11]
[p3, p6, p9, p12]
```

property n_points

The number of points on the target.

Type int

property source

The source **PointCloud** that is used in the alignment.

The source is not mutable.

Type PointCloud

property target

The current **PointCloud** that this object produces.

To change the target, use set_target().

Type PointCloud

property translation_component

The translation component of this affine transform.

Type (n_dims,) *ndarray*

Alignments

DifferentiablePiecewiseAffine

```
class menpofit.transform.DifferentiablePiecewiseAffine (source, target)
    Bases: CachedPWA, DL, DX
```

A differentiable Piecewise Affine Transformation.

This is composed of a number of triangles defined be a set of *source* and *target* vertices. These vertices are related by a common triangle *list*. No limitations on the nature of the triangle *list* are imposed. Points can then be mapped via barycentric coordinates from the *source* to the *target* space. Trying to map points that are not contained by any source triangle throws a *TriangleContainmentError*, which contains diagnostic information.

The transform can compute its own derivative with respect to spatial changes, as well as anchor landmark changes.

aligned_source()

The result of applying self to source

Type PointCloud

```
alignment_error()
```

The Frobenius Norm of the difference between the target and the aligned source.

Type *float*

```
apply (x, batch_size=None, **kwargs)
```

Applies this transform to x.

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply () method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (*int*, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwarqs** (*dict*) Passed through to apply ().

Returns transformed (type (x)) – The transformed object or array

apply_inplace(*args, **kwargs)

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

compose_after(transform)

Returns a TransformChain that represents this transform composed after the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

Parameters transform (Transform) – Transform to be applied before self

Returns transform (TransformChain) – The resulting transform chain.

compose_before (transform)

Returns a **TransformChain** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

Parameters transform (Transform) – Transform to be applied after self

Returns transform (TransformChain) – The resulting transform chain.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) – A copy of this object

d_dl (points)

The derivative of the warp with respect to spatial changes in anchor landmark points or centres, evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

 $\mathbf{d_dl}$ ((n_points, n_centres, n_dims) ndarray) — The Jacobian wrt landmark changes.

d_dl[i, k, m] is the scalar differential change that the any dimension of the i'th point experiences due to a first order change in the m'th dimension of the k'th landmark point.

Note that at present this assumes that the change in every dimension is equal.

d_dx (points)

The first order derivative of the warp with respect to spatial changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

d_dx ((n_points, n_dims, n_dims) *ndarray*) - The Jacobian wrt spatial changes.

 $d_dx[i, j, k]$ is the scalar differential change that the j'th dimension of the i'th point experiences due to a first order change in the k'th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

Raises TriangleContainmentError: - If any point is outside any triangle of this PWA.

index_alpha_beta(points)

Finds for each input point the index of its bounding triangle and the *alpha* and *beta* value for that point in the triangle. Note this means that the following statements will always be true:

```
alpha + beta <= 1
alpha >= 0
beta >= 0
```

for each triangle result.

Trying to map a point that does not exist in a triangle throws a *TriangleContainmentError*.

Parameters points ((K, 2) ndarray) – Points to test.

Returns

- **tri_index** ((L,) *ndarray*) Triangle index for each of the *points*, assigning each point to it's containing triangle.
- alpha ((L,) ndarray) Alpha for containing triangle of each point.
- **beta** ((L,) *ndarray*) Beta for containing triangle of each point.

Raises TriangleContainmentError – All *points* must be contained in a source triangle. Check *error.points_outside_source_domain* to handle this case.

pseudoinverse()

The pseudoinverse of the transform - that is, the transform that results from swapping *source* and *target*, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

```
Type type (self)
```

set_target (new_target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and regenerate.

property has_true_inverse

The inverse is true.

```
Type True
```

property n_dims

The number of dimensions of the target.

Type int

property n_dims_output

The output of the data from the transform.

None if the output of the transform is not dimension specific.

Type int or None

property n_points

The number of points on the target.

Type int

property n_tris

The number of triangles in the triangle list.

Type int

property source

The source **PointCloud** that is used in the alignment.

The source is not mutable.

Type PointCloud

property target

The current **PointCloud** that this object produces.

To change the target, use set_target().

Type PointCloud

property trilist

The triangle list.

Type (n_tris, 3) ndarray

DifferentiableThinPlateSplines

Bases: ThinPlateSplines, DL, DX

The Thin Plate Splines (TPS) alignment between 2D *source* and *target* landmarks. The transform can compute its own derivative with respect to spatial changes, as well as anchor landmark changes.

Parameters

- source ((N, 2) ndarray) The source points to apply the tps from
- target ((N, 2) *ndarray*) The target points to apply the tps to
- **kernel** (class or None, optional) The differentiable kernel to apply. Possible options are DifferentiableR2LogRRBF and DifferentiableR2LogR2RBF. If None, then DifferentiableR2LogR2RBF is used.

aligned_source()

The result of applying self to source

Type PointCloud

```
alignment_error()
```

The Frobenius Norm of the difference between the target and the aligned source.

```
Type float
```

```
apply (x, batch_size=None, **kwargs)
```

Applies this transform to x.

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply() method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (int, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (*dict*) Passed through to _apply().

Returns transformed (type(x)) – The transformed object or array

```
apply_inplace(*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

compose_after (transform)

Returns a TransformChain that represents this transform composed after the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

Parameters transform (Transform) – Transform to be applied before self

Returns transform (TransformChain) – The resulting transform chain.

compose_before (transform)

Returns a **TransformChain** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

Parameters transform (Transform) - Transform to be applied after self

Returns transform (TransformChain) – The resulting transform chain.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

d_dl (points)

Calculates the Jacobian of the TPS warp wrt to the source landmarks assuming that he target is equal to the source. This is a special case of the Jacobian wrt to the source landmarks that is used in AAMs to weight the relative importance of each pixel in the reference frame wrt to each one of the source landmarks.

```
\mathbf{dW\_dl} = \mathbf{dOmega\_dl} * \mathbf{k(points)} = T * d_L * -1_dl * \mathbf{k(points)} = T * -L * -1_dl L_dl L * * -1_dl L_dl L * -1_dl L *
```

per point (c, d) = (d, c+3) (c+3, c+3) (c+3, c+3, c, d) (c+3, c+3) (c+3) (c, d) = (d, c+3) (c+3, c+3, c, d) (c+3,) (c, d) = (d, c+3) (c+3, c+3, c, d)

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns dW/dl ((*n_points*, *n_params*, *n_dims*) *ndarray*) – The Jacobian of the transform wrt to the source landmarks evaluated at the previous points and assuming that the target is equal to the source.

d dx (points)

The first order derivative of this TPS warp wrt spatial changes evaluated at points.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns

d_dx ((n_points, n_dims, n_dims) *ndarray*) - The Jacobian wrt spatial changes.

 $d_{x[i, j, k]}$ is the scalar differential change that the j'th dimension of the i'th point experiences due to a first order change in the k'th dimension.

It may be the case that the Jacobian is constant across space - in this case axis zero may have length 1 to allow for broadcasting.

pseudoinverse()

The pseudoinverse of the transform - that is, the transform that results from swapping *source* and *target*, or more formally, negating the transforms parameters. If the transform has a true inverse this is returned instead.

Type type(self)

set target(new target)

Update this object so that it attempts to recreate the new_target.

Parameters new_target (PointCloud) – The new target that this object should try and regenerate.

property has_true_inverse

False

Type type

property n_dims

The number of dimensions of the target.

Type int

property n_dims_output

The output of the data from the transform.

None if the output of the transform is not dimension specific.

Type int or None

property n_points

The number of points on the target.

Type int

property source

The source **PointCloud** that is used in the alignment.

The source is not mutable.

Type PointCloud

property target

The current **PointCloud** that this object produces.

To change the target, use set_target().

Type PointCloud

RBF

DifferentiableR2LogR2RBF

```
class menpofit.transform.DifferentiableR2LogR2RBF(c)
```

Bases: R2LogR2RBF, DL

The $r^2 \log r^2$ basis function.

The derivative of this function is $2r(\log r^2 + 1)$, where r = ||x - c||.

It can compute its own derivative with respect to landmark changes.

```
apply (x, batch size=None, **kwargs)
```

Applies this transform to x.

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply() method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (int, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (*dict*) Passed through to _apply().

Returns transformed (type (x)) – The transformed object or array

```
apply_inplace(*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

compose_after(transform)

Returns a TransformChain that represents this transform composed after the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

Parameters transform (Transform) – Transform to be applied before self

Returns transform (TransformChain) – The resulting transform chain.

compose_before (transform)

Returns a TransformChain that represents this transform composed before the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

Parameters transform (Transform) - Transform to be applied after self

Returns transform (TransformChain) – The resulting transform chain.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

d_dl (points)

Apply the derivative of the basis function wrt the centres and the points given by *points*. Let *points* be x, then $2(x-c)^T(\log r_{x,l}^2+1)=2(x-c)^T(2\log r_{x,l}+1)$ where $r_{x,l}=\|x-c\|$.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns d_dl ((n_points, n_centres, n_dims) *ndarray*) – The jacobian tensor representing the first order derivative of the radius from each centre wrt the centre's position, evaluated at each point.

property n_centres

The number of centres.

Type int

property n_dims

The RBF can only be applied on points with the same dimensionality as the centres.

Type int

property n_dims_output

The result of the transform has a dimension (weight) for every centre.

Type int

DifferentiableR2LogRRBF

```
class menpofit.transform.DifferentiableR2LogRRBF (c)
Bases: R2LogRRBF, DL
```

Calculates the $r^2 \log r$ basis function.

The derivative of this function is $r(1 + 2 \log r)$, where r = ||x - c||.

It can compute its own derivative with respect to landmark changes.

```
apply (x, batch_size=None, **kwargs)
```

Applies this transform to x.

If x is **Transformable**, x will be handed this transform object to transform itself non-destructively (a transformed copy of the object will be returned).

If not, x is assumed to be an *ndarray*. The transformation will be non-destructive, returning the transformed version.

Any kwargs will be passed to the specific transform _apply () method.

Parameters

- **x** (Transformable or (n_points, n_dims) *ndarray*) The array or object to be transformed.
- batch_size (int, optional) If not None, this determines how many items from the numpy array will be passed through the transform at a time. This is useful for operations that require large intermediate matrices to be computed.
- **kwargs** (*dict*) Passed through to _apply().

Returns transformed (type (x)) – The transformed object or array

```
apply_inplace (*args, **kwargs)
```

Deprecated as public supported API, use the non-mutating apply() instead.

For internal performance-specific uses, see _apply_inplace().

compose after(transform)

Returns a TransformChain that represents this transform composed after the given transform:

```
c = a.compose_after(b)
c.apply(p) == a.apply(b.apply(p))
```

a and b are left unchanged.

This corresponds to the usual mathematical formalism for the compose operator, o.

Parameters transform (Transform) – Transform to be applied before self

Returns transform (TransformChain) – The resulting transform chain.

compose_before (transform)

Returns a **TransformChain** that represents **this** transform composed **before** the given transform:

```
c = a.compose_before(b)
c.apply(p) == b.apply(a.apply(p))
```

a and b are left unchanged.

Parameters transform (Transform) – Transform to be applied after self

Returns transform (TransformChain) – The resulting transform chain.

copy()

Generate an efficient copy of this object.

Note that Numpy arrays and other **Copyable** objects on self will be deeply copied. Dictionaries and sets will be shallow copied, and everything else will be assigned (no copy will be made).

Classes that store state other than numpy arrays and immutable types should overwrite this method to ensure all state is copied.

Returns type (self) - A copy of this object

d dl (points)

The derivative of the basis function wrt the coordinate system evaluated at *points*. Let *points* be x, then $(x-c)^T(1+2\log r_{x,l})$, where $r_{x,l}=\|x-c\|$.

Parameters points ((n_points, n_dims) *ndarray*) – The spatial points at which the derivative should be evaluated.

Returns d_dl ((n_points, n_centres, n_dims) *ndarray*) - The Jacobian wrt landmark changes.

property n_centres

The number of centres.

Type int

property n_dims

The RBF can only be applied on points with the same dimensionality as the centres.

Type int

property n_dims_output

The result of the transform has a dimension (weight) for every centre.

Type int

2.2.11 menpofit.visualize

Print Utilities

print progress

```
menpofit.visualize.print_progress(iterable, prefix=", n_items=None, offset=0, show_bar=True, show_count=True, show_eta=True, end with newline=True, verbose=True)
```

Print the remaining time needed to compute over an iterable.

To use, wrap an existing iterable with this function before processing in a for loop (see example).

The estimate of the remaining time is based on a moving average of the last 100 items completed in the loop.

This method is identical to *menpo.visualize.print_progress*, but adds a *verbose* flag which allows the printing to be skipped if necessary.

Parameters

• **iterable** (*iterable*) – An iterable that will be processed. The iterable is passed through by this function, with the time taken for each complete iteration logged.

- prefix (str, optional) If provided a string that will be prepended to the progress report at each level.
- n_items (*int*, optional) Allows for iterator to be a generator whose length will be assumed to be *n_items*. If not provided, then iterator needs to be *Sizable*.
- **offset** (*int*, optional) Useful in combination with n_items report back the progress as if *offset* items have already been handled. n_items will be left unchanged.
- **show_bar** (*bool*, optional) If False, The progress bar (e.g. [======]) will be hidden
- show_count (bool, optional) If False, The item count (e.g. (4/25)) will be hidden.
- **show_eta** (*bool*, optional) If False, The estimated time to finish (e.g. 00:00:03 remaining) will be hidden.
- end_with_newline (bool, optional) If False, there will be no new line added at the end of the dynamic printing. This means the next print statement will overwrite the dynamic report presented here. Useful if you want to follow up a print_progress with a second print_progress, where the second overwrites the first on the same line.
- **verbose** (*bool*, optional) Printing is performed only if set to True.

Raises ValueError - offset provided without n_items

Examples

This for loop:

```
from time import sleep
for i in print_progress(range(100)):
    sleep(1)
```

prints a progress report of the form:

```
[========= ] 70% (7/10) - 00:00:03 remaining
```

Errors Visualization

statistics_table

```
menpofit.visualize.statistics_table(errors, method_names, auc_max_error, auc_error_step, auc_min_error=0.0, stats_types=None, stats_names=None, sort_by=None, precision=4)
```

Function that generates a table with statistical measures on the fitting results of various methods using pandas. It supports multiple types of statistical measures.

Note that the returned object is a pandas table which can be further converted to Latex tabular or simply a string. See the examples for more details.

Parameters

- errors (list of list of float) A list that contains lists of float with the errors per method.
- **method_names** (*list* of *str*) The *list* with the names that will appear for each method. Note that it must have the same length as *errors*.

- auc_max_error (float) The maximum error value for computing the area under the curve.
- auc_error_step (float) The sampling step of the error bins for computing the area under the curve.
- auc_min_error (*float*, optional) The minimum error value for computing the area under the curve.
- **stats_types** (*list* of *str* or None, optional) The types of statistical measures to compute. Possible options are:

Value	Description
mean	The mean value of the errors.
std	The standard deviation of the errors.
median	The median value of the errors.
mad	The median absolute deviation of the errors.
max	The max value of the errors.
аис	The area under the curve based on the CED of the errors.
fr	The failure rate (percentage of images that failed).

If None, then all of them will be used with the above order.

- **stats_names** (*list* of *str*, optional) The *list* with the names that will appear for each statistical measure type selected in *stats_types*. Note that it must have the same length as *stats_types*.
- **sort_by** (*str* or None, optional) The column to use for sorting the methods. If None, then no sorting is performed and the methods will appear in the provided order of *method_names*. Possible options are:

Value	Description
mean	The mean value of the errors.
std	The standard deviation of the errors.
median	The median value of the errors.
mad	The median absolute deviation of the errors.
max	The max value of the errors.
аис	The area under the curve based on the CED of the errors.
fr	The failure rate (percentage of images that failed).

• **precision** (*int*, optional) – The precision of the reported values, i.e. the number of decimals.

Raises

- ValueError stat_type must be selected from [mean, std, median, mad, max, auc, fr]
- ValueError sort_by must be selected from [mean, std, median, mad, max, auc, fr]
- ValueError stats_types and stats_names must have the same length

Returns table (*pandas.DataFrame*) – The pandas table. It can be further converted to various format, such as Latex tabular or *str*.

Examples

Let us create some errors for 3 methods sampled from Normal distributions with different mean and standard deviations:

We can create a pandas *DataFrame* as:

Pandas offers excellent functionalities. For example, the table can be converted to an str as:

```
print(tab.to_string())
```

or to a Latex tabular as:

```
print(tab.to_latex())
```

plot cumulative error distribution

```
menpofit.visualize.plot_cumulative_error_distribution(errors,
                                                                                    error_range=None,
                                                                          figure_id=None,
                                                                          new_figure=False,
                                                                                                    ti-
                                                                          tle='Cumulative Error Distri-
                                                                          bution', x_label='Normalized
                                                                          Point-to-Point
                                                                                                Error',
                                                                          y label='Images
                                                                                               Propor-
                                                                          tion', legend_entries=None,
                                                                          render_lines=True,
                                                                          line colour=None,
                                                                          line_style='-', line_width=2,
                                                                          render_markers=True,
                                                                          marker style='s',
                                                                          marker\_size=7,
                                                                          marker_face_colour='w',
                                                                          marker_edge_colour=None,
                                                                          marker_edge_width=2,
                                                                          render_legend=True,
                                                                          legend_title=None,
                                                                          legend_font_name='sans-
                                                                          serif',
                                                                                                   leg-
                                                                          end_font_style='normal',
                                                                          legend_font_size=10,
                                                                                                   leg-
                                                                          end_font_weight='normal',
                                                                          legend marker scale=1.0,
                                                                          legend_location=2,
                                                                          end_bbox_to_anchor=(1.05,
                                                                          1.0),
                                                                                                   leg-
                                                                          end_border_axes_pad=1.0,
                                                                          legend_n\_columns=1,
                                                                          end_horizontal_spacing=1.0,
                                                                          legend_vertical_spacing=1.0,
                                                                          legend_border=True,
                                                                                                   leg-
                                                                          end_border_padding=0.5,
                                                                          legend_shadow=False,
                                                                                                   leg-
                                                                          end_rounded_corners=False,
                                                                          render_axes=True,
                                                                          axes_font_name='sans-
                                                                          serif',
                                                                                    axes\_font\_size=10,
                                                                          axes_font_style='normal',
                                                                          axes_font_weight='normal',
                                                                          axes_x_limits=None,
                                                                          axes_y_limits=None,
                                                                          axes_x_ticks=None,
                                                                          axes_y_ticks=None,
                                                                          figure\_size=(7,
                                                                                                    7),
                                                                          render_grid=True,
                                                                          grid_line_style='--',
                                                                          grid\_line\_width=0.5)
```

Plot the cumulative error distribution (CED) of the provided fitting errors.

Parameters

- errors (list of lists) A list with lists of fitting errors. A separate CED curve will be rendered for each errors list.
- error_range (list of float with length 3, optional) Specifies the horizontal axis range,
 i.e.

```
error_range[0] = min_error
error_range[1] = max_error
error_range[2] = error_step
```

If None, then 'error_range = [0., 0.101, 0.005]'.

- **figure_id** (*object*, optional) The id of the figure to be used.
- **new_figure** (*bool*, optional) If True, a new figure is created.
- title (*str*, optional) The figure's title.
- **x_label** (*str*, optional) The label of the horizontal axis.
- **y_label** (*str*, optional) The label of the vertical axis.
- **legend_entries** (*list of `str* or None, optional) If *list* of *str*, it must have the same length as *errors list* and each *str* will be used to name each curve. If None, the CED curves will be named as *'Curve %d'*.
- **render_lines** (*bool* or *list* of *bool*, optional) If True, the line will be rendered. If *bool*, this value will be used for all curves. If *list*, a value must be specified for each fitting errors curve, thus it must have the same length as *errors*.
- line_colour (*colour* or *list* of *colour* or None, optional) The colour of the lines. If not a *list*, this value will be used for all curves. If *list*, a value must be specified for each curve, thus it must have the same length as *y_axis*. If None, the colours will be linearly sampled from jet colormap. Example *colour* options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- line_style ({ '-', '--', '-.', ':'} or *list* of those, optional) The style of the lines. If not a *list*, this value will be used for all curves. If *list*, a value must be specified for each curve, thus it must have the same length as *errors*.
- line_width (*float* or *list* of *float*, optional) The width of the lines. If *float*, this value will be used for all curves. If *list*, a value must be specified for each curve, thus it must have the same length as *errors*.
- **render_markers** (*bool* or *list* of *bool*, optional) If True, the markers will be rendered. If *bool*, this value will be used for all curves. If *list*, a value must be specified for each curve, thus it must have the same length as *errors*.
- marker_style (marker or list of markers, optional) The style of the markers. If not a list, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as errors. Example marker options

```
{'.', ',', 'o', 'v', '^', '<', '>', '+', 'x', 'D', 'd', 's', 'p', '*', 'h', 'H', '1', '2', '3', '4', '8'}
```

• marker_size (*int* or *list* of *int*, optional) – The size of the markers in points. If *int*, this value will be used for all curves. If *list*, a value must be specified for each curve, thus it must have the same length as *errors*.

• marker_face_colour (colour or list of colour or None, optional) – The face (filling) colour of the markers. If not a list, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as errors. If None, the colours will be linearly sampled from jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

• marker_edge_colour (colour or list of colour or None, optional) – The edge colour of the markers. If not a list, this value will be used for all curves. If list, a value must be specified for each curve, thus it must have the same length as errors. If None, the colours will be linearly sampled from jet colormap. Example colour options are

```
{'r', 'g', 'b', 'c', 'm', 'k', 'w'}
or
(3, ) ndarray
```

- marker_edge_width (*float* or *list* of *float*, optional) The width of the markers' edge. If *float*, this value will be used for all curves. If *list*, a value must be specified for each curve, thus it must have the same length as *errors*.
- render_legend (*bool*, optional) If True, the legend will be rendered.
- **legend_title** (*str*, optional) The title of the legend.
- legend_font_name (See below, optional) The font of the legend. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- legend_font_style ({'normal', 'italic', 'oblique'}, optional) The font style of the legend.
- **legend_font_size** (*int*, optional) The font size of the legend.
- legend_font_weight (See below, optional) The font weight of the legend. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

- **legend_marker_scale** (*float*, optional) The relative size of the legend markers with respect to the original
- **legend_location** (*int*, optional) The location of the legend. The predefined values are:

'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

- legend_bbox_to_anchor ((float, float), optional) The bbox that the legend will be anchored.
- legend_border_axes_pad (float, optional) The pad between the axes and legend border.
- legend_n_columns (int, optional) The number of the legend's columns.
- legend horizontal spacing (*float*, optional) The spacing between the columns.
- **legend_vertical_spacing** (*float*, optional) The vertical space between the legend entries.
- legend_border (bool, optional) If True, a frame will be drawn around the legend.
- legend_border_padding (float, optional) The fractional whitespace inside the legend border.
- legend_shadow (bool, optional) If True, a shadow will be drawn behind legend.
- **legend_rounded_corners** (*bool*, optional) If True, the frame's corners will be rounded (fancybox).
- render_axes (bool, optional) If True, the axes will be rendered.
- axes_font_name (See below, optional) The font of the axes. Example options

```
{'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
```

- **axes_font_size** (*int*, optional) The font size of the axes.
- axes_font_style({'normal', 'italic', 'oblique'}, optional) The font style of the axes.
- axes_font_weight (See below, optional) The font weight of the axes. Example options

```
{'ultralight', 'light', 'normal', 'regular', 'book', 'medium',
  'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy',
  'extra bold', 'black'}
```

• axes_x_limits (*float* or (*float*, *float*) or None, optional) – The limits of the x axis. If *float*, then it sets padding on the right and left of the graph as a percentage of the curves' width. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set to (0., error range[1]).

- axes_y_limits (*float* or (*float*, *float*) or None, optional) The limits of the y axis. If *float*, then it sets padding on the top and bottom of the graph as a percentage of the curves' height. If *tuple* or *list*, then it defines the axis limits. If None, then the limits are set to (0., 1.).
- axes_x_ticks (list or tuple or None, optional) The ticks of the x axis.
- axes_y_ticks (list or tuple or None, optional) The ticks of the y axis.
- figure_size ((float, float) or None, optional) The size of the figure in inches.
- render_grid (bool, optional) If True, the grid will be rendered.
- grid_line_style ({ '-', '--', '-.', ':'}, optional) The style of the grid lines.
- grid_line_width (*float*, optional) The width of the grid lines.

Raises ValueError – legend_entries list has different length than errors list

Returns viewer (menpo.visualize.GraphPlotter) – The viewer object.

INDEX

A	pofit.aps.result.APSAlgorithmResult property),
AAM (class in menpofit.aam.base), 13	129
aam() (menpofit.aam.LucasKanadeAAMFitter property), 29	appearance_costs() (men- pofit.aps.result.APSResult property), 112
AAMAlgorithmResult (class in men- pofit.aam.result), 69 AAMResult (class in menpofit.aam.result), 52	appearance_model() (men- pofit.aam.AlternatingForwardCompositional property), 33
ActiveShapeModel (class in menpofit.clm), 149	appearance_model() (men-
align_shape_with_bounding_box() (in mod- ule menpofit.fitter), 313	pofit.aam.AlternatingInverseCompositional property), 34
align_shapes() (in module menpofit.builder), 287	appearance_model() (men-
<pre>aligned_source()</pre>	pofit.aam.AppearanceWeightsGaussNewton property), 45
method), 431	appearance_model() (men-
aligned_source() (men- pofit.transform.DifferentiableAlignmentSimilarity	pofit.aam.AppearanceWeightsNewton prop- erty), 43
method), 425	appearance_model() (men-
aligned_source() (men- pofit.transform.DifferentiablePiecewiseAffine	pofit.aam.MeanTemplateGaussNewton property), 48
method), 436	appearance_model() (men-
aligned_source() (men- pofit.transform.DifferentiableThinPlateSplines	<pre>pofit.aam.MeanTemplateNewton property), 46</pre>
method), 439	appearance_model() (men-
alignment_error() (men-	po fit. a am. Modified Alternating Forward Composition and the property of t
pofit.transform.DifferentiableAlignmentAffine	property), 34
method), 431	appearance_model() (men-
alignment_error() (men-	pofit. a am. Modified Alternating Inverse Composition al
pofit.transform.Differentiable Alignment Similarity	property), 35
method), 426	appearance_model() (men-
alignment_error() (men- pofit.transform.DifferentiablePiecewiseAffine	pofit.aam.ProjectOutForwardCompositional property), 36
method), 436	appearance_model() (men-
<pre>alignment_error()</pre>	pofit.aam.ProjectOutGaussNewton property), 51
method), 440	appearance_model() (men-
AlternatingForwardCompositional (class in menpofit.aam), 32	pofit.aam.ProjectOutInverseCompositional property), 37
AlternatingInverseCompositional (class in	appearance_model() (men-
menpofit.aam), 33	pofit.aam.ProjectOutNewton property), 50
${\tt Alternating Regularised Landmark Mean Shift}$	appearance_model() (men-
(class in menpofit.unified_aam_clm), 161	pofit.aam.SimultaneousForwardCompositional
appearance_costs() (men-	property), 38

appearance_model()	(men-	method), 436
pofit.aam.SimultaneousInverseComposit property), 39	ional	apply() (menpofit.transform.DifferentiableR2LogR2RBF method), 442
<pre>appearance_model()</pre>	(men-	apply() (menpofit.transform.DifferentiableR2LogRRBF method), 444
<pre>property), 40 appearance_model()</pre>	(men-	apply() (menpofit.transform.DifferentiableSimilarity method), 421
pofit.aam.WibergInverseCompositional erty), 41	prop-	apply() (menpofit.transform.DifferentiableThinPlateSplines method), 440
appearance_model() (menpofit.aps.Forward erty), 95	d prop-	apply() (menpofit.transform.LinearOrthoMDTransform method), 412
appearance_model() (menpofit.aps.Inverse erty), 94	prop-	apply() (menpofit.transform.OrthoMDTransform method), 409
appearance_parameters()	(men-	apply_inplace() (men-
pofit.aam.result.AAMAlgorithmResult erty), 85	prop-	pofit.transform.DifferentiableAffine method), 416
<pre>appearance_parameters() pofit.aam.result.AAMResult property), 6</pre>	(men- 8	<pre>apply_inplace()</pre>
<pre>appearance_parameters()</pre>	(men- MCI MAI	method), 431 geriphm <u>Re</u> isulplace() (men-
property), 196		$po {\it fit.trans} form. Differentiable A lignment Similarity$
appearance_parameters()	(men-	method), 426
pofit.unified_aam_clm.result.UnifiedAAl property), 179		$po {\it fit.trans} form. Differentiable Piecewise Affine$
appearance_reconstructions()	(men-	method), 437
pofit.aam.base.AAM method), 14 appearance_reconstructions() pofit.aam.LinearAAM method), 19	(men-	apply_inplace() (men- pofit.transform.DifferentiableR2LogR2RBF method), 442
<pre>appearance_reconstructions() pofit.aam.LinearMaskedAAM method), 2</pre>	(men- 22	<pre>apply_inplace()</pre>
appearance_reconstructions()	(men-	method), 444
pofit.aam.LucasKanadeAAMFitter m 27	ethod),	<pre>apply_inplace()</pre>
appearance_reconstructions()	(men-	method), 421
pofit.aam.MaskedAAM method), 17 appearance_reconstructions() pofit.aam.PatchAAM method), 24	(men-	apply_inplace() (men- pofit.transform.DifferentiableThinPlateSplines method), 440
appearance_reconstructions() pofit.unified_aam_clm.base.UnifiedAAM method), 156		apply_inplace() (men- pofit.transform.LinearOrthoMDTransform method), 413
appearance_reconstructions() pofit.unified_aam_clm.UnifiedAAMCLM method), 158	(men- Fitter	apply_inplace() (men- pofit.transform.OrthoMDTransform method), 410
AppearanceWeightsGaussNewton (class i pofit.aam), 43	n men-	aps () (menpofit.aps.GaussNewtonAPSFitter property), 93
AppearanceWeightsNewton (class in pofit.aam), 42	men-	APSAlgorithmResult (class in menpofit.aps.result), 113
* *	leAffine	APSResult (class in menpofit.aps.result), 96 area_under_curve_and_failure_rate() (in
apply()(<i>menpofit.transform.DifferentiableAlign</i>	ımentAffi	ine module menpofit.error), 304
<pre>method), 431 apply() (menpofit.transform.DifferentiableAlign method), 426</pre>	ımentSim	as_non_alignment() (men- nilarity pofit.transform.DifferentiableAlignmentAffine method), 431
apply() (menpofit.transform.DifferentiablePiece	ewiseAffi	

pofit.transform.DifferentiableAlignmentSi	milarity	
method), 426	IDDIA	pofit.atm.MaskedATM method), 133
_	PDM	build_fitter_interfaces() (men-
method), 330	•7	pofit.atm.PatchATM method), 139
<pre>as_vector() (menpofit.modelinstance.GlobalSin method), 327</pre>	nilarityl	Whodeld_fitter_interfaces() (men- pofit.unified_aam_clm.base.UnifiedAAMCLM
as_vector() (menpofit.modelinstance.ModelIns	stance	method), 157
method), 325		<pre>build_patch_reference_frame() (in module</pre>
as_vector() (menpofit.modelinstance.Ortho		menpofit.builder), 288
method), 333		build_reference_frame() (in module men-
as_vector() (menpofit.modelinstance.PDM mea		pofit.builder), 288
328	,	pojimomuler), 2 00
as_vector() (menpofit.transform.Differentiable.	Affine	C
method), 416	rijjuic	
as_vector() (menpofit.transform.Differentiable.	Alianma	check_algorithm_cls() (in module men-
method), 431	nugnme	
	Alionne	check_callable() (in module menpofit.checks), 294
as_vector() (menpoju.transjorm.Dijjerenilable)	Augnme	ausiwikarity agonal () (in module menpofit.checks), 292
method), 426	aı .	check_graph() (in module menpofit.checks), 296
as_vector() (menpojii.transform.Differentiable.	Similari	teheck_landmark_trilist() (in module men-
method), 421	.	pofit.checks), 292
as_vector() (menpofit.transform.LinearOrthoM	IDTrans	Teffeck_max_components() (in module men-
method), 413		pofit.checks), 295
as_vector() (menpofit.transform.OrthoMDTran method), 410	ısform	check_max_iters() (in module menpofit.checks), 294
ATM (class in menpofit.atm.base), 130		check_model() (in module menpofit.checks), 293
atm() (menpofit.atm.LucasKanadeATMFitter prop	perty).	check_multi_scale_param() (in module men-
142		pofit.checks), 293
		* *
В		check_patch_shape() (in module menpofit.checks), 294
bb_area() (in module menpofit.error), 298		
	man	check_sampling() (in module menpofit.checks), 296
bb_avg_edge_length() (in module pofit.error), 298	men-	check_scales() (in module menpofit.checks), 293
		check_trilist() (in module menpofit.checks), 293
bb_avg_edge_length_49_euclidean_err	ror()	CLM (class in menpofit.clm), 145
(in module menpofit.error), 306	()	clm() (menpofit.clm.GradientDescentCLMFitter prop-
bb_avg_edge_length_68_euclidean_err	cor()	erty), 148
(in module menpofit.error), 306		compose_after() (men-
bb_diagonal() (in module menpofit.error), 299		pofit.transform.DifferentiableAffine method),
bb_perimeter() (in module menpofit.error), 29	18	416
<u> </u>	(men-	compose_after() (men-
pofit.aam.base.AAM method), 14		pofit.trans form. Differentiable Alignment Affine
	(men-	method), 431
pofit.aam.LinearAAM method), 20		compose_after() (men-
<pre>build_fitter_interfaces()</pre>	(men-	pofit.transform.DifferentiableAlignmentSimilari
pofit.aam.LinearMaskedAAM method), 22	2	method), 426
<pre>build_fitter_interfaces()</pre>	(men-	compose_after() (men-
pofit.aam.MaskedAAM method), 17		pofit.transform.DifferentiablePiecewiseAffine
	(men-	method), 437
pofit.aam.PatchAAM method), 25		compose_after() (men-
	(men-	pofit.transform.DifferentiableR2LogR2RBF
pofit.atm.base.ATM method), 131		method), 443
	(men-	
pofit.atm.LinearATM method), 135	(men-	compose_after() (men-
* *	(men-	pofit.transform.DifferentiableR2LogRRBF
pofit.atm.LinearMaskedATM method), 13'	`	method), 444
pojii.aim.LinearWaskeaATW meilloa), 13	1	

compose_after() (men-	compose_before() (men-
pofit.transform.DifferentiableSimilarity	pofit.transform.DifferentiableR2LogRRBF
method), 421	method), 444
compose_after() (men-	compose_before() (men-
pofit.transform.DifferentiableThinPlateSplines	pofit.transform.DifferentiableSimilarity
method), 440	method), 422
compose_after() (men-	compose_before() (men-
pofit.transform.LinearOrthoMDTransform	pofit.transform.DifferentiableThinPlateSplines
method), 413	method), 440
compose_after() (men-	compose_before() (men-
pofit.transform.OrthoMDTransform method),	pofit.transform.LinearOrthoMDTransform
410	method), 413
<pre>compose_after_from_vector_inplace() (menpofit.transform.DifferentiableAffine method), 417</pre>	compose_before() (men- pofit.transform.OrthoMDTransform method), 410
<pre>compose_after_from_vector_inplace() (menpofit.transform.DifferentiableAlignmentAffin method), 432</pre>	compose_before_inplace() (men- ne pofit.transform.DifferentiableAffine method), 417
<pre>compose_after_from_vector_inplace() (menpofit.transform.DifferentiableAlignmentSimi method), 426</pre>	compose_before_inplace() (men- larity pofit.transform.DifferentiableAlignmentAffine method), 432
<pre>compose_after_from_vector_inplace() (menpofit.transform.DifferentiableSimilarity method), 422</pre>	compose_before_inplace() (men- pofit.transform.DifferentiableAlignmentSimilarity method), 427
<pre>compose_after_from_vector_inplace() (menpofit.transform.OrthoMDTransform method), 410</pre>	compose_before_inplace() (men- pofit.transform.DifferentiableSimilarity method), 422
compose_after_inplace() (men-	composes_inplace_with() (men-
pofit.transform.DifferentiableAffine method),	pofit.transform.DifferentiableAffine property),
417	420
<pre>compose_after_inplace() (men- pofit.transform.DifferentiableAlignmentAffine method), 432</pre>	composes_inplace_with() (men- pofit.transform.DifferentiableAlignmentAffine property), 435
compose_after_inplace() (men-	composes_inplace_with() (men-
pofit.transform.DifferentiableAlignmentSimilarity	pofit.transform.DifferentiableAlignmentSimilarity
method), 427	property), 429
<pre>compose_after_inplace() (men- pofit.transform.DifferentiableSimilarity method), 422</pre>	composes_inplace_with() (men- pofit.transform.DifferentiableSimilarity prop- erty), 424
compose_before() (men-	composes_with() (men-
pofit.transform.DifferentiableAffine method),	pofit.transform.DifferentiableAffine property),
417	420
compose_before() (men-	composes_with() (men-
pofit.transform.DifferentiableAlignmentAffine	pofit.transform.DifferentiableAlignmentAffine
method), 432	property), 435
compose_before() (men-	composes_with() (men-
pofit.transform.DifferentiableAlignmentSimilarity	pofit.transform.DifferentiableAlignmentSimilarity
method), 427	property), 429
compose_before() (men-	composes_with() (men-
pofit.transform.DifferentiablePiecewiseAffine	pofit.transform.DifferentiableSimilarity prop-
method), 437	erty), 424
compose_before() (men- pofit.transform.DifferentiableR2LogR2RBF method), 443	<pre>compute_cumulative_error() (in module men-</pre>

	288	costs($) \ (menpo fit. result. Multi Scale Non Parametric Iterative Result$
comput	e_reference_shape() (in module men-		property), 391
	pofit.builder), 289	costs() (menpofit.result.MultiScaleParametricIterativeResult
comput	e_statistical_measures() (in module		property), 408
	menpofit.error), 305	costs() (menpofit.result.NonParametricIterativeResult
сору()	(menpofit.modelinstance.GlobalPDM method), 331	202+2/	property), 357) (menpofit.result.ParametricIterativeResult
conv()	(menpofit.modelinstance.GlobalSimilarityModel	costs(property), 374
сору ()	method), 327	costs() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmRe
copy()	$(menpo {\it fit.} model in stance. Model {\it Instance}$		property), 196
	method), 325	costs() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult
copy()			property), 179
	333	D	
	(menpofit.modelinstance.PDM method), 328	D	
copy()	(menpofit.transform.DifferentiableAffine		(menpofit.differentiable.DL method), 297
	method), 417		(menpofit.transform.DifferentiablePiecewiseAffine
copy()	(menpofit.transform.DifferentiableAlignmentAffine		method), 437
()	method), 432		(menpofit.transform.DifferentiableR2LogR2RBF
copy()	(menpofit.transform.DifferentiableAlignmentSimil		method), 443
()	method), 427	d_dl()	
coby()	(menpofit.transform.DifferentiablePiecewiseAffine method), 437		method), 445
2021/		d_d1()	(menpofit.transform.DifferentiableThinPlateSplines
сору ()	(menpofit.transform.DifferentiableR2LogR2RBF method), 443	1 1 ()	method), 441
conv()	(menpofit.transform.DifferentiableR2LogRRBF		(menpofit.differentiable.DP method), 297
сору ()	method), 445	a_ap()	(menpofit.modelinstance.GlobalPDM method), 331
copy()	(menpofit.transform.DifferentiableSimilarity	d dn ()	
сору ()	method), 422	a_ap()	(menpofit.modelinstance.GlobalSimilarityModel method), 327
() vgoo	(menpofit.transform.DifferentiableThinPlateSpline	281 dn ()	(menpofit.modelinstance.ModelInstance
0011()	method), 440	-u_up()	method), 325
() vgoo	(menpofit.transform.LinearOrthoMDTransform	d_dp()	(menpofit.modelinstance.OrthoPDM method),
1 2 17	method), 413	a_ap ()	333
сору()	(menpofit.transform.OrthoMDTransform	d dp()	(menpofit.modelinstance.PDM method), 329
1 2	method), 411	d_dp()	(menpofit.transform.DifferentiableAffine
Correl	ationFilterExpertEnsemble (class in	<u>a_ap</u> ()	method), 418
	menpofit.clm), 151	d dp()	(menpofit.transform.DifferentiableAlignmentAffine
cost_c	losure() (menpofit.lk.ECC method), 209	1 (/	method), 433
cost_c	losure() (menpofit.lk.FourierSSD method),	d dp()	(menpofit.transform.DifferentiableAlignmentSimilarity
	208		method), 427
cost_c	losure() (menpofit.lk.GradientCorrelation	d_dp()	(menpofit.transform.DifferentiableSimilarity
	method), 212		method), 423
cost_c	losure() (menpofit.lk.GradientImages	d_dp()	(menpofit.transform.LinearOrthoMDTransform
	method), 211		method), 414
cost_c	losure() (menpofit.lk.SSD method), 206	d_dp()	(menpofit.transform.OrthoMDTransform
costs(method), 411
	property), 85	d_dx()	(menpofit.differentiable.DX method), 297
) (menpofit.aam.result.AAMResult property), 68	$d_dx()$	$(menpo {\it fit.trans} form. Differentiable Affine$
costs(method), 418
	property), 129	d_dx ()	$(menpo {\it fit.trans} form. Differentiable A lignment Affine$
) (menpofit.aps.result.APSResult property), 112		method), 433
costs() (menpofit.lk.result.LucasKanadeAlgorithmResult	$t_{d_dx}()$	(menpo fit.transform. Differentiable A lignment Similarity
	property), 247		method), 428
costs() (menpofit.lk.result.LucasKanadeResult prop-	d_dx()	(menpofit.transform.DifferentiablePiecewiseAffine
	erty), 230		method), 438

<pre>d_dx() (menpofit.transform.DifferentiableSimilarity</pre>	method), 231
method), 423	displacements() (men-
<pre>d_dx() (menpofit.transform.DifferentiableThinPlateSplin method), 441</pre>	pes pofit.lk.result.LucasKanadeResult method), 214
decompose() (menpofit.transform.DifferentiableAffine	displacements() (men-
method), 418	pofit.result.MultiScaleNonParametricIterativeResult
${\tt decompose()} \ (\textit{menpofit.transform.DifferentiableAlignm}$	entAffine method), 375
method), 433	displacements() (men-
decompose() (menpofit.transform.DifferentiableAlignm method), 428	entSimilari t yofit.result.MultiScaleParametricIterativeResult method), 392
decompose() (menpofit.transform.DifferentiableSimilar	
method), 423	pofit.result.NonParametricIterativeResult
deformation_costs() (men-	method), 342
pofit.aps.result.APSAlgorithmResult property),	displacements() (men-
129	pofit.result.ParametricIterativeResult method),
deformation_costs() (men-	358
pofit.aps.result.APSResult property), 112	displacements() (men-
<pre>deformation_model() (menpofit.aps.Forward</pre>	pofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 181
<pre>deformation_model() (menpofit.aps.Inverse prop-</pre>	displacements() (men-
erty), 94	pofit.unified_aam_clm.result.UnifiedAAMCLMResult
dense_target() (men-	method), 163
pofit.transform.Linear OrthoMDT ransform	displacements_stats() (men-
property), 415	$po {\it fit.} aam. {\it result.} AAM Algorithm Result\ method),$
densify_shapes() (in module menpofit.builder),	70
289	displacements_stats() (men-
DifferentiableAffine (class in men-	pofit.aam.result.AAMResult method), 52
pofit.transform), 416	displacements_stats() (men-
DifferentiableAlignmentAffine (class in menpofit.transform), 431	pofit.aps.result.APSAlgorithmResult method), 114
DifferentiableAlignmentSimilarity (class in menpofit.transform), 425	displacements_stats() (men- pofit.aps.result.APSResult method), 96
DifferentiablePiecewiseAffine (class in	displacements_stats() (men-
menpofit.transform), 436	pofit.lk.result.Lucas Kanade Algorithm Result
DifferentiableR2LogR2RBF (class in men-	method), 231
pofit.transform), 442	displacements_stats() (men-
DifferentiableR2LogRRBF (class in men- pofit.transform), 444	pofit.lk.result.LucasKanadeResult method), 214
DifferentiableSimilarity (class in men-	displacements_stats() (men-
pofit.transform), 421	$po {\it fit.} result. Multi Scale Non Parametric Iterative Result$
${\tt DifferentiableThinPlateSplines} \ \ \textit{(class in}$	method), 375
menpofit.transform), 439	displacements_stats() (men-
displacements() (men-	$po {\it fit.} result. Multi Scale Parametric Iterative Result$
pofit.aam.result.AAMAlgorithmResult method),	method), 392
69	displacements_stats() (men-
<pre>displacements() (menpofit.aam.result.AAMResult method), 52</pre>	pofit.result.NonParametricIterativeResult method), 342
displacements() (men-	displacements_stats() (men-
pofit.aps.result.APSAlgorithmResult method), 114	pofit.result.ParametricIterativeResult method), 358
displacements() (menpofit.aps.result.APSResult	displacements_stats() (men-
method), 96	$po fit. unified_a am_clm. result. Unified AAMCL MAlgorithm Result$
displacements() (men-	method), 181
pofit.lk.result.Lucas Kanade Algorithm Result	displacements_stats() (men-

pofit.unified_aam_clm.result.UnifiedAAMCLMRe method), 163	Restultnal_error() (menpofit.aps.result.APSResult method), 97
pofit.error), 299	pofit.lk.result.LucasKanadeAlgorithmResult
DL (class in menpofit.differentiable), 297	method), 232
Dlibert (class in menpofit.dlib), 197	final_error() (men-
DlibWrapper (class in menpofit.dlib), 200	pofit.lk.result.LucasKanadeResult method),
DP (class in menpofit.differentiable), 297	214
DX (class in menpojit.differentiable), 297	final_error() (men-
21 (class in menpojnanjjerennasie), 257	pofit.result.MultiScaleNonParametricIterativeResult
E	method), 376
ECC (class in menpofit.lk), 209	final_error() (men-
errors() (menpofit.aam.result.AAMAlgorithmResult	pofit.result.MultiScaleParametricIterativeResult
method), 70	method), 393
errors() (menpofit.aam.result.AAMResult method), 52	final_error() (men-
errors() (menpofit.aps.result.APSAlgorithmResult	pofit.result.NonParametricIterativeResult
method), 114	method), 342
errors () (menpofit.aps.result.APSResult method), 96	final_error() (men-
errors () (menpofit.lk.result.LucasKanadeAlgorithmResu	pofit.result.ParametricIterativeResult method),
method), 232	359
errors() (menpofit.lk.result.LucasKanadeResult	<pre>final_error() (menpofit.result.Result method), 335</pre>
method), 214	final_error() (men-
errors() (menpofit.result.MultiScaleNonParametricItera	$rative Result po fit. unified_aam_clm. result. Unified AAMCLM Algorithm Result$
method), 375	method), 181
errors() (menpofit.result.MultiScaleParametricIterative	reResulal_error() (men-
method), 392	pofit.unified_aam_clm.result.UnifiedAAMCLMResult
errors() (menpofit.result.NonParametricIterativeResult	t method), 164
method), 342	final_shape() (men-
errors() (menpofit.result.ParametricIterativeResult	pofit.aam.result.AAMAlgorithmResult prop-
method), 359	erty), 85
errors() (menpofit.unified_aam_clm.result.UnifiedAAM	ACLMAlgofillmaresult (menpofit.aam.result.AAMResult
method), 181	property), 68
errors() (menpofit.unified_aam_clm.result.UnifiedAAM	MCLIMResuthape() (men-
method), 163	pofit. aps. result. APSA lgorithmResult property),
euclidean_bb_normalised_error() (in mod-	129
ule menpofit.error), 302	final_shape() (menpofit.aps.result.APSResult prop-
euclidean_distance_indexed_normalised_er	error() <i>erty</i>), 112
(in module menpofit.error), 303	final_shape() (men-
<pre>euclidean_distance_normalised_error()</pre>	pofit.lk.result.LucasKanadeAlgorithmResult
(in module menpofit.error), 303	property), 247
euclidean_error() (in module menpofit.error), 302	final_shape() (men-
<pre>extract_patches() (in module menpofit.builder),</pre>	pofit.lk.result.LucasKanadeResult property),
289	230
F	final_shape() (men-
Г	pofit.result.MultiScaleNonParametricIterativeResult
final_error() (men-	property), 391
pofit.aam.result.AAMAlgorithmResult method),	final_shape() (men-
70	pofit.result.MultiScaleParametricIterativeResult
final_error() (menpofit.aam.result.AAMResult	property), 408
method), 53	final_shape() (men-
final_error() (men-	pofit.result.NonParametricIterativeResult
pofit.aps.result.APSAlgorithmResult method),	property), 357
114	final_shape() (men- pofit result ParametricIterativeResult prop-
	DOMETENNIE LATAMETICAETALIVENENAL DIOD-

erty), 374	pofit.atm.LucasKanadeATMFitter method),
<pre>final_shape() (menpofit.result.Result property), 341</pre>	142
final_shape() (men-	<pre>fit_from_shape()</pre> (men-
pofit.unified_aam_clm.result.UnifiedAAMCLMA	lgorithmRe pd fit.clm.GradientDescentCLMFitter method),
property), 196	148
<pre>final_shape() (men-</pre>	<pre>fit_from_shape() (menpofit.dlib.DlibERT method),</pre>
pofit.unified_aam_clm.result.UnifiedAAMCLMR	esult 199
property), 179	<pre>fit_from_shape() (menpofit.dlib.DlibWrapper</pre>
<pre>fit_from_bb() (men-</pre>	method), 200
pofit.aam.LucasKanadeAAMFitter method),	<pre>fit_from_shape() (men-</pre>
27	pofit.fitter.MultiScaleNonParametricFitter
<pre>fit_from_bb() (men-</pre>	method), 310
pofit.aam.SupervisedDescentAAMFitter	fit_from_shape() (men-
method), 30	pofit.fitter.MultiScaleParametricFitter method),
fit_from_bb() (men-	312
pofit.aps.GaussNewtonAPSFitter method),	fit_from_shape() (men-
92	pofit.io.PickleWrappedFitter method), 318
fit_from_bb() (men-	fit_from_shape() (menpofit.lk.LucasKanadeFitter
pofit.atm.LucasKanadeATMFitter method),	method), 202
141	fit_from_shape() (menpofit.sdm.RegularizedSDM
fit_from_bb() (men-	method), 255
pofit.clm.GradientDescentCLMFitter method),	fit_from_shape() (men-
147	pofit.sdm.SupervisedDescentFitter method),
fit_from_bb() (menpofit.dlib.DlibERT method), 199	250 method),
<pre>fit_from_bb() (menpofit.dlib.DlibWrapper method),</pre>	fit_from_shape() (men-
200	pofit.unified_aam_clm.UnifiedAAMCLMFitter
fit_from_bb() (men-	method), 159
pofit.fitter.MultiScaleNonParametricFitter	Forward (class in menpofit.aps), 95
method), 310	ForwardAdditive (class in menpofit.lk), 203
fit_from_bb() (men-	ForwardCompositional (class in menpofit.atm),
pofit.fitter.MultiScaleParametricFitter method),	143
312	ForwardCompositional (class in menpofit.lk), 204
<pre>fit_from_bb() (menpofit.io.PickleWrappedFitter</pre>	FourierSSD (class in menpofit.lk), 208
method), 318	<pre>frequency_filter_images() (men-</pre>
<pre>fit_from_bb() (menpofit.lk.LucasKanadeFitter</pre>	$po {\it fit.clm.} Correlation Filter Expert Ensemble$
method), 202	property), 152
<pre>fit_from_bb() (menpofit.sdm.RegularizedSDM</pre>	<pre>from_vector() (menpofit.modelinstance.GlobalPDM</pre>
method), 254	method), 331
<pre>fit_from_bb() (men-</pre>	from_vector() (men-
pofit.sdm.SupervisedDescentFitter method),	pofit.modelinstance.GlobalSimilarityModel
250	method), 327
fit from bb() (men-	from_vector() (men-
pofit.unified_aam_clm.UnifiedAAMCLMFitter	pofit.modelinstance.ModelInstance method),
method), 159	325
<pre>fit_from_shape()</pre>	<pre>from_vector() (menpofit.modelinstance.OrthoPDM</pre>
pofit.aam.LucasKanadeAAMFitter method),	method), 333
28	from_vector() (menpofit.modelinstance.PDM
fit_from_shape() (men-	method), 329
pofit.aam.SupervisedDescentAAMFitter (men	from_vector() (men-
method), 31	pofit.transform.DifferentiableAffine method),
fit_from_shape() (men-	poju.transjorm.DijjereniuweAjjine meinow), 418
pofit.aps.GaussNewtonAPSFitter method),	
92.	from_vector() (men-
	pofit.transform.DifferentiableAlignmentAffine
fit_from_shape() (men-	method), 433

from_vector() (men	- G
pofit.transform.DifferentiableAlignmentSimilar	rity GaussNewtonAPSFitter (class in menpofit.aps), 91
method), 428	generate_perturbations_from_gt() (in mod-
from_vector() (men	ule menpofit.fitter), 314
pofit.transform.DifferentiableSimilarity	GenerativeAPS (class in menpofit.aps), 87
method), 423	global_parameters() (men-
from_vector() (men	
pofit.transform.Linear OrthoMDT ransform	332
method), 414	global_parameters() (men-
from_vector() (men	pofit.modelinstance.OrthoPDM property),
pofit.transform.OrthoMDTransform method	334
411	global_parameters() (men-
from_vector_inplace() (men	pofit.transform.LinearOrthoMDTransform
pofit.modelinstance.GlobalPDM method)	property), 415
331	GlobalPDM (class in menpofit.modelinstance), 330
from_vector_inplace() (men	GlobalSimilarityModel (class in men-
$po {\it fit.} model in stance. Global Similarity Model$	pofit.modelinstance), 327
method), 327	gradient () (menpofit.lk.ECC class method), 209
from_vector_inplace() (men	gradient() (menpofit.lk.FourierSSD class method).
pofit.modelinstance.ModelInstance method	208
326	gradient() (menpofit.lk.GradientCorrelation class
from_vector_inplace() (men	memou), 212
pofit.modelinstance.OrthoPDM method	gradient() (menpofit.lk.GradientImages class
333	method), 211
from_vector_inplace() (men	gradient() (menpofit.lk.SSD class method), 206
pofit.modelinstance.PDM method), 329	GradientCorrelation (class in menpofit.lk), 212
from_vector_inplace() (men	Gradienthescentchmilter (class in men-
pofit.transform.DifferentiableAffine method	poju.cim), 147
418	GradientImages (class in menpofit.lk), 211
from_vector_inplace() (men	gc_snape() (menpojn.aam.resun.aannangorummesun
pofit.transform.DifferentiableAlignmentAffine	property), 85
<pre>method), 433 from_vector_inplace() (men</pre>	gt_shape()
from_vector_inplace() (men	- erty), 68
method), 428	rity gt_shape() (menpofit.aps.result.APSAlgorithmResult
from_vector_inplace() (men	property), 129
pofit.transform.DifferentiableSimilarity (men	gt_snape() (menpoju.ups.resuu.AF Skesuu property),
method), 423	112
from_vector_inplace() (men	gt_shape() (menpofit.lk.result.LucasKanadeAlgorithmResult
pofit.transform.LinearOrthoMDTransform	property), 247
method), 414	gt_shape() (menpofit.lk.result.LucasKanadeResult
from_vector_inplace() (men	property), 230
pofit.transform.OrthoMDTransform method	gc_snape() (menpojn.resun.muniscutervom urametricherutiveresun
411	<pre>property), 391 gt_shape() (menpofit.result.MultiScaleParametricIterativeResult</pre>
FullyParametricMeanTemplateNewton (clas	s property), 408
in menpofit.sdm), 282	gt_shape() (menpofit.result.NonParametricIterativeResult
FullyParametricProjectOutGaussNewton	property), 357
(class in menpofit.sdm), 281	gt_shape() (menpofit.result.ParametricIterativeResult
FullyParametricProjectOutNewton (class in	property), 374
menpofit.sdm), 279	gt_shape() (menpofit.result.Result property), 341
FullyParametricProjectOutOPP (class in men	gt_shape() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorith
pofit.sdm), 286	property), 196
${\tt FullyParametricWeightsNewton}\ ({\it class\ in\ men}$	gt_shape() (menpofit.unified_aam_clm.result.UnifiedAAMCLMResult

property), 179

pofit.sdm), 284

Н	$po {\it fit.trans} for {\it m.Differentiable Affine property}),$
h_matrix() (menpofit.transform.DifferentiableAffine	420
property), 420	has_true_inverse() (men-
h_matrix() (menpofit.transform.DifferentiableAlignmen property), 435	ntAffine pofit.transform.DifferentiableAlignmentAffine property), 435
h_matrix()(<i>menpofit.transform.DifferentiableAlignmen</i>	ndSimilative_inverse() (men-
property), 429	pojit.transjorm.DifferentiableAlignmentSimilarit
h_matrix() (menpofit.transform.DifferentiableSimilarity	property), 430
property), 424	has_true_inverse() (men-
h_matrix_is_mutable() (men-	pofit.transform.DifferentiablePiecewiseAffine
pofit.transform.DifferentiableAffine property),	property), 438
420	has_true_inverse() (men-
h_matrix_is_mutable() (men-	pofit.transform.DifferentiableSimilarity prop-
po fit. trans form. Differentiable Alignment Affine	erty), 425
property), 435	has_true_inverse() (men-
h_matrix_is_mutable() (men-	pofit.transform.DifferentiableThinPlateSplines
pofit.transform.DifferentiableAlignmentSimilarity	, property), 441 has_true_inverse() (men-
property), 430	pofit.transform.OrthoMDTransform property),
h_matrix_is_mutable() (men-	412
pofit.transform.DifferentiableSimilarity prop-	hessian () (menpofit.lk.ECC method), 209
erty), 425 has nan values() (men-	hessian() (menpofit.lk.FourierSSD method), 208
has_nan_values() (men- pofit.modelinstance.GlobalPDM method),	hessian() (menpofit.lk.GradientCorrelation method),
331	212
has_nan_values() (men-	hessian() (menpofit.lk.GradientImages method), 211
pofit.modelinstance.GlobalSimilarityModel	hessian() (menpofit.lk.SSD method), 207
method), 327	holistic_features() (men-
has_nan_values() (men-	pofit.aam.LucasKanadeAAMFitter property),
pofit.modelinstance.ModelInstance method),	29
326	holistic_features() (men-
has_nan_values() (men-	pofit. aam. Supervised Descent AAMF itter
pofit.modelinstance.OrthoPDM method),	property), 32
333	holistic_features() (men-
has_nan_values() (menpofit.modelinstance.PDM method), 329	pofit.aps.GaussNewtonAPSFitter property), 93
has_nan_values() (men-	holistic_features() (men-
pofit.transform.DifferentiableAffine method), 419	pofit.atm.LucasKanadeATMFitter property), 142
has_nan_values() (men-	holistic_features() (men-
pofit.transform.DifferentiableAlignmentAffine	pofit.clm.GradientDescentCLMFitter prop-
method), 433	erty), 148
has_nan_values() (men-	holistic_features() (menpofit.dlib.DlibERT
pofit.transform.DifferentiableAlignmentSimilarity	property), 199
method), 428	holistic_features() (men-
has_nan_values() (men-	pofit.fitter. MultiScale Non Parametric Fitter
pofit.transform.DifferentiableSimilarity	property), 311
method), 423	holistic_features() (men-
has_nan_values() (men-	pofit.fitter.MultiScaleParametricFitter prop-
pofit.transform.LinearOrthoMDT ransform	erty), 313
method), 414	holistic_features() (men-
has_nan_values() (men-	pofit.lk.LucasKanadeFitter property), 203
$po {\it fit.trans} for {\it m.OrthoMDT} rans for {\it m.Or$	holistic_features() (men-
411	pofit.sdm.RegularizedSDM property), 256
has_true_inverse() (men-	holistic_features() (men-

```
pofit.sdm.SupervisedDescentFitter property),
                                                 increment () (menpofit.aam.PatchAAM method), 25
        251
                                                  increment() (menpofit.aam.ProjectOutGaussNewton
holistic features()
                                           (men-
                                                          method), 50
        pofit.unified_aam_clm.UnifiedAAMCLMFitter
                                                                     (menpofit.aam.ProjectOutNewton
                                                 increment()
        property), 160
                                                          method), 49
HolisticAAM (in module menpofit.aam), 16
                                                 increment() (menpofit.aam.SupervisedDescentAAMFitter
HolisticATM (in module menpofit.atm), 132
                                                          method), 31
homogeneous parameters()
                                                 increment () (menpofit.aps.GenerativeAPS method),
                                           (men-
        pofit.lk.result.LucasKanadeAlgorithmResult
        property), 247
                                                  increment () (menpofit.atm.base.ATM method), 131
homogeneous_parameters()
                                           (men-
                                                 increment () (menpofit.atm.LinearATM method), 136
        pofit.lk.result.LucasKanadeResult
                                                                     (menpofit.atm.LinearMaskedATM
                                       property),
                                                 increment()
                                                          method), 138
                                                                 (menpofit.atm.MaskedATM method),
                                                  increment()
I
                                                          134
IIRLRegression (class in menpofit.math), 319
                                                 increment () (menpofit.atm.PatchATM method), 140
                                                 increment () (menpofit.clm.CLM method), 146
            (menpofit.aam.result.AAMAlgorithmResult
image()
        property), 85
                                                 increment() (menpofit.clm.CorrelationFilterExpertEnsemble
                                                          method), 151
image () (menpofit.aam.result.AAMResult property), 68
                                                 increment() (menpofit.clm.IncrementalCorrelationFilterThinWrapper
image()
             (menpofit.aps.result.APSAlgorithmResult
        property), 129
                                                          method), 153
image() (menpofit.aps.result.APSResult property), 112
                                                 increment()
                                                                       (menpofit.math.IIRLRegression
                                                          method), 319
image() (menpofit.lk.result.LucasKanadeAlgorithmResult
        property), 247
                                                 increment () (menpofit.math.IRLRegression method),
                                                          319
image() (menpofit.lk.result.LucasKanadeResult prop-
        erty), 230
                                                 increment()
                                                                       (menpofit.math.OPPRegression
image () (menpofit.result.MultiScaleNonParametricIterativeResult method), 321
                                                 increment() (menpofit.math.OptimalLinearRegression
        property), 391
image() (menpofit.result.MultiScaleParametricIterativeResult
                                                          method), 321
                                                  increment()
                                                                       (menpofit.math.PCRRegression
        property), 408
image() (menpofit.result.NonParametricIterativeResult
                                                          method), 320
                                                 increment()
                                                                  (menpofit.modelinstance.GlobalPDM
        property), 357
            (menpo fit.result. Parametric Iterative Result
                                                          method), 331
image()
                                                 increment()
                                                                   (menpo fit.modelin stance. Or tho PDM
        property), 374
                                                          method), 334
image () (menpofit.result.Result property), 341
\verb|image|| () (menpofit.unified\_aam\_clm.result.UnifiedAAMCLiMAlgorithmResulmenpofit.modelinstance.PDM method),
        property), 196
                                                          329
method), 283
        property), 179
                                                 increment() (menpofit.sdm.FullyParametricProjectOutGaussNewton
imccf() (in module menpofit.math), 323
imosse() (in module menpofit.math), 324
                                                          method), 281
increment () (menpofit.aam.AppearanceWeightsGaussNewtonement () (menpofit.sdm.FullyParametricProjectOutNewton
        method), 44
                                                          method), 280
method), 286
        method), 42
                                                 increment() (menpofit.sdm.FullyParametricWeightsNewton
increment () (menpofit.aam.base.AAM method), 15
increment () (menpofit.aam.LinearAAM method), 20
                                                          method), 284
                                                 increment() (menpofit.sdm.NonParametricGaussNewton
increment()
                   (menpofit.aam.LinearMaskedAAM
        method), 22
                                                          method), 258
increment () (menpofit.aam.MaskedAAM method), 17
                                                 increment()
                                                                  (menpofit.sdm.NonParametricNewton
                                                          method), 257
increment() (menpofit.aam.MeanTemplateGaussNewton
                                                  increment () (menpofit.sdm.NonParametricOPPRegression
        method), 47
                                                          method), 262
increment()
                (menpofit.aam.MeanTemplateNewton
                                                 increment () (menpofit.sdm.NonParametricOptimalRegression
        method), 45
```

method), 261		init_i	dentity()	(men-
increment () (menpofit.sdm.NonParametricPCF	RRegress	rion	pofit.transform.DifferentiableAlignmentS	limilarity
method), 259			class method), 429	
increment() (menpofit.sdm.ParametricAppeara	іпсеМеа			(men-
method), 275	14		pofit.transform.DifferentiableSimilarity	class
increment () (menpofit.sdm.ParametricAppeara	іпсемеа			(
<pre>method), 274 increment() (menpofit.sdm.ParametricAppeara</pre>	maaDroi		l_error()	(men-
method), 272			70	
<pre>increment() (menpofit.sdm.ParametricAppeara</pre>	ınceProj	eictOuttNev	<pre>Atomerror() (menpofit.aam.result.AAM. method), 53</pre>	<i>IResult</i>
increment() (menpofit.sdm.ParametricAppeara	ınceWeig	htsGtass		(men-
method), 278			pofit.aps.result.APSAlgorithmResult ma	*
increment () (menpofit.sdm.ParametricAppeara	ınceWeig			
method), 276		initia	l_error() (menpofit.aps.result.APS	SResult
increment () (menpofit.sdm.ParametricShapeGa	aussNew	rton	method), 97	
method), 265		initia	l_error()	(men-
<pre>increment() (menpofit.sdm.ParametricShapeN</pre>	lewton		pofit.lk.result.LucasKanadeAlgorithmRes method), 232	sult
increment() (menpofit.sdm.ParametricShapeO	PPRegre	esisioint.ia		(men-
method), 270	1110670			ethod),
increment() (menpofit.sdm.ParametricShapeOp	ptimalRe			, ,
method), 268			l_error()	(men-
increment () (menpofit.sdm.ParametricShapePC	CRRegre		pofit.result.MultiScaleNonParametricIter	rativeResult
method), 267			method), 376	
increment() (menpofit.sdm.Regularize	dSDM	initia	l_error()	(men-
method), 255			po fit. result. Multi Scale Parametric Iterativ	eResult
increment() (menpofit.sdm.SupervisedDescen	tFitter		method), 393	
method), 251			l_error()	(men-
increment () (menpofit.transform.LinearOrthoM	ADTrans	sform	pofit.result.NonParametricIterativeResul method), 342	t
<pre>method), 414 IncrementalCorrelationFilterThinWr</pre>	anner	initia		(men-
(class in menpofit.clm), 153	apper		e1101 () pofit.result.ParametricIterativeResult m	`
	(men-		359	cinou),
pofit.transform.DifferentiablePiecewiseAj	`	initia	l_error() (menpofit.result.Result m	ethod).
method), 438	9		335	, ,
<pre>init_from_2d_shear()</pre>	(men-	initia	l_error()	(men-
$po {\it fit.trans} for {\it m.Differentiable} Af {\it fine}$	class		$pofit.unified_aam_clm.result.UnifiedAAM$	ACLMAlgorithmResult
method), 419			method), 181	
``	(men-	initia	l_error()	(men-
pofit.trans form. Differentiable Alignment Action for the property of the pr	ffine		pofit.unified_aam_clm.result.UnifiedAAN	1CLMResult
class method), 434			method), 164	
	(men-		l_shape()	(men-
pofit.transform.DifferentiableAlignmentSi class method), 428	imilarity	,	pofit.aam.result.AAMAlgorithmResult erty), 85	prop-
init_from_2d_shear()	(men-	initia	l_shape() (menpofit.aam.result.AAM	I Result
pofit.trans form. Differentiable Similarity	class		property), 68	
method), 424			l_shape()	(men-
<pre>init_identity()</pre>	(men-		pofit.aps.result.APSAlgorithmResult pro	perty),
pofit.transform.DifferentiableAffine	class		129	
method), 419	,	initia	l_shape() (menpofit.aps.result.APS	SResult
init_identity()	(men-		property), 112	
pofit.transform.DifferentiableAlignmentA class method) 434	<i></i> ппе		l_shape() pofit lk result LucasKanadeAloorithmRes	(men-
CIASS METHOD 1 4 14			- ποια τκ τεχμα ΕμιζαχΚανασεΑισονίτην Κε	NIII I

property), 247	is_iterative() (men-
initial_shape() (men-	$po {\it fit.lk.} result. Lucas Kanade Algorithm Result$
pofit.lk.result.LucasKanadeResult property),	property), 247
230	is_iterative() (men-
initial_shape() (men-	pofit.lk.result.LucasKanadeResult property),
$po {\it fit.} result. Multi Scale Non Parametric Iterative Results and the property of the pro$	
property), 391	is_iterative() (men-
initial_shape() (men-	pofit.result.MultiScaleNonParametricIterativeResult
pofit.result.MultiScaleParametricIterativeResult	property), 391
property), 408	is_iterative() (men-
initial_shape() (men-	pofit.result.MultiScaleParametricIterativeResult
pofit.result.NonParametricIterativeResult	property), 408
property), 357	is_iterative() (men-
initial_shape() (men-	pofit.result.NonParametricIterativeResult
pofit.result.ParametricIterativeResult prop-	property), 357
erty), 374	is_iterative() (men-
<pre>initial_shape() (menpofit.result.Result property),</pre>	pofit.result.ParametricIterativeResult property), 374
initial_shape() (men-	<pre>is_iterative() (menpofit.result.Result property),</pre>
$po fit.unified_a am_clm.result. Unified AAMCLMAR and the substitution of the substit$	gorithmResi 41
property), 196	is_iterative() (men-
initial_shape() (men-	$pofit.unified_aam_clm.result. Unified AAM CLM Algorithm Result$
pofit.unified_aam_clm.result.UnifiedAAMCLMRe	
property), 179	is_iterative() (men-
instance() (menpofit.aam.base.AAM method), 15	pofit.unified_aam_clm.result.UnifiedAAMCLMResult
instance() (menpofit.aam.LinearAAM method), 20	property), 179
instance() (menpofit.aam.LinearMaskedAAM	T.
method), 23	J
instance() (menpofit.aam.MaskedAAM method), 18	Jp () (menpofit.transform.OrthoMDTransform method),
instance() (menpofit.aam.PatchAAM method), 25	409
instance() (menpofit.aps.GenerativeAPS method), 89	1
instance() (menpofit.atm.base.ATM method), 132	L
instance() (menpofit.atm.LinearATM method), 136	linear_component() (men-
instance() (menpofit.atm.LinearMaskedATM	pofit.transform.DifferentiableAffine property),
<pre>method), 138 instance() (menpofit.atm.MaskedATM method), 134</pre>	420
instance() (menpojit.atm.PatchATM method), 140	linear_component() (men-
instance() (menpofit.unified_aam_clm.base.UnifiedAA	pofit.transform.DifferentiableAlignmentAffine
method), 157	F F
Inverse (class in menpofit.aps), 94	linear_component() (men-
InverseCompositional (class in menpofit.atm),	pofit.transform.DifferentiableAlignmentSimilarity
144	<pre>property), 430 linear_component()</pre>
InverseCompositional (class in menpofit.lk), 205	pofit.transform.DifferentiableSimilarity prop-
IRLRegression (class in menpofit.math), 319	erty), 425
is_iterative() (men-	LinearAAM (class in menpofit.aam), 18
pofit.aam.result.AAMAlgorithmResult prop-	LinearATM (class in menpofit.atm), 135
erty), 85	LinearMaskedAAM (class in menpofit.aam), 21
is_iterative() (menpofit.aam.result.AAMResult	LinearMaskedATM (class in menpofit.atm), 137
property), 68	LinearOrthoMDTransform (class in men-
is_iterative() (men-	pofit.transform), 412
pofit.aps.result.APSAlgorithmResult property), 130	<pre>load_balanced_frontal_face_fitter() (in</pre>
is_iterative() (menpofit.aps.result.APSResult	module menpofit.aam), 86
property), 112	LucasKanadeAAMFitter (class in menpofit.aam), 26

IncaskanadeAlgorithmResult (class in men-	n_centres() (menpofit.transform.DifferentiableR2LogRRBF			
pofit.lk.result), 231	property), 445			
LucasKanadeATMFitter (class in menpofit.atm), 141	n_dims() (menpofit.modelinstance.GlobalPDM property), 332			
LucasKanadeFitter (class in menpofit.lk), 201 LucasKanadeResult (class in menpofit.lk.result),	_dims() (menpofit.modelinstance.GlobalSimilarityModel property), 327			
214	n_dims() (menpofit.modelinstance.ModelInstance			
217	property), 326			
M	n_dims() (menpofit.modelinstance.OrthoPDM prop-			
mad() (in module menpofit.error), 305	erty), 334			
MaskedAAM (class in menpofit.aam), 16	n_dims() (menpofit.modelinstance.PDM property),			
MaskedATM (class in menpofit.atm), 133	330			
mccf() (in module menpofit.math), 322	n_dims() (menpofit.transform.DifferentiableAffine			
<pre>mean_pupil_49_error() (in module men-</pre>	property), 420			
pofit.error), 307	n_dims() (menpofit.transform.DifferentiableAlignmentAffine			
mean_pupil_68_error() (in module men-	property), 435			
pofit.error), 307	n_dims() (menpofit.transform.DifferentiableAlignmentSimilarity			
MeanTemplateGaussNewton (class in men-	property), 430			
pofit.aam), 47	n_dims() (menpofit.transform.DifferentiablePiecewiseAffine			
MeanTemplateNewton (class in menpofit.aam), 45 MenpoFitBuilderWarning (class in men-	property), 439 n_dims () (menpofit.transform.DifferentiableR2LogR2RBF			
MenpoFitBuilderWarning (class in men- pofit.builder), 292	property), 443			
MenpoFitModelBuilderWarning (class in men-	n_dims() (menpofit.transform.DifferentiableR2LogRRBF			
pofit.builder), 292	property), 445			
ModelInstance (class in menpofit.modelinstance),	n_dims() (menpofit.transform.DifferentiableSimilarity			
325	property), 425			
	n_dims() (menpofit.transform.DifferentiableThinPlateSplines			
(class in menpofit.aam), 34	property), 441			
${\tt ModifiedAlternatingInverseCompositional}$	$n_dims()$ (menpofit.transform.LinearOrthoMDTransform			
(class in menpofit.aam), 35	property), 415			
mosse() (in module menpofit.math), 323	n_dims() (menpofit.transform.OrthoMDTransform			
MultiScaleNonParametricFitter (class in	property), 412			
menpofit.fitter), 310	n_dims_output() (men-			
MultiScaleNonParametricIterativeResult	pofit.transform.DifferentiableAffine property), 420			
(class in menpofit.result), 375 MultiScaleParametricFitter (class in men-	n_dims_output() (men-			
pofit.fitter), 311	pofit.transform.DifferentiableAlignmentAffine			
MultiScaleParametricIterativeResult	property), 435			
(class in menpofit.result), 392	n_dims_output() (men-			
	pofit.transform.DifferentiableAlignmentSimilarity			
N	property), 430			
n_active_components() (men-	n_dims_output() (men-			
pofit.modelinstance.GlobalPDM property),	$po {\it fit.trans} for m. Differentiable Piecewise Affine$			
332	property), 439			
n_active_components() (men-	n_dims_output() (men-			
pofit.modelinstance.OrthoPDM property),	pofit.transform.DifferentiableR2LogR2RBF			
334	property), 443			
n_active_components() (men-	n_dims_output() (men-			
pofit.modelinstance.PDM property), 330	pofit.transform.DifferentiableR2LogRRBF property), 445			
n_active_components() (men-	n_dims_output() (men-			
pofit.transform.LinearOrthoMDTransform property), 415	pofit.transform.DifferentiableSimilarity prop-			
n_centres() (menpofit.transform.DifferentiableR2LogN	425			
property), 443	n_dims_output() (men-			

```
pofit.transform.DifferentiableThinPlateSplines
                                                              pofit.result.MultiScaleParametricIterativeResult
        property), 441
                                                              property), 408
                                                     n iters per scale()
n dims output()
        pofit.transform.LinearOrthoMDTransform
                                                              pofit.unified_aam_clm.result.UnifiedAAMCLMResult
        property), 415
                                                              property), 179
n dims output()
                                                     n landmarks()
                                                                                                    (men-
                                              (men-
        pofit.transform.OrthoMDTransform property),
                                                              pofit.transform.LinearOrthoMDTransform
                                                              property), 415
n_experts() (menpofit.clm.CorrelationFilterExpertEnsemblearameters()
                                                                                                    (men-
                                                              pofit.modelinstance.GlobalPDM
        property), 152
                                                                                                property),
n_global_parameters()
                                              (men-
        pofit.modelinstance.GlobalPDM
                                          property),
                                                     n_parameters()
                                                                                                    (men-
                                                              pofit.modelinstance.GlobalSimilarityModel
n_global_parameters()
                                                              property), 328
                                              (men-
        pofit.modelinstance.OrthoPDM
                                                     n_parameters()
                                                                                                    (men-
                                          property),
        334
                                                              pofit.modelinstance.ModelInstance property),
                                                               326
n_global_parameters()
                                              (men-
                                                     n_parameters()
        pofit.transform.LinearOrthoMDTransform
                                                                                                    (men-
                                                              pofit.modelinstance.OrthoPDM
        property), 415
                                                                                                property),
n iters() (menpofit.aam.result.AAMAlgorithmResult
                                                               334
        property), 85
                                                     n_parameters()
                                                                              (menpofit.modelinstance.PDM
n_iters() (menpofit.aam.result.AAMResult property),
                                                              property), 330
        68
                                                      n parameters()
                                                                                                    (men-
              (menpofit.aps.result.APSAlgorithmResult
                                                              pofit.transform.DifferentiableAffine property),
n iters()
        property), 130
                                                               420
n_iters() (menpofit.aps.result.APSResult property), n_parameters()
                                                                                                    (men-
                                                              po fit. transform. Differentiable A lignment Affine \\
n_iters() (menpofit.lk.result.LucasKanadeAlgorithmResult
                                                              property), 435
        property), 247
                                                      n_parameters()
                                                                                                    (men-
                                                              pofit.transform.DifferentiableAlignmentSimilarity
                (menpofit.lk.result.LucasKanadeResult
n_iters()
        property), 230
                                                              property), 430
n_iters() (menpofit.result.MultiScaleNonParametricIterativeResulteters()
                                                                                                    (men-
                                                              pofit.transform.DifferentiableSimilarity
        property), 391
                                                                                                    prop-
\verb|n_iters()| (menpofit.result.MultiScaleParametricIterativeResult|
                                                              erty), 425
        property), 408
                                                      n parameters()
                                                                                                    (men-
\verb|n_iters()| (menpofit.result.NonParametricIterativeResult|
                                                              pofit.transform.LinearOrthoMDTransform
        property), 357
                                                              property), 415
n_iters() (menpofit.result.ParametricIterativeResult n_parameters()
                                                                                                    (men-
        property), 374
                                                              pofit.transform.OrthoMDTransform property),
n_iters() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgo+ithmResult
                                                      n points()
                                                                       (menpofit.modelinstance.GlobalPDM
        property), 197
n iters() (menpofit.unified aam clm.result.UnifiedAAMCLMResultoperty), 332
        property), 179
                                                     n points() (menpofit.modelinstance.GlobalSimilarityModel
n_iters_per_scale()
                                              (men-
                                                              property), 328
        pofit.aam.result.AAMResult property), 68
                                                     n_points() (menpofit.modelinstance.ModelInstance
n_iters_per_scale()
                                              (men-
                                                              property), 326
        pofit.aps.result.APSResult property), 112
                                                                        (menpofit.modelinstance.OrthoPDM
                                                     n_points()
n_iters_per_scale()
                                              (men-
                                                              property), 334
                                                     n_points() (menpofit.modelinstance.PDM property),
        pofit.lk.result.LucasKanadeResult
                                          property),
                                              (men- n_points() (menpofit.transform.DifferentiableAlignmentAffine
n_iters_per_scale()
        pofit.result.MultiScaleNonParametricIterativeResult
                                                              property), 435
                                                      n points () (menpofit.transform.DifferentiableAlignmentSimilarity
        property), 391
n_iters_per_scale()
                                              (men-
                                                              property), 430
```

```
n points() (menpofit.transform.DifferentiablePiecewiseAffine
                                                           erty), 256
        property), 439
                                                                 (menpofit.sdm.SupervisedDescentFitter
                                                  n scales()
n points() (menpofit.transform.DifferentiableThinPlateSplines
                                                          property), 251
                                                  n_scales() (menpofit.unified_aam_clm.base.UnifiedAAMCLM
        property), 442
n_points() (menpofit.transform.LinearOrthoMDTransform
                                                          property), 157
        property), 415
                                                  n scales() (menpofit.unified aam clm.result.UnifiedAAMCLMResult
n points() (menpofit.transform.OrthoMDTransform
                                                          property), 179
                                                  \verb|n_scales()| (menpofit.unified_aam_clm.UnifiedAAMCLMFitter|
        property), 412
                                                          property), 160
n_sample_offsets()
                                           (men-
        po fit. clm. Correlation Filter Expert Ensemble \\
                                                  n_tris() (menpofit.transform.DifferentiablePiecewiseAffine
        property), 152
                                                          property), 439
n_scales() (menpofit.aam.base.AAM property), 16
                                                                   (menpofit.modelinstance.GlobalPDM
                                                  n_weights()
                                                          property), 332
n_scales() (menpofit.aam.LinearAAM property), 21
n_scales() (menpofit.aam.LinearMaskedAAM prop-
                                                  n_weights()(menpofit.modelinstance.GlobalSimilarityModel
        erty), 23
                                                           property), 328
n_scales()
               (menpofit.aam.LucasKanadeAAMFitter
                                                  n_weights() (menpofit.modelinstance.ModelInstance
        property), 29
                                                          property), 326
n scales() (menpofit.aam.MaskedAAM property), 18
                                                                    (menpofit.modelinstance.OrthoPDM
                                                  n weights()
n_scales() (menpofit.aam.PatchAAM property), 26
                                                          property), 335
n scales() (menpofit.aam.result.AAMResult prop-
                                                  n weights() (menpofit.modelinstance.PDM prop-
        erty), 68
                                                           erty), 330
n_scales() (menpofit.aam.SupervisedDescentAAMFittern_weights() (menpofit.transform.LinearOrthoMDTransform
        property), 32
                                                           property), 415
                (menpofit.aps.GaussNewtonAPSFitter noisy_alignment_similarity_transform()
n scales()
        property), 93
                                                           (in module menpofit.fitter), 314
n_scales() (menpofit.aps.GenerativeAPS property),
                                                  noisy_shape_from_bounding_box() (in mod-
                                                           ule menpofit.fitter), 315
n_scales() (menpofit.aps.result.APSResult property),
                                                  noisy_shape_from_shape() (in module men-
                                                          pofit.fitter), 315
        113
n_scales() (menpofit.atm.base.ATM property), 132
                                                  noisy_target_alignment_transform()
                                                                                                (in
n_scales() (menpofit.atm.LinearATM property), 136
                                                           module menpofit.fitter), 316
n_scales() (menpofit.atm.LinearMaskedATM prop-
                                                  NonParametricGaussNewton (class in
                                                                                              men-
        erty), 138
                                                          pofit.sdm), 258
n_scales()
               (menpofit.atm.LucasKanadeATMFitter
                                                  NonParametricIterativeResult (class in men-
        property), 142
                                                           pofit.result), 341
n_scales() (menpofit.atm.MaskedATM property), 134
                                                  NonParametricNewton (class in menpofit.sdm), 256
n scales () (menpofit.atm.PatchATM property), 140
                                                  NonParametricOPPRegression (class in men-
n_scales() (menpofit.clm.CLM property), 146
                                                           pofit.sdm), 262
n scales()(menpofit.clm.GradientDescentCLMFitter
                                                  NonParametricOptimalRegression (class in
        property), 148
                                                           menpofit.sdm), 260
n scales() (menpofit.dlib.DlibERT property), 200
                                                  NonParametricPCRRegression (class in men-
n scales() (menpofit.fitter.MultiScaleNonParametricFitter
                                                          pofit.sdm), 259
        property), 311
                                                  normalization_wrt_reference_shape()
n_scales()(menpofit.fitter.MultiScaleParametricFitter
                                                           module menpofit.builder), 290
        property), 313
              (menpofit.lk.LucasKanadeFitter prop-
n_scales()
        erty), 203
                                                  OPPRegression (class in menpofit.math), 321
               (menpofit.lk.result.LucasKanadeResult
n_scales()
                                                  OptimalLinearRegression
                                                                                  (class
                                                                                              men-
        property), 230
                                                           pofit.math), 321
n_scales() (menpofit.result.MultiScaleNonParametricIterativeResultransform (class in menpofit.transform),
        property), 391
property), 408
                                                  outer_eye_corner_49_euclidean_error()
n scales() (menpofit.sdm.RegularizedSDM prop-
                                                           (in module menpofit.error), 309
```

outer_eye_corner_51_euclidean_error() (in module menpofit.error), 308	plot_costs() (men- pofit.result.MultiScaleNonParametricIterativeResult
outer_eye_corner_68_euclidean_error()	method), 376
(in module menpofit.error), 308	plot_costs() (men-
P	pofit.result.MultiScaleParametricIterativeResult method), 393
padded_size() (men-	plot_costs() (men-
pofit.clm.CorrelationFilterExpertEnsemble property), 152	pofit.result.NonParametricIterativeResult method), 343
${ t Parametric Appearance Mean Template Guass Ne}$	wplat_costs() (men-
(class in menpofit.sdm), 275	pofit.result.ParametricIterativeResult method),
ParametricAppearanceMeanTemplateNewton	359
(class in menpofit.sdm), 274	plot_costs() (men-
ParametricAppearanceProjectOutGuassNewt	
(class in menpofit.sdm), 272	method), 182
ParametricAppearanceProjectOutNewton	plot_costs() (men-
(class in menpofit.sdm), 271	pofit.unified_aam_clm.result.UnifiedAAMCLMResult method), 164
ParametricAppearanceWeightsGuassNewton	plot_cumulative_error_distribution() (in
(class in menpofit.sdm), 278	module menpofit.visualize), 449
ParametricAppearanceWeightsNewton (class	plot_displacements() (men-
in menpofit.sdm), 276	pofit.aam.result.AAMAlgorithmResult method),
ParametricIterativeResult (class in men- pofit.result), 358	72
poju.resuu),336 ParametricShapeGaussNewton (class in men-	plot_displacements() (men-
pofit.sdm), 265	pofit.aam.result.AAMResult method), 55
ParametricShapeNewton (class in menpofit.sdm),	plot_displacements() (men-
263	pofit.aps.result.APSAlgorithmResult method),
ParametricShapeOPPRegression (class in men-	116
pofit.sdm), 269	plot_displacements() (men-
ParametricShapeOptimalRegression (class in	pofit.aps.result.APSResult method), 99
menpofit.sdm), 268	plot_displacements() (men-
ParametricShapePCRRegression (class in men-	pofit.lk.result.Lucas Kanade Algorithm Result
pofit.sdm), 266	method), 234
PatchAAM (class in menpofit.aam), 23	plot_displacements() (men-
PatchATM (class in menpofit.atm), 139	pofit.lk.result.LucasKanadeResult method),
PCRRegression (class in menpofit.math), 320	216
PDM (class in menpofit.modelinstance), 328	plot_displacements() (men-
PickleWrappedFitter (class in menpofit.io), 316	pofit.result.MultiScaleNonParametricIterativeResult
plot_costs() (men-	method), 378
pofit.aam.result.AAMAlgorithmResult method),	plot_displacements() (men-
70	pofit.result.MultiScaleParametricIterativeResult
plot_costs() (menpofit.aam.result.AAMResult	method), 395
method), 53	plot_displacements() (men- pofit.result.NonParametricIterativeResult
plot_costs() (men-	method), 344
pofit.aps.result.APSAlgorithmResult method), 115	plot_displacements() (men-
	pofit.result.ParametricIterativeResult method),
plot_costs() (menpofit.aps.result.APSResult method), 97	361
plot_costs() (men-	plot_displacements() (men-
pofit.lk.result.LucasKanadeAlgorithmResult	pofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult
method), 232	method), 183
plot_costs() (menpofit.lk.result.LucasKanadeResult	plot_displacements() (men-
method), 215	$pofit.unified_aam_clm.result.Unified AAMCLMResult$
	method), 166

plot_errors() (men-	<pre>project() (menpofit.aam.AppearanceWeightsNewton</pre>
pofit.aam.result.AAMAlgorithmResult method),	method), 42
74	project_out() (men-
plot_errors() (menpofit.aam.result.AAMResult method), 56	pofit.aam.ProjectOutForwardCompositional method), 36
plot_errors() (men-	project_out() (men-
pofit.aps.result.APSAlgorithmResult method),	pofit.aam.ProjectOutGaussNewton method),
118	51
plot_errors() (menpofit.aps.result.APSResult method), 100	<pre>project_out()</pre>
plot_errors() (men-	method), 37
pofit.lk.result.LucasKanadeAlgorithmResult method), 235	<pre>project_out()</pre>
plot_errors() (men-	${\tt ProjectOutForwardCompositional} \ \ \textit{(class in}$
pofit.lk.result.LucasKanadeResult method),	menpofit.aam), 36
218	ProjectOutGaussNewton (class in menpofit.aam),
plot_errors() (men-	50
method), 379	<pre>sultrojectOutInverseCompositional (class in</pre>
	ProjectOutNewton (class in menpofit.aam), 48
method), 396	ProjectOutRegularisedLandmarkMeanShift (class in menpofit.unified_aam_clm), 162
=	pseudoinverse() (men-
<pre>pofit.result.NonParametricIterativeResult method), 346</pre>	pofit.transform.DifferentiableAffine method), 419
plot_errors() (men- pofit.result.ParametricIterativeResult method), 362	pseudoinverse() (men- pofit.transform.DifferentiableAlignmentAffine method), 434
	pseudoinverse() (men-
	gorithmRespdfit.transform.DifferentiableAlignmentSimilarity
method), 185	method), 429 pseudoinverse() (men-
<pre>plot_errors()</pre>	
method), 167	method), 438
predict() (menpofit.math.IIRLRegression method), 320	$po {\it fit.trans} for {\it m.Differentiable Similarity}$
predict() (menpofit.math.IRLRegression method), 319	<pre>method), 424 pseudoinverse()</pre>
predict() (menpofit.math.OPPRegression method), 321	pofit.transform.DifferentiableThinPlateSplines method), 441
<pre>predict() (menpofit.math.OptimalLinearRegression</pre>	pseudoinverse() (men-
method), 321	$po {\it fit.trans} for {\it m.OrthoMDT} rans for {\it m.Or$
<pre>predict() (menpofit.math.PCRRegression method),</pre>	411
320	pseudoinverse_vector() (men-
predict_probability() (men-	pofit.transform.DifferentiableAffine method),
pofit.clm.CorrelationFilterExpertEnsemble	419
method), 152	pseudoinverse_vector() (men-
<pre>predict_response()</pre>	pofit.transform.DifferentiableAlignmentAffine method), 434
method), 152	pseudoinverse_vector() (men-
<pre>print_progress() (in module menpofit.visualize),</pre>	pofit.transform.DifferentiableAlignmentSimilarity method), 429
<pre>project() (menpofit.aam.AppearanceWeightsGaussNew</pre>	pofit.transform.DifferentiableSimilarity (men-

<pre>method), 424 pseudoinverse_vector()</pre>	reconstructed_initial_error() (men- pofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult method), 186
411 R	reconstructed_initial_error() (men- pofit.unified_aam_clm.result.UnifiedAAMCLMResult method), 169
random_instance() (menpofit.aam.base.AAM method), 15	reconstructed_initial_shape() (men- pofit.aam.result.AAMAlgorithmResult prop-
random_instance() (menpofit.aam.LinearAAM method), 21	<pre>erty), 86 reconstructed_initial_shape() (men-</pre>
random_instance() (men- pofit.aam.LinearMaskedAAM method), 23	pofit.aps.result.APSAlgorithmResult property), 130
random_instance() (menpofit.aam.MaskedAAM method), 18	reconstructed_initial_shape() (men- pofit.lk.result.LucasKanadeAlgorithmResult
random_instance() (menpofit.aam.PatchAAM method), 26	<pre>property), 247 reconstructed_initial_shape() (men-</pre>
random_instance() (menpofit.aps.GenerativeAPS method), 89	pofit.result.ParametricIterativeResult property), 374
random_instance() (menpofit.atm.base.ATM method), 132	reconstructed_initial_shape() (men- pofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithmResult
random_instance() (menpofit.atm.LinearATM method), 136 random_instance() (men-	<pre>property), 197 reconstructed_initial_shapes() (men- pofit.aam.result.AAMResult property), 69</pre>
pofit.atm.LinearMaskedATM method), 138 random_instance() (menpofit.atm.MaskedATM	reconstructed_initial_shapes() (men- pofit.aps.result.APSResult property), 113
method), 134 random_instance() (menpofit.atm.PatchATM method), 140	reconstructed_initial_shapes() (men- pofit.lk.result.LucasKanadeResult property), 231
random_instance() (men- pofit.unified_aam_clm.base.UnifiedAAMCLM method), 157	reconstructed_initial_shapes() (men- pofit.result.MultiScaleParametricIterativeResult property), 408
	reconstructed_initial_shapes() (men- pofit.unified_aam_clm.result.UnifiedAAMCLMResult property), 180
reconstructed_initial_error() (men-pofit.aam.result.AAMResult method), 58	reference_shape() (men- pofit.aam.LucasKanadeAAMFitter property),
reconstructed_initial_error() (men- pofit.aps.result.APSAlgorithmResult method), 119	29 reference_shape() (men- pofit.aam.SupervisedDescentAAMFitter
reconstructed_initial_error() (men- pofit.aps.result.APSResult method), 102	property), 32 reference_shape() (men-
reconstructed_initial_error() (men- pofit.lk.result.LucasKanadeAlgorithmResult	pofit.aps.GaussNewtonAPSFitter property), 93
<pre>method), 237 reconstructed_initial_error() (men- pofit.lk.result.LucasKanadeResult method),</pre>	reference_shape() (men- pofit.atm.LucasKanadeATMFitter property), 142
220	reference_shape() (men-
reconstructed_initial_error() (men- pofit.result.MultiScaleParametricIterativeResult	pofit.clm.GradientDescentCLMFitter prop- erty), 148 reference_shape() (menpofit.dlib.DlibERT prop-
<pre>method), 398 reconstructed_initial_error() (men-</pre>	erty), 200
pofit.result.ParametricIterativeResult method), 364	reference_shape() (men- pofit.fitter.MultiScaleNonParametricFitter property), 311

refer	ence_shape()	(men-		method), 38
	pofit.fitter.MultiScaleParametricFitter erty), 313	prop-	run()	(menpofit.aam.SimultaneousInverseCompositional method), 39
refer	ence_shape()	(men-	run()	(menpofit.aam.WibergForwardCompositional
	pofit.lk.LucasKanadeFitter property), 20	`	.,	method), 40
refer	ence_shape()	(men-	run()	(menpofit.aam.WibergInverseCompositional
	pofit.sdm.RegularizedSDM property), 25	56		method), 41
refer	ence_shape()	(men-		(menpofit.aps.Forward method), 95
	pofit.sdm.SupervisedDescentFitter pro 251	operty),	run() run()	(menpofit.aps.Inverse method), 94 (menpofit.atm.ForwardCompositional method),
rafar	ence_shape()	(men-	I ()	143
TOTOL	pofit.unified_aam_clm.UnifiedAAMCLM. property), 160		run()	(menpofit.atm.InverseCompositional method), 144
Regul	arisedLandmarkMeanShift (class	in men-	run()	(menpofit.clm.ActiveShapeModel method), 149
,	pofit.clm), 150		run()	(menpofit.clm.RegularisedLandmarkMeanShift
Regul	arizedSDM (class in menpofit.sdm), 253		- ()	method), 150
	le_images_to_reference_shape		run ()	(menpofit.lk.ForwardAdditive method), 203
	module menpofit.builder), 290	()		(menpofit.lk.ForwardCompositional method), 204
respo	nse_covariance()	(men-		(menpofit.lk.InverseCompositional method), 205
10010	pofit.unified_aam_clm.UnifiedAAMCLM	`		(menpofit.sdm.FullyParametricMeanTemplateNewton
	property), 160	II tite!	I ()	method), 283
Pagul:	t (class in menpofit.result), 335		run ()	(menpofit.sdm.FullyParametricProjectOutGaussNewton
	mean_square_bb_normalised_er:	ror()	Luii ()	method), 282
1000_	(in module menpofit.error), 300	LOL()	run ()	(menpofit.sdm.FullyParametricProjectOutNewton
root	mean_square_distance_indexed	norma		
1000_	(in module menpofit.error), 301	_IIOIIIIa.	run()	(menpofit.sdm.FullyParametricProjectOutOPP
root	mean_square_distance_normali:	ead ar		method), 286
1000_	(in module menpofit.error), 301	seu_eri	run()	(menpofit.sdm.FullyParametricWeightsNewton
root	mean_square_error() (in module	man	Luii ()	method), 285
1001_	pofit.error), 299	men-	mun ()	(menpofit.sdm.NonParametricGaussNewton
		aition al	run()	method), 258
run()	(menpofit.aam.AlternatingForwardCompo	siiionai	()	
()	method), 32	aiti an al	run()	(menpofit.sdm.NonParametricNewton method), 257
run()	(menpofit.aam.AlternatingInverseCompo	sitionai		
()	method), 33	A 7	run()	(menpofit.sdm.NonParametricOPPRegression
run()	(menpofit.aam.AppearanceWeightsGauss	Newton	()	method), 262
()	method), 44	3.7	run()	(menpofit.sdm.NonParametricOptimalRegression
run()	(menpofit.aam.AppearanceWeights	Newton		method), 261
	method), 42		run()	
run()	(menpofit.aam.MeanTemplateGauss	Newton		method), 260
	method), 47		run()	(menpofit.sdm. Parametric Appearance Mean Template Guass Newton
run()	(menpofit.aam.MeanTemplateNewton m	iethod),		method), 276
	46			(menpofit.sdm.ParametricAppearanceMeanTemplateNewton
run()	(menpofit.aam.ModifiedAlternatingForwa	rdCompo		method), 274
	method), 34	_		(menpofit.sdm.ParametricAppearanceProjectOutGuassNewton
run()	(menpo fit. a am. Modified Alternating Inverse	eComposi		method), 273
	method), 35		run()	(menpo fit. sdm. Parametric Appearance Project Out Newton
run()	(menpo fit. a am. Project Out Forward Composition and Composition Forward Compositio	sitional		method), 271
	method), 36		run()	(menpo fit. sdm. Parametric Appearance Weights Guass Newton
run()	$(menpo fit. aam. Project Out Gauss Newton\ m$	iethod),		method), 278
	51		run()	$(menpo {\it fit.sdm.} Parametric Appearance Weights Newton$
run()	(menpofit.aam.ProjectOutInverseCompo	sitional		method), 277
	method), 37		run()	(menpo fit. sdm. Parametric Shape Gauss Newton
	$(menpo fit. aam. Project Out Newton\ method)$			method), 266
run()	(menpofit.aam.SimultaneousForwardCom	positiona	lrun()	(menpofit.sdm.ParametricShapeNewton method),

	264	<pre>set_models_components() (in module men-</pre>
run()	$(menpo {\it fit.sdm.} Parametric Shape OPP Regression$	pofit.checks), 295
	method), 270	<pre>set_target() (menpofit.modelinstance.GlobalPDM</pre>
run()	(menpo fit. sdm. Parametric Shape Optimal Regression	method), 332
	method), 269	set_target() (men-
run()	(menpofit.sdm.ParametricShapePCRRegression method), 267	pofit.modelinstance.GlobalSimilarityModel method), 327
run()	$(menpo fit.unified_aam_clm. Alternating Regular is education and the control of the control of$	Isand <u>m</u> arkMeetnShift (men-
	method), 161	pofit.modelinstance.ModelInstance method),
run()	(menpofit.unified_aam_clm.ProjectOutRegularised)	
	method), 162	set_target() (menpofit.modelinstance.OrthoPDM
S		method), 334
		set_target() (menpofit.modelinstance.PDM
	_images() (in module menpofit.builder), 291	method), 329
scale	1 0	set_target() (men- pofit.transform.DifferentiableAlignmentAffine
-	property), 29	
scale	s() (menpofit.aam.SupervisedDescentAAMFitter	method), 434 set_target() (men-
7	property), 32	pofit.transform.DifferentiableAlignmentSimilarit
scale	s() (menpofit.aps.GaussNewtonAPSFitter prop-	method), 429
7	erty), 93	set_target() (men-
scale	s () (menpofit.atm.LucasKanadeATMFitter prop-	pofit.transform.DifferentiablePiecewiseAffine
7	erty), 143	method), 438
scale		set_target() (men-
7 .	property), 148	pofit.transform.DifferentiableThinPlateSplines
	s () (menpofit.dlib.DlibERT property), 200	T T 1
scare	s () (menpofit.fitter.MultiScaleNonParametricFitter	set_target() (men-
99910	property), 311 s () (menpofit.fitter.MultiScaleParametricFitter	pofit.transform.LinearOrthoMDTransform
scale	property), 313	method), 415
caslo	s() (menpofit.lk.LucasKanadeFitter property),	set_target() (men-
scare	203	pofit.transform.OrthoMDTransform method),
ccalo	s() (menpofit.sdm.RegularizedSDM property),	412
scare	256	<pre>shape_instance() (menpofit.clm.CLM method),</pre>
scale		146
bcarc	property), 252	<pre>shape_instance() (men-</pre>
scale	s () (menpofit.unified_aam_clm.UnifiedAAMCLMF	
Doure	property), 160	method), 157
SDM()	(in module menpofit.sdm), 252	<pre>shape_parameters()</pre>
	h_shape() (men-	pofit.aam.result.AAMAlgorithmResult prop-
00010	pofit.clm.CorrelationFilterExpertEnsemble	erty), 86
	property), 152	shape_parameters() (men-
set. h	_matrix() (men-	pofit.aam.result.AAMResult property), 69
	pofit.transform.DifferentiableAffine method),	<pre>shape_parameters()</pre>
	419	pofit.aps.result.APSAlgorithmResult property),
set h	_matrix() (men-	130
	pofit.transform.DifferentiableAlignmentAffine	<pre>shape_parameters() (men-</pre>
	method), 434	pofit.aps.result.APSResult property), 113
set_h	_matrix() (men-	<pre>shape_parameters() (men-</pre>
	pofit.transform.DifferentiableAlignmentSimilarity	$po {\it fit.lk.} result. Lucas Kanade Algorithm Result$
	method), 429	property), 247
set_h	_matrix() (men-	<pre>shape_parameters() (men-</pre>
	pofit.transform.DifferentiableSimilarity	pofit.lk.result.LucasKanadeResult property),
	method), 424	231
		<pre>shape_parameters() (men-</pre>

shape_parameters()	EC en- en- ED EC
pofit.result.ParametricIterativeResult prop- erty), 374 steepest_descent_images() (menpofit.lk.EC shape_parameters() (men- pofit.unified_aam_clm.result.UnifiedAAMCLMAlgoridumResult_descent_images() (men property), 197 pofit.lk.FourierSSD method), 208 shape_parameters() (men- pofit.unified_aam_clm.result.UnifiedAAMCLMResult pofit.lk.GradientCorrelation method), 213 property), 180 steepest_descent_images() (men shapes() (menpofit.aam.result.AAMAlgorithmResult property), 86 shapes() (menpofit.aam.result.AAMResult property), 69 shapes() (menpofit.aps.result.APSAlgorithmResult property), 130 steepest_descent_update() (menpofit.lk.EC shapes() (menpofit.aps.result.APSResult property), 130 steepest_descent_update() (menpofit.lk.EC shapes() (menpofit.aps.result.APSResult property), 130 steepest_descent_update() (menpofit.lk.FourierSSD method), 209 113 steepest_descent_update() (menpofit.lk.FourierSSD method), 209 steepest_descent_update() (menpofit.lk.FourierSSD method), 209 steepest_descent_update() (menpofit.lk.FourierSSD method), 209 steepest_descent_update() (menpofit.lk.FourierSSD method), 209	EC en- en- ED EC
shape_parameters() (men- pofit.unified_aam_clm.result.UnifiedAAMCLMAlgoridmpResult_descent_images() (menpofit.lk.EC property), 197 pofit.lk.FourierSSD method), 208 shape_parameters() (men- pofit.unified_aam_clm.result.UnifiedAAMCLMResult pofit.lk.GradientCorrelation method), 213 property), 180 steepest_descent_images() (menpofit.aam.result.AAMAlgorithmResult property), 86 shapes() (menpofit.aam.result.AAMAlgorithmResult property), 69 steepest_descent_images() (menpofit.lk.SS shapes() (menpofit.aps.result.APSAlgorithmResult property), 30 steepest_descent_update() (menpofit.lk.EC shapes() (menpofit.aps.result.APSResult property), pofit.lk.FourierSSD method), 209 steepest_descent_update() (menpofit.lk.EC shapes() (menpofit.aps.result.APSResult property), pofit.lk.FourierSSD method), 209 steepest_descent_update() (menpofit.lk.EC steepest_descent_update() (menpo	en- en- SD
shape_parameters()	en- en- SD
pofit.unified_aam_clm.result.UnifiedAAMCLMAlgoridemResult_descent_images() (memorporety), 197 pofit.lk.FourierSSD method), 208 shape_parameters() (men-steepest_descent_images() (memorporety), 180 steepest_descent_images() (memorporety), 180 steepest_descent_images() (memorporety), 86 steepest_descent_images() (memorporety), 86 steepest_descent_images() (memorporety), 86 steepest_descent_images() (memorporety), 69 steepest_descent_update() (memorporety), 86 steepest_descent_update() (memorporety), 130 steepest_	en- en- SD EC
property), 197 shape_parameters() pofit.unified_aam_clm.result.UnifiedAAMCLMResult property), 180 shapes() (menpofit.aam.result.AAMAlgorithmResult property), 86 shapes() (menpofit.aam.result.AAMResult property), 69 shapes() (menpofit.aps.result.APSAlgorithmResult property), 130 shapes() (menpofit.aps.result.APSResult property), shapes() (menpofit.aps.result.APSResult property), property), 130 steepest_descent_update() (menpofit.lk.ECCC)	en- en- SD CC
shape_parameters()	en- SD CC
pofit.unified_aam_clm.result.UnifiedAAMCLMResult pofit.lk.GradientCorrelation method), 213 property), 180 steepest_descent_images() (method), 211 property), 86 pofit.lk.GradientImages method), 211 property), 86 steepest_descent_images() (menpofit.lk.SS shapes() (menpofit.aam.result.AAMResult property), 69 steepest_descent_update() (menpofit.lk.EC shapes() (menpofit.aps.result.APSAlgorithmResult property), 130 steepest_descent_update() (menpofit.lk.EC shapes() (menpofit.aps.result.APSResult property), 210 property), 130 steepest_descent_update() (menpofit.lk.EC steepest_	en- SD CC
property), 180 shapes () (menpofit.aam.result.AAMAlgorithmResult property), 86 shapes () (menpofit.aam.result.AAMResult property), 69 shapes () (menpofit.aps.result.APSAlgorithmResult property), 130 shapes () (menpofit.aps.result.APSResult property), 130 shapes () (menpofit.aps.result.APSResult property), 130 shapes () (menpofit.aps.result.APSResult property), 130 steepest_descent_update () (menpofit.lk.ECS) steepest_desce	SD CC
shapes() (menpofit.aam.result.AAMAlgorithmResult property), 86 shapes() (menpofit.aam.result.AAMResult property), 69 shapes() (menpofit.aps.result.APSAlgorithmResult property), 130 steepest_descent_update() (menpofit.lk.EC steepest_descent_	SD CC
property), 86 shapes() (menpofit.aam.result.AAMResult property), 69 shapes() (menpofit.aps.result.APSAlgorithmResult property), 130 shapes() (menpofit.aps.result.APSResult property), 130 shapes() (menpofit.aps.result.APSResult property), 130 steepest_descent_update() (menpofit.lk.ECsteepest_descent_update() (menpofit.lk.EC	CC
shapes() (menpofit.aam.result.AAMResult property), 69 steepest_descent_update() (menpofit.lk.EC shapes() (menpofit.aps.result.APSAlgorithmResult property), 130 steepest_descent_update() (menpofit.aps.result.APSResult property), 60 pofit.lk.FourierSSD method), 210 steepest_descent_update() (menpofit.aps.result.APSResult property), 60 pofit.lk.FourierSSD method), 209 steepest_descent_update() (menpofit.aps.result.APSResult property), 60 pofit.lk.FourierSSD method), 209 pofit.lk.FourierSSD method), 200 pofit.lk.FourierSSD method)	CC
shapes() (menpofit.aam.result.AAMResult property), 69 steepest_descent_update() (menpofit.lk.EC shapes() (menpofit.aps.result.APSAlgorithmResult property), 130 steepest_descent_update() (menpofit.lk.FourierSSD method), 209 steepest_descent_update() (menpofit.lk.EC steepest_descent_update() (menpofit.lk	CC
shapes() (menpofit.aps.result.APSAlgorithmResult property), 130 steepest_descent_update() (menpofit.lk.EC method), 210 steepest_descent_update() (menpofit.aps.result.APSResult property), pofit.lk.FourierSSD method), 209 steepest_descent_update() (menpofit.lk.EC method), 210 steepest_de	
<pre>shapes() (menpofit.aps.result.APSAlgorithmResult property), 130</pre>	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	'n-
shapes() (menpofit.aps.result.APSResult property), pofit.lk.FourierSSD method), 209 113 steepest_descent_update() (me	
113 steepest_descent_update() (me	
	n-
Shapes () (menpopulik.resun.LucusKunudeAigorumKesun populik.OrdanemCorretation memoa), 213	
property), 248 steepest_descent_update() (me	n-
shapes () (menpofit.lk.result.LucasKanadeResult prop-pofit.lk.GradientImages method), 212	
erty), 231 steepest_descent_update() (menpofit.lk.SS	SD
shapes () (menpofit.result.MultiScaleNonParametricIterativeResult method), 207	
property), 391 SupervisedDescentAAMFitter (class in me	n-
shapes () (menpofit.result.MultiScaleParametricIterativeResult pofit.aam), 29	
property), 409 SupervisedDescentFitter (class in me	n-
shapes () (menpofit.result.NonParametricIterativeResult pofit.sdm), 248	
property), 357	
shapes () (menpofit.result.ParametricIterativeResult T	
property), 374 target() (menpofit.modelinstance.GlobalPDM pro	m_
shapes () (menpofit.unified_aam_clm.result.UnifiedAAMCLMAlgorithm.Result	Ρ
property), 197 target () (menpofit.modelinstance.GlobalSimilarity)	Model
shapes () (menpofit.unified_aam_clm.result.UnifiedAAMCLMResultproperty), 328	Touci
property), 180 target() (menpofit.modelinstance.ModelInstan	CP
similarity_2d_instance_model() (in module property), 326	cc
2 - property), 320	
	n-
menpofit.modelinstance), 326 target() (menpofit.modelinstance.OrthoPDM pro	p-
menpofit.modelinstance), 326 target() (menpofit.modelinstance.OrthoPDM pro SimultaneousForwardCompositional (class in erty), 335	•
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 target() (menpofit.modelinstance.OrthoPDM proefit.modelinstance.PDM propert	•
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in 330 target() (menpofit.modelinstance.OrthoPDM propert 335 erty), 335 target() (menpofit.modelinstance.PDM propert 330	y),
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 target() (menpofit.modelinstance.OrthoPDM propert 330 target() (menpofit.modelinstance.PDM propert 330 target() (menpofit.modelinstance.PDM propert 330) target() (menpofit.transform.DifferentiableAlignment)	y),
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 source() (menpofit.transform.DifferentiableAlignmentAffine menpofit.transform.DifferentiableAlignmentAffine property), 436 target() (menpofit.modelinstance.OrthoPDM property), 335 target() (menpofit.modelinstance.PDM property), 335 target() (menpofit.transform.DifferentiableAlignmentAffine property), 436	y), entAffine
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 source() (menpofit.transform.DifferentiableAlignmentAffine property), 436 property), 436 target() (menpofit.modelinstance.OrthoPDM property), 335 target() (menpofit.modelinstance.PDM property), 330 target() (menpofit.transform.DifferentiableAlignmentAffine property), 436 target() (menpofit.transform.DifferentiableAlignmentAffine property), 436	y), entAffine
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 source() (menpofit.transform.DifferentiableAlignmentAffine property), 436 property), 436 source() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 target() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430	y), ntAffine ntSimilarit
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 source() (menpofit.transform.DifferentiableAlignmentAffine property), 436 property), 436 source() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 property), 430 target() (menpofit.transform.DifferentiablePiecewis	y), ntAffine ntSimilarit
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 Source() (menpofit.transform.DifferentiableAlignmentAffine property), 436 property), 436 source() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 property), 430 source() (menpofit.transform.DifferentiablePiecewiseAffine property), 439 target() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 source() (menpofit.transform.DifferentiablePiecewiseAffine property), 439	y), ntAffine ntSimilarit seAffine
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 Source() (menpofit.transform.DifferentiableAlignmentAffine property), 436 property), 436 source() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 property), 430 source() (menpofit.transform.DifferentiablePiecewiseAffine property), 439 property), 439 target() (menpofit.modelinstance.OrthoPDM property), 335 target() (menpofit.modelinstance.PDM property), 330 target() (menpofit.transform.DifferentiableAlignmentSimilerity property), 436 target() (menpofit.transform.DifferentiablePiecewiseAffine property), 430 target() (menpofit.transform.DifferentiablePiecewiseAffine property), 439 target() (menpofit.transform.DifferentiableThinPlanety), 439	y), ntAffine ntSimilarit seAffine
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 Source() (menpofit.transform.DifferentiableAlignmentAffine property), 436 property), 436 source() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 property), 430 source() (menpofit.transform.DifferentiablePiecewiseAffine property), 439 property), 439 source() (menpofit.transform.DifferentiableThinPlateSplines property), 442	y), ntAffine ntSimilarit seAffine teSplines
simultaneousForwardCompositional (class in menpofit.aam), 38 target () (menpofit.modelinstance.OrthoPDM profit.aam), 38 target () (menpofit.modelinstance.PDM propert 330 target () (menpofit.transform.DifferentiableAlignmentAffine property), 436 target () (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 target () (menpofit.transform.DifferentiablePiecewiseAffine property), 439 target () (menpofit.transform.DifferentiableThinPlateSplines property), 442 target () (menpofit.transform.LinearOrthoMDTransform.	y), ntAffine ntSimilarit seAffine teSplines
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 Source() (menpofit.transform.DifferentiableAlignmentAffine property), 436 property), 436 source() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 source() (menpofit.transform.DifferentiablePiecewiseAffine property), 439 source() (menpofit.transform.DifferentiablePiecewiseAffine property), 439 source() (menpofit.transform.DifferentiableThinPlateSplines property), 442 property), 442 sparse_target() (men-sofit.transform.LinearOrthoMDTransform.Line	y), ntAffine ntSimilarit seAffine teSplines form
SimultaneousForwardCompositional (class in menpofit.aam), 38 target () (menpofit.modelinstance.OrthoPDM profit.aam), 38 target () (menpofit.modelinstance.PDM properts), 335 target () (menpofit.transform.DifferentiableAlignmentAffine property), 436 target () (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 target () (menpofit.transform.DifferentiablePiecewiseAffine property), 439 target () (menpofit.transform.DifferentiablePiecewiseAffine property), 439 target () (menpofit.transform.DifferentiableThinPlateSplines property), 442 target () (menpofit.transform.LinearOrthoMDTransform property), 415 pofit.transform.LinearOrthoMDTransform target () (menpofit.transform.OrthoMDTransform.DifferentiableThinPlateSplines property), 415 target () (menpofit.transform.OrthoMDTransform.DifferentiableThinPlateSplines property), 415 target () (menpofit.transform.OrthoMDTransform.	y), ntAffine ntSimilarit seAffine teSplines form
menpofit.modelinstance), 326 SimultaneousForwardCompositional (class in menpofit.aam), 38 SimultaneousInverseCompositional (class in menpofit.aam), 39 Source() (menpofit.transform.DifferentiableAlignmentAffine property), 436 property), 436 source() (menpofit.transform.DifferentiableAlignmentSimilarity property), 430 source() (menpofit.transform.DifferentiablePiecewiseAffine property), 439 source() (menpofit.transform.DifferentiablePiecewiseAffine property), 439 source() (menpofit.transform.DifferentiableThinPlateSplines property), 442 property), 442 sparse_target() (men-sofit.transform.LinearOrthoMDTransform.Line	y), ntAffine ntSimilarit seAffine teSplines form

```
template() (menpofit.aam.AlternatingInverseCompositional in ()
                                                                       (menpofit.aam.MeanTemplateNewton
        property), 34
                                                              method), 46
template() (menpofit.aam.ModifiedAlternatingForwardCompiositional
                                                                     (menpofit.aam.ProjectOutGaussNewton
                                                              method), 51
        property), 34
template() (menpofit.aam.ModifiedAlternatingInverseCompositionalmenpofit.aam.ProjectOutNewton method), 49
                                                     train() (menpofit.clm.IncrementalCorrelationFilterThinWrapper
        property), 35
template() (menpofit.aam.ProjectOutForwardCompositional
                                                              method), 154
                                                     train() (menpofit.math.IIRLRegression method), 320
        property), 36
template() (menpofit.aam.ProjectOutInverseCompositionalain() (menpofit.math.IRLRegression method), 319
                                                     train() (menpofit.math.OPPRegression method), 322
        property), 37
template() (menpofit.aam.SimultaneousForwardCompostitioniah()
                                                                   (menpofit.math.OptimalLinearRegression
        property), 38
                                                              method), 321
template() (menpofit.aam.SimultaneousInverseCompositionciln() (menpofit.math.PCRRegression method), 320
                                                     train() (menpofit.sdm.FullyParametricMeanTemplateNewton
        property), 39
template() (menpofit.aam.WibergForwardCompositional
                                                              method), 284
        property), 40
                                                     train() (menpofit.sdm.FullyParametricProjectOutGaussNewton
template() (menpofit.aam.WibergInverseCompositional
                                                              method), 282
        property), 41
                                                     train() (menpofit.sdm.FullyParametricProjectOutNewton
template() (menpofit.aps.Forward property), 95
                                                              method), 280
template() (menpofit.aps.Inverse property), 94
                                                     train() (menpofit.sdm.FullyParametricProjectOutOPP
template()
                 (menpofit.atm.ForwardCompositional
                                                              method), 287
                                                     train() (menpofit.sdm.FullyParametricWeightsNewton
        property), 143
                  (menpofit.atm.InverseCompositional
template()
                                                              method), 285
        property), 144
                                                                 (menpofit.sdm.NonParametricGaussNewton
                                                     train()
                                                              method), 259
to_result() (menpofit.aam.result.AAMAlgorithmResult
        method), 75
                                                     train()
                                                                       (menpofit.sdm.NonParametricNewton
to_result()
                      (menpofit.aam.result.AAMResult
                                                              method), 257
                                                     train() (menpofit.sdm.NonParametricOPPRegression
        method), 58
to_result() (menpofit.aps.result.APSAlgorithmResult
                                                              method), 263
        method), 120
                                                     train() (menpofit.sdm.NonParametricOptimalRegression
to_result() (menpofit.aps.result.APSResult method),
                                                              method), 261
         102
                                                     train() (menpofit.sdm.NonParametricPCRRegression
to_result() (menpofit.lk.result.LucasKanadeAlgorithmResult
                                                              method), 260
                                                     train() (menpofit.sdm.ParametricAppearanceMeanTemplateGuassNewton)
        method), 237
to result() (menpofit.lk.result.LucasKanadeResult
                                                              method), 276
                                                     \verb|train()| (menpofit.sdm.ParametricAppearanceMeanTemplateNewton
        method), 220
to result () (menpofit.result.MultiScaleNonParametricIterativeResulthod), 274
        method), 381
                                                     train() (menpofit.sdm.ParametricAppearanceProjectOutGuassNewton
to_result() (menpofit.result.MultiScaleParametricIterativeResultmethod), 273
        method), 398
                                                     train() (menpofit.sdm.ParametricAppearanceProjectOutNewton
to result() (menpofit.result.NonParametricIterativeResult
                                                              method), 272
        method), 347
                                                     train() (menpofit.sdm.ParametricAppearanceWeightsGuassNewton
to result() (menpofit.result.ParametricIterativeResult
                                                              method), 278
                                                     train() (menpofit.sdm.ParametricAppearanceWeightsNewton
        method), 364
to_result() (menpofit.unified_aam_clm.result.UnifiedAAMCLMAnbgathioldin,Result
        method), 187
                                                     train() (menpofit.sdm.ParametricShapeGaussNewton
to_result() (menpofit.unified_aam_clm.result.UnifiedAAMCLMRecattod), 266
                                                                     (menpofit.sdm.ParametricShapeNewton
        method), 169
                                                     train()
train() (menpofit.aam.AppearanceWeightsGaussNewton
                                                              method), 264
                                                     train() (menpofit.sdm.ParametricShapeOPPRegression
        method), 44
             (menpofit.aam.AppearanceWeightsNewton
                                                              method), 270
train()
                                                     train() (menpofit.sdm.ParametricShapeOptimalRegression
                                                              method), 269
            (menpofit.aam.MeanTemplateGaussNewton
train()
        method), 48
                                                     train() (menpofit.sdm.ParametricShapePCRRegression
```

method), 267	U				
transform() (menpofit.aam.AlternatingForwardCompo	sitional: _	d sam alm()			(men-
property), 33	unitte	pofit.unified_aam	clm Unified	AMCIMI	`
transform() (menpofit.aam.AlternatingInverseCompositions)	itional	property), 160	i_cim.Onijieuz	1/1/1/CLM1	·iiiei
property), 34		daamclm	(class	in	men-
transform() (menpofit.aam.AppearanceWeightsGaussN	Newton	pofit.unified_aam	*		men-
property), 45		dAAMCLMAlgor			111.011
transform() (menpofit.aam.AppearanceWeightsNewton	l Onitite	pofit.unified_aam			men-
property), 43		dAAMCLMFitte		_	111.011
transform() (menpofit.aam.MeanTemplateGaussNewto	n	pofit.unified_aam	,	in	men-
property), 48		dAAMCLMResul		in	111.011
transform() (menpofit.aam.MeanTemplateNewton	OHITTE	pofit.unified_aam	`		men-
property), 46		pojii.unijieu_aam	ı_cım.resuii),	103	
transform() (menpofit.aam.ModifiedAlternatingForwar	rd tompos	ritional			
. > 24				Alaanithaa	Dogult
property), 34 transform() (menpofit.aam.ModifiedAlternatingInverse	Composii	tional 76	m.resuu.AAM	Aigorunm	Kesuu
property), 35					
transform() (menpofit.aam.ProjectOutForwardCompos	sitional ((menpofit.aam.res			
property), 36	view()		aps.result.APS	Aigorunm	Kesuu
transform() (menpofit.aam.ProjectOutGaussNewton	()	method), 120	ult A DCD aquilt	mathad) 1	102
	view()	(menpofit.aps.rest	uu.AF SKesuu t I waasVanad	meinoa), 1 aAlaarithu	102 • Degult
<pre>property), 51 transform() (menpofit.aam.ProjectOutInverseComposi</pre>	tional	(menpoju.ik.resui	ı.Lucas K anaa	eAigoriinn	ikesuii
property), 37					
transform() (menpofit.aam.ProjectOutNewton prop-	view()	method), 220	it.lk.result.Luc	as K anaae	Kesuu
	771 OT ()	(mannofit result M	Aulti Saala Non	Daramatri	Itarativa Pagult
erty), 50 transform() (menpofit.aam.SimultaneousForwardComp	positional	method), 381	ини эсшеноп.	<i>i</i> arameirio	HeranveKesun
nuonautu) 20		(menpofit.result.M			
property), 38 transform() (menpofit.aam.SimultaneousInverseCompo	ositional	method), 398	iumscatei ara	inieiricher	шичекезин
property), 39	view()	(menpofit.result.	NonParametr	icItarativa	Pasult
transform() (menpofit.aam.WibergForwardCompositio	nat	method), 348	nom aramen	ichieranve.	Resuit
property), 40	171 OU ()	* *	sult.Parametr	icItarativa	Pasult
property), 40 transform() (menpofit.aam.WibergInverseComposition property) 41	$a\tilde{l}^{\perp \in W()}$	method), 365	suii.1 arameir	ichieranve.	Resuit
property), 41	771 AW ()	(menpofit.result.R	Posult mothod)	336	
transform() (menpofit.aps.Forward property), 95			,		AAMCLMAlgorithmRes
transform() (menpofit.aps.Inverse property), 94	view ()	method), 187	_cant_cint.rest	iii. Oriijica	IIII CEMI II GOT III III CESI
transform() (menpofit.atm.ForwardCompositional	view()	(menpofit.unified_	aam clm resi	ılt Unified.	AAMCLMResult
property), 143	view ()	method), 169	_cant_cint.rest	iii. Oriijica	пинсыни
transform() (menpofit.atm.InverseCompositional	view d	eformation_m	odel()		(men-
property), 144	vicw_a	pofit.aps.Generat			(men
translation_component() (men-	wiew i	terations()	iverii 5 memo	α), σ	(men-
pofit.transform.DifferentiableAffine property),	^ T C W _ T	pofit.aam.result.A	AMAlgorithn	nRosult mo	`
420		81	n inn it got titti	iicsiiii me	mou),
translation_component() (men-	wiew i	terations()			(men-
pofit.trans form. Differentiable Alignment Affine	v10W_1	pofit.aam.result.A	AAMResult me		(men
property), 436	view i	terations()	ii ii/iii ee	,, 05	(men-
translation_component() (men-	v ± 0 ±	pofit.aps.result.A	PSAlgorithmR	Pesult me	`
pofit.trans form. Differentiable Alignment Similarity	,	125	2 211/30 / 1/1/1/12		,,
property), 430	view i	terations() (menpofit.aps.	result.APS	Result
translation_component() (men-		method), 107			
pofit.transform.DifferentiableSimilarity prop-	view i	terations()			(men-
erty), 425		pofit.lk.result.Luc	casKanadeAlo		`
trilist() (menpofit.transform.DifferentiablePiecewiseA	Affine	method), 242			
property), 439	view i	terations()			(men-
	_	pofit.lk.result.Luc	casKanadeRes		thod),

```
225
                                                    weights() (menpofit.modelinstance.PDM property),
view_iterations()
                                            (men-
                                                             330
        pofit.result.MultiScaleNonParametricIterativeResult=ights() (menpofit.transform.LinearOrthoMDTransform
        method), 386
                                                            property), 416
view iterations()
                                            (men-
                                                    WibergForwardCompositional (class in men-
        pofit.result.MultiScaleParametricIterativeResult
                                                            pofit.aam), 40
        method), 403
                                                    WibergInverseCompositional (class in men-
                                            (men-
view iterations()
                                                            pofit.aam), 41
        pofit.result.NonParametricIterativeResult
                                                    with_traceback()
                                                                                                (men-
                                                            pofit.builder.MenpoFitBuilderWarning
        method), 353
                                                            method), 292
view_iterations()
                                            (men-
        pofit.result.ParametricIterativeResult method),
                                                    with_traceback()
                                                                                                (men-
                                                            pofit.builder.MenpoFitModelBuilderWarning
view_iterations()
                                            (men-
                                                            method), 292
        pofit.unified\_aam\_clm.result.UnifiedAAMCLMAlgorithmResult
        method), 192
view_iterations()
                                            (men-
        pofit.unified aam clm.result.UnifiedAAMCLMResult
        method), 174
W
warp_images() (in module menpofit.builder), 291
warped_images()
                                            (men-
        pofit.aam.LucasKanadeAAMFitter
                                          method),
warped_images()
                                            (men-
        pofit.aam.SupervisedDescentAAMFitter
        method), 32
warped_images()
                                            (men-
        pofit.aps.GaussNewtonAPSFitter
                                          method),
        93
warped_images()
                                            (men-
        pofit.atm.LucasKanadeATMFitter
                                          method),
warped_images()
                        (menpofit.lk.ForwardAdditive
        method), 204
warped_images()
                                            (men-
        pofit.lk.ForwardCompositional
                                          method),
        205
warped_images() (menpofit.lk.InverseCompositional
        method), 206
warped_images()
                      (menpofit.lk.LucasKanadeFitter
        method), 203
warped_images()
                                            (men-
        pofit.unified aam clm.UnifiedAAMCLMFitter
        method), 160
weights() (menpofit.modelinstance.GlobalPDM prop-
        erty), 332
weights()(menpofit.modelinstance.GlobalSimilarityModel
        property), 328
               (menpofit.modelinstance.ModelInstance
weights()
        property), 326
weights() (menpofit.modelinstance.OrthoPDM prop-
        erty), 335
```