MenpoFit Documentation

Release 0.7.0

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

Jan 14, 2021

CONTENTS

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.

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:

- `MultiScaleNonParametricFitter` - multi-scale fitting 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:

```
from menpofit.aam import HolisticAAM
from menpo.feature import fast_dsift

aam = HolisticAAM(training_images, group='PTS', reference_shape=None,
                  diagonal=200, scales=(0.25, 0.5, 1.0),
                  holistic_features=fast_dsift, verbose=True)
```

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:

```
from menpofit.aam import LucasKanadeAAMFitter, WibergInverseCompositional

fitter = LucasKanadeAAMFitter(aam,
                              lk_algorithm_cls=WibergInverseCompositional,
                              n_shape=[5, 10, 15], n_appearance=150)
```

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:

```
from menpofit.sdm import SupervisedDescentFitter, NonParametricNewton

fitter = SupervisedDescentFitter(training_images, group='PTS',
                                 sd_algorithm_cls=NonParametricNewton,
                                 verbose=True)
```

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:

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

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.

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.

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.
16. 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.

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

class `menpofit.aam.base.`**`AAM`**(*images*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *transform=<class 'menpofit.transform.piecewiseaffine.DifferentiablePiecewiseAffine'>*, *shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>*, *max_shape_components=None*, *max_appearance_components=None*, *verbose=False*, *batch_size=None*)

Bases: `object`

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

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.

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.

increment(*images*, *group=None*, *shape_forgetting_factor=1.0*, *appearance_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)

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*

HolisticAAM

menpofit.aam.**HolisticAAM**

alias of *AAM*

MaskedAAM

class menpofit.aam.**MaskedAAM**(*images*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *patch_shape=(17, 17)*, *shape_model_cls=<class 'menpofit.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 Masked Active Appearance Model. The appearance of this model is formulated by simply masking an image with a patch-based mask.

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.
- **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.

increment(*images*, *group=None*, *shape_forgetting_factor=1.0*, *appearance_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)

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*

LinearAAM

`class` `menpofit.aam.`**`LinearAAM`**(*images*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *transform=<class 'menpofit.transform.thinsplatesplines.DifferentiableThinPlateSplines'>*, *shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>*, *max_shape_components=None*, *max_appearance_components=None*, *verbose=False*, *batch_size=None*)

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.

- **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.

increment(*images*, *group=None*, *shape_forgetting_factor=1.0*, *appearance_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)
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*

LinearMaskedAAM

class menpofit.aam.**LinearMaskedAAM**(*images, group=None, holistic_features=<function no_op>, reference_shape=None, diagonal=None, scales=(0.5, 1.0), patch_shape=(17, 17), shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>, max_shape_components=None, max_appearance_components=None, verbose=False, batch_size=None*)

Bases: *AAM*

Class for training a multi-scale Linear Masked 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.
- **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.

`increment`(*images*, *group=None*, *shape_forgetting_factor=1.0*, *appearance_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)

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*

PatchAAM

`class` `menpofit.aam.`**`PatchAAM`**(*images*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *patch_shape=(17, 17)*, *patch_normalisation=<function no_op>*, *shape_model_cls=<class 'menpofit.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.

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.
- **`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.

`increment`(*images*, *group=None*, *shape_forgetting_factor=1.0*, *appearance_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)

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, sampling=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*.

Parameters

- **`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.

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.

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 `AAMResult`.

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 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

class menpofit.aam.**SupervisedDescentAAMFitter**(*images*, *aam*, *group=None*, *bounding_box_group_glob=None*, *n_shape=None*, *n_appearance=None*, *sampling=None*, *sd_algorithm_cls=<class 'menpofit.aam.algorithm.sd.ProjectOutNewton'>*, *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 multi-scale cascaded-regression Supervised Descent AAM fitter.

Parameters

- **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*
MeanTemplateGaussNewton		*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.

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.

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

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.

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

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.

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*

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*

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`*

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`*

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`*

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.**AppearanceWeightsNewton**(*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

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

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

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

Bases: `ProjectOut`

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

- **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.

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*

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

class menpofit.aam.result.**AAMResult**(*results*, *scales*, *affine_transforms*, *scale_transforms*, *image=None*, *gt_shape=None*)

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_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_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, render_gt_shape=False, subplots_enabled=True, 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', initial_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, 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_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='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, 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_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='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 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.

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

class `menpofit.aam.result.`**AAMAlgorithmResult**(*shapes*, *shape_parameters*, *appearance_parameters*, *initial_shape=None*, *image=None*, *gt_shape=None*, *costs=None*)

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.

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.
- **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.
- **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.

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*, *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'*, *initial_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*, *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_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='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, 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_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='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 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 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.

GenerativeAPS

class menpofit.aps.**GenerativeAPS**(*images*, *group=None*, *appearance_graph=None*, *shape_graph=None*, *deformation_graph=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *patch_shape=(17, 17)*, *patch_normalisation=<function 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.

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.
- **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*, *colour_map='jet'*, *image_view=True*, *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*, *render_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.

- **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*.

Parameters

- **aps** (`GenerativeAPS` 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

class menpofit.aps.result.**APSResult**(*results*, *scales*, *affine_transforms*, *scale_transforms*, *image=None*, *gt_shape=None*)

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_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_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, render_gt_shape=False, subplots_enabled=True, 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', initial_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, 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_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='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, 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_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='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 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

class `menpofit.aps.result.`**APSAlgorithmResult**(*shapes*, *shape_parameters*, *initial_shape=None*, *image=None*, *gt_shape=None*, *appearance_costs=None*, *deformation_costs=None*, *costs=None*)

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='-'*, *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_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*, *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'*, *initial_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*, *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_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='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, 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_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='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 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 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

class menpofit.atm.base.**ATM**(*template*, *shapes*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *transform=<class 'menpofit.transform.piecewiseaffine.DifferentiablePiecewiseAffine'>*, *shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>*, *max_shape_components=None*, *verbose=False*, *batch_size=None*)

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.

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

increment(*template*, *shapes*, *group=None*, *shape_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)
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

class menpofit.atm.**MaskedATM**(*template*, *shapes*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *patch_shape=(17, 17)*, *max_shape_components=None*, *verbose=False*, *batch_size=None*)

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.

increment(*template*, *shapes*, *group=None*, *shape_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)

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

class menpofit.atm.**LinearATM**(*template*, *shapes*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *transform=<class 'menpofit.transform.thinsplatesplines.DifferentiableThinPlateSplines'>*, *max_shape_components=None*, *verbose=False*, *batch_size=None*)

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.

`increment`(*template*, *shapes*, *group=None*, *shape_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)

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

class menpofit.atm.**LinearMaskedATM**(*template*, *shapes*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *patch_shape=(17, 17)*, *max_shape_components=None*, *verbose=False*, *batch_size=None*)

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.

`increment`(*template*, *shapes*, *group=None*, *shape_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)

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

class menpofit.atm.**PatchATM**(*template*, *shapes*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *patch_shape=(17, 17)*, *patch_normalisation=<function no_op>*, *max_shape_components=None*, *verbose=False*, *batch_size=None*)

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.

`increment`(*template*, *shapes*, *group=None*, *shape_forgetting_factor=1.0*, *verbose=False*, *batch_size=None*)

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.

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 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

class menpofit.clm.**CLM**(*images*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5*, *1)*, *patch_shape=(17*, *17)*, *patch_normalisation=<function no_op>*, *context_shape=(34*, *34)*, *cosine_mask=True*, *sample_offsets=None*, *shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>*, *expert_ensemble_cls=<class 'menpofit.clm.expert.ensemble.CorrelationFilterExpertEnsemble'>*, *max_shape_components=None*, *verbose=False*, *batch_size=None*)

Bases: `object`

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

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.
- **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

class `menpofit.clm.`**`GradientDescentCLMFitter`**(*clm*, *gd_algorithm_cls=<class 'menpofit.clm.algorithm.gd.RegularisedLandmarkMeanShift'>*, *n_shape=None*)

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.

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 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

class menpofit.clm.**ActiveShapeModel**(*expert_ensemble*, *shape_model*, *gaussian_covariance=10*, *eps=1e-05*)

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

class menpofit.clm.**RegularisedLandmarkMeanShift**(*expert_ensemble*, *shape_model*, *kernel_covariance=10*, *eps=1e-05*)

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 'menpofit.clm.expert.base.IncrementalCorrelationFilterThinWra*, *patch_shape=(17, 17)*, *context_shape=(34, 34)*, *response_covariance=3*, *patch_normalisation=functools.partial(<function normalize_norm>, mode='per_channel', error_on_divide_by_zero=False)*, *cosine_mask=True*, *sample_offsets=None*, *prefix=''*, *verbose=False*)

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

- **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

class menpofit.clm.**IncrementalCorrelationFilterThinWrapper**(*cf_callable=<function mccf>*, *icf_callable=<function imccf>*)

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*, *group=None*, *holistic_features=<function no_op>*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *expert_ensemble_cls=<class 'menpofit.clm.expert.ensemble.CorrelationFilterExpertEnsemble'*, *patch_shape=(17, 17)*, *context_shape=(34, 34)*, *sample_offsets=None*, *transform=<class 'menpofit.transform.piecewiseaffine.DifferentiablePiecewiseAffine*, *shape_model_cls=<class 'menpofit.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*, *verbose=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.

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 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.

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.

`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*

Fitter

The model optimised with a combination of Lucas-Kanade and Regularised Landmark Mean Shift algorithms.

UnifiedAAMCLMFitter

class menpofit.unified_aam_clm.**UnifiedAAMCLMFitter**(*unified_aam_clm, algorithm_cls=<class 'menpofit.unified_aam_clm.algorithm.AlternatingRegularisedLa n_shape=None, n_appearance=None, sampling=None*)

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*.

Parameters

- **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 `UnifiedAAMCLMResult` 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.

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 `UnifiedAAMCLMResult`.

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

class menpofit.unified_aam_clm.**AlternatingRegularisedLandmarkMeanShift**(*aam_interface*, *expert_ensemble*, *patch_shape*, *response_covariance*, *eps=1e-05*, ***kwargs*)

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

ProjectOutRegularisedLandmarkMeanShift

class `menpofit.unified_aam_clm.`**`ProjectOutRegularisedLandmarkMeanShift`**(*aam_interface*, *expert_ensemble*, *patch_shape*, *response_covariance*, *eps=1e-05*, ***kwargs*)

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

class `menpofit.unified_aam_clm.result.`**`UnifiedAAMCLMResult`**(*results*, *scales*, *affine_transforms*, *scale_transforms*, *image=None*, *gt_shape=None*)

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.

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_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, render_gt_shape=False, subplots_enabled=True, 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', initial_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, 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_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='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, 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_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='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 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.

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.

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.
- **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.
- **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.

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*, *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'*, *initial_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*, *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_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='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, 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_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='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 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 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*, *group=None*, *bounding_box_group_glob=None*, *reference_shape=None*, *diagonal=None*, *scales=(0.5, 1.0)*, *n_perturbations=30*, *n_dlib_perturbations=1*, *perturb_from_gt_bounding_box=<function noisy_shape_from_bounding_box>*, *n_iterations=10*, *feature_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*)
Bases: *MultiScaleNonParametricFitter*

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.

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.
- **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.

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.lk`

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

class menpofit.lk.**LucasKanadeFitter**(*template*, *group=None*, *holistic_features=<function no_op>*, *diagonal=None*, *transform=<class 'menpofit.transform.homogeneous.DifferentiableAlignmentAffine'>*, *scales=(0.5, 1.0)*, *algorithm_cls=<class 'menpofit.lk.algorithm.InverseCompositional'>*, *residual_cls=<class 'menpofit.lk.residual.SSD'>*)

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.

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 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.

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.

ForwardCompositional

class menpofit.lk.**ForwardCompositional** (*template*, *transform*, *residual*, *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.

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.

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.

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:

$$\mathbf{J}\mathbf{J}^{\mathbf{T}}$$

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:

$$\sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x;p))]$$

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.

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:

$$\mathbf{J}\mathbf{J}^{\mathbf{T}}$$

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:

$$\sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x;p))]$$

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)

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:

$$\sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x;p))]$$

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:

$$\mathbf{J}\mathbf{J}^{\mathbf{T}}$$

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:

$$\sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x;p))]$$

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:

$$\sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x;p))]$$

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.

Fitting Result

LucasKanadeResult

class menpofit.lk.result.**LucasKanadeResult**(*results*, *scales*, *affine_transforms*, *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_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_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*, *render_gt_shape=False*, *subplots_enabled=True*, *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'*, *initial_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*, *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_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='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, 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_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='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 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

class `menpofit.lk.result.`**LucasKanadeAlgorithmResult**(*shapes*, *homogeneous_parameters*, *initial_shape=None*, *image=None*, *gt_shape=None*, *costs=None*)

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, 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_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, 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', initial_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, 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_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='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, 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_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='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 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

class `menpofit.sdm.`**`SupervisedDescentFitter`**(*images*, *group=None*, *bounding_box_group_glob=None*, *sd_algorithm_cls=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*)

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*
ParametricShapeOptimalRegression	*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.

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

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*.

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.
- **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.

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.
- **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.

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.

`increment`(*images*, *group=None*, *bounding_box_group_glob=None*, *verbose=False*, *batch_size=None*)

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*

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*).

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.

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

class menpofit.sdm.**NonParametricGaussNewton**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *compute_error=<function euclidean_bb_normalised_error>*, *alpha=0*, *bias=True*, *alpha2=0*)

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.

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.

NonParametricPCRRegression

class `menpofit.sdm.`**`NonParametricPCRRegression`**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *compute_error=<function euclidean_bb_normalised_error>*, *variance=None*, *bias=True*)

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.

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.

NonParametricOptimalRegression

class menpofit.sdm.**NonParametricOptimalRegression**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *compute_error=<function euclidean_bb_normalised_error>*, *variance=None*, *bias=True*)

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.

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.

NonParametricOPPRegression

class `menpofit.sdm.`**`NonParametricOPPRegression`**(*patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, compute_error=<function euclidean_bb_normalised_error>, bias=True*)

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

class menpofit.sdm.**ParametricShapeNewton**(*patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True*)

Bases: `ParametricShapeSDAlgorithm`

Class for training a cascaded-regression algorithm that employs a parametric shape model using Incremental Regularized Linear Regression (`IRLRegression`).

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.

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.

ParametricShapeGaussNewton

class `menpofit.sdm.`**ParametricShapeGaussNewton**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>*, *compute_error=<function euclidean_bb_normalised_error>*, *alpha=0*, *bias=True*, *alpha2=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.

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.

ParametricShapePCRRegression

class menpofit.sdm.**ParametricShapePCRRegression** (*patch_features=<function 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 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).
- **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.

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.

ParametricShapeOptimalRegression

class menpofit.sdm.**ParametricShapeOptimalRegression**(*patch_features=<function 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.

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.

ParametricShapeOPPRegression

class menpofit.sdm.**ParametricShapeOPPRegression**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *shape_model_cls=<class 'menpofit.modelinstance.OrthoPDM'>*, *compute_error=<function euclidean_bb_normalised_error>*, *whiten=False*, *bias=True*)

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`).

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.
- **`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.

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

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

class menpofit.sdm.**ParametricAppearanceProjectOutNewton**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>*, *compute_error=<function euclidean_bb_normalised_error>*, *alpha=0*, *bias=True*)

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.

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.

ParametricAppearanceProjectOutGuassNewton

class menpofit.sdm.**ParametricAppearanceProjectOutGuassNewton**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>*, *compute_error=<function euclidean_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 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

class menpofit.sdm.**ParametricAppearanceMeanTemplateNewton**(*patch_features=<function no_op>, patch_shape=(17, 17), n_iterations=3, appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>, compute_error=<function euclidean_bb_normalised_error>, alpha=0, bias=True*)

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.

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.

ParametricAppearanceMeanTemplateGuassNewton

class menpofit.sdm.**ParametricAppearanceMeanTemplateGuassNewton**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>*, *compute_error=<function euclidean_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.

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.

ParametricAppearanceWeightsNewton

class menpofit.sdm.**ParametricAppearanceWeightsNewton** (*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>*, *compute_error=<function euclidean_bb_normalised_error>*, *alpha=0*, *bias=True*)

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.

ParametricAppearanceWeightsGuassNewton

class menpofit.sdm.**ParametricAppearanceWeightsGuassNewton**(*patch_features=<function no_op>*, *patch_shape=(17, 17)*, *n_iterations=3*, *appearance_model_cls=<class 'menpo.model.pca.PCAVectorModel'>*, *compute_error=<function euclidean_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

class menpofit.sdm.**FullyParametricProjectOutNewton**(*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*)

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.
- **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.

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.

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.

FullyParametricMeanTemplateNewton

class menpofit.sdm.**FullyParametricMeanTemplateNewton** (*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*)

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.

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.

FullyParametricWeightsNewton

class menpofit.sdm.**FullyParametricWeightsNewton**(*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*)

Bases: `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.

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.

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.

FullyParametricProjectOutOPP

class `menpofit.sdm.`**`FullyParametricProjectOutOPP`**(*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>, bias=True*)

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*, *verbose=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

`menpofit.builder.`**`rescale_images_to_reference_shape`**(*images*, *group*, *reference_shape*, *verbose=False*)

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=''*, *verbose=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 <= float <= 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

d_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.

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

`menpofit.error.`**`root_mean_square_bb_normalised_error`**(*shape*, *gt_shape*, *norm_shape=None*, *norm_type='avg_edge_length'*)

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 *norm_shape*'s bounding box
bb_perimeter	Perimeter of *norm_shape*'s bounding box
bb_avg_edge_length	Average edge length of *norm_shape*'s bbox
bb_diagonal	Diagonal of *norm_shape*'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

`menpofit.error.`**`root_mean_square_distance_indexed_normalised_error`**`(`*shape*, *gt_shape*, *index1*, *index2*`)`

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

`menpofit.error.`**`euclidean_bb_normalised_error`**(*shape*, *gt_shape*, *norm_shape=None*, *norm_type='avg_edge_length'*)

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 *norm_shape*'s bounding box
bb_perimeter	Perimeter of *norm_shape*'s bounding box
bb_avg_edge_length	Average edge length of *norm_shape*'s bbox
bb_diagonal	Diagonal of *norm_shape*'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

`menpofit.error.`**`euclidean_distance_indexed_normalised_error`**(*shape*, *gt_shape*, *index1*, *index2*)

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_avg_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

class `menpofit.fitter.`**`MultiScaleNonParametricFitter`**(*scales*, *reference_shape*, *holistic_features*, *algorithms*)

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.

`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*

MultiScaleParametricFitter

class menpofit.fitter.**MultiScaleParametricFitter**(*scales*, *reference_shape*, *holistic_features*, *algorithms*)

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 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

- **`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*, *alignment_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*, *target*, *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

`menpofit.fitter.`**`noisy_target_alignment_transform`**(*source*, *target*, *alignment_transform_cls=<class 'menpo.transform.homogeneous.affine.AlignmentAffine'>*, *noise_std=0.1*, ***kwargs*)

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

`class` `menpofit.io.`**`PickleWrappedFitter`**(*fitter_cls*, *fitter_args*, *fitter_kwargs*, *fit_from_bb_kwargs*, *fit_from_shape_kwargs*, *image_preprocess=<function image_greyscale_crop_preprocess>*)

Bases: `object`

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

```
from menpofit.io import PickleWrappedFitter, image_greyscale_crop_preprocess
from functools import partial

# LucasKanadeAAMFitter only takes one argument, a trained aam.
fitter_args = (aam, )

# kwargs for fitter construction. Note that here sampling is a
# list of numpy arrays we have already constructed (one per level)
fitter_kwargs = dict(lk_algorithm_cls=WibergInverseCompositional,
                     sampling=sampling)

# kwargs for fitter.fit_from_{bb, shape}
# (note here we reuse the same kwargs twice)
fit_kwargs = dict(max_iters=[25, 5])

# Partial over the PickleWrappedFitter to prepare an object that can be
# invoked at load time
```

(continues on next page)

(continued from previous page)

```
fitter_wrapper = partial(PickleWrappedFitter, LucasKanadeAAMFitter,
                         fitter_args, fitter_kwargs,
                         fit_kwargs, fit_kwargs,
                         image_preprocess=image_greyscale_crop_preprocess)

# save the pickle down.
mio.export_pickle(fitter_wrapper, 'pretrained_aam.pkl')

# ----------------------- L O A D  T I M E ---------------------------#

# at load time, invoke the partial to instantiate this class (and build
# the internally-held wrapped fitter)
fitter = mio.import_pickle('pretrained_aam.pkl')()
```

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`** (*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 **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.
- **l** (*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.
- **l** (*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.
- **l** (*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.
- **l** (*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

d_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,) 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*

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*

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, new_figure=False, render_image=True, render_final_shape=True, render_initial_shape=False, render_gt_shape=False, subplots_enabled=True, 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', initial_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, 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_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='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='-'*, *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_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.

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*, *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'*, *initial_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*, *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_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='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*, *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_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='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 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

class menpofit.result.**ParametricIterativeResult**(*shapes*, *shape_parameters*, *initial_shape=None*, *image=None*, *gt_shape=None*, *costs=None*)

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_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.

`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, 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', initial_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, 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_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='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, 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_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='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 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

class `menpofit.result.`**`MultiScaleNonParametricIterativeResult`**(*results*, *scales*, *affine_transforms*, *scale_transforms*, *image=None*, *gt_shape=None*)

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_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.

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*, *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'*, *initial_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*, *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_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='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, 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_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='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 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

class `menpofit.result.`**`MultiScaleParametricIterativeResult`**(*results*, *scales*, *affine_transforms*, *scale_transforms*, *image=None*, *gt_shape=None*)

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_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_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, render_gt_shape=False, subplots_enabled=True, 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', initial_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, 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_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='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, 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_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='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 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.
- **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.
- **`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

DifferentiableAffine

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 *`composes_with`* 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

d_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 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.

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.

Type `(n_dims, n_dims)` *ndarray*

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 *composes_with* 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

d_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.

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.

Type `(n_dims, n_dims)` *ndarray*

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 *composes_with* 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 **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

d_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.

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.

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.

Type `(n_dims, n_dims)` *ndarray*

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

```
[(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 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 *composes_with* 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 **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

d_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 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 explicit 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.

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.

Type `(n_dims, n_dims)` *ndarray*

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.
- **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 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

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

class `menpofit.transform.`**DifferentiableThinPlateSplines**(*source*, *target*, *kernel=None*)

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.

dW_dl = dOmega_dl * k(points) = T * d_L**-1_dl * k(points) = T * -L**-1 dL_dl L**-1 * k(points)

# 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, d) (c, d) = ( ) (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_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.

`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.
auc	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.
auc	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:

```
import numpy as np
from menpofit.visualize import statistics_table

method_names = ['Method_1', 'Method_2', 'Method_3']
errors = [list(np.random.normal(0.07, 0.02, 400)),
          list(np.random.normal(0.06, 0.03, 400)),
          list(np.random.normal(0.08, 0.04, 400))]
```

We can create a pandas *DataFrame* as:

```
tab = statistics_table(errors, method_names, auc_max_error=0.1,
                       auc_error_step=0.001, sort_by='auc')
tab
```

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, title='Cumulative Error Distribution', x_label='Normalized Point-to-Point Error', y_label='Images Proportion', 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', legend_font_style='normal', legend_font_size=10, legend_font_weight='normal', legend_marker_scale=1.0, legend_location=2, legend_bbox_to_anchor=(1.05, 1.0), legend_border_axes_pad=1.0, legend_n_columns=1, legend_horizontal_spacing=1.0, legend_vertical_spacing=1.0, legend_border=True, legend_border_padding=0.5, legend_shadow=False, legend_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