Active Appearance Model

Practical Assignments for Advanced Digital Image Processing Course

LKEB - LUMC 2006

Chapter 1

Model Building

In this assignment, we are going to create an appearance model from the myocardial contour shapes, taken from the short-axis MR images. This requires the following steps:

1. Building a statistical shape model, i.e.:

$$\boldsymbol{x} \simeq \bar{\boldsymbol{x}} + \boldsymbol{\Phi}_{\mathrm{s}} \boldsymbol{b}_{\mathrm{s}} \tag{1.1}$$

   where $\boldsymbol{x}$, $\bar{\boldsymbol{x}}$, $\boldsymbol{\Phi}_{\mathrm{s}}$ and $\boldsymbol{b}_{\mathrm{s}}$ are the training shape, the mean shape, the principal components and the shape parameters, respectively. The convention used in this assignment for a shape vector with N landmark points is as follows:

$$\boldsymbol{x} = [x_1, y_1, x_2, y_2, \ldots, x_N, y_N]^T \tag{1.2}$$

2. Building a statistical grey-level/texture/intensity model, i.e.:

$$\boldsymbol{g} \simeq \bar{\boldsymbol{g}} + \boldsymbol{\Phi}_{\mathrm{g}} \boldsymbol{b}_{\mathrm{g}} \tag{1.3}$$

   where $\boldsymbol{g}$, $\bar{\boldsymbol{g}}$, $\boldsymbol{\Phi}_{\mathrm{g}}$ and $\boldsymbol{b}_{\mathrm{g}}$ are the texture training vector, the mean texture vector, the principal components and the texture parameters, respectively.

3. Building an appearance model, i.e.:

$$\boldsymbol{b} \simeq \boldsymbol{\Phi}_{\mathrm{c}} \boldsymbol{c} \tag{1.4}$$

   The vector $\boldsymbol{b}$ is the combined shape parameters $\boldsymbol{b}_{\mathrm{s}}$ and texture parameters $\boldsymbol{b}_{\mathrm{g}}$. Note that since both shape and texture parameters have zero mean, so does $\boldsymbol{c}$.

A detailed and complete explanation about AAM is given in the Cootes' report: `http://www.isbe.man.ac.uk/ bim/Models/app_models.pdf`

1.1 Shape Modeling

The training shapes are endocardial and epicardial contours from short-axes MRI images. They have been drawn manually and saved in the MATLAB file: `shortaxes.mat`. Fig. 1.1 shows one of training shapes overlaid on top of its MRI image.

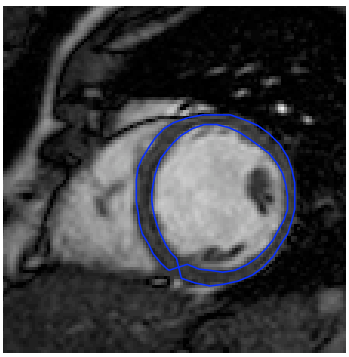


Figure 1.1: An example of training shape plotted with its source image.

Exercise 1.1

Go to the practical assignment directory and perform the following steps to startup path and load the training shapes:

```
>> startup;
>> S = importdata('shortaxes.mat');
```

How many landmark points and how many shapes are there? Use `plot_shapes` function to plot the shapes (see Appendix A.4).

Exercise 1.2 Shape Alignment

After we have loaded the training shape set, the next step is to align all the shapes, such that all shapes are invariant under translation, scaling and rotation. Use `align_all_shapes` function (see Appendix A.1) to align the training shapes.

1. Plot the aligned shapes and compare it with the unaligned shapes.
2. Plot the mean shape.
3. Explain how the point correspondence is guaranteed in this specific training shapes.

4. Experiment the shape alignment with different initial mean shape option of the `align_all_shapes` function. Can you see the difference if you take different shape as initial mean shape?

5. In the shape alignment, projection of points onto their tangential line may help to reduce the non-linearity of the point distribution. Can you see the difference with and without tangential projection in our training shapes? (*Hint: use the* `'tangential_projection'` *option.*)

Exercise 1.3 Shape Modeling with PCA

Compute the principal component analysis of our aligned training shapes (see Appendix A.2). How many principal components do we need to retain 98% of total variance? Reconstruct a shape from the PCA model with the 98% total variance explained. Plot it together with the original shape. (*Hint: use Equation 1.1*) Can you see the difference between the two?

Do the same step with 90%, 85% and 80% dimensionality reducton. For each reduction, notice how many dimensions that are retained.

Only for interested reader. *The quality of the model to describe its training shape can be given by a reconstruction error analysis. You can calculate the reconstruction error using the root mean square error between each landmark point.*

Exercise 1.4 Mode of Variation

The function `shape_viewer` can be used to inspect the mode of variation. Call the function as follows (let `shape_pca` be the PCA model variables):

```
>> shape_viewer(shape_pca);
```

Click and drag the mouse inside the axis to see the shape generated by the PCA model by changing the parameter vector $\boldsymbol{b}$ on two selected principal components. As default the x-axis and y-axis represent the first and the second principal components. You can set any combination of 2 principal components you want to mix from the Model Settings menu.

As you click-and-drag the mouse further the color becomes red, showing that the shape is not statistically plausible with respect to the model. As a default it uses $\pm 3\sqrt{\sigma_i}$, where σ is the eigen value of the i-th principal component. You can set this parameter in the menu.

What do the first 3 mode of variations describe by the model?

1.2 Texture Modeling

In the texture (or intensity or grey-level) modeling, we are going to perform the warping technique, intensity sampling and photometric normalization. After that we can perform the same PCA approach on the normalized texture vectors and play with its mode of variation.

Exercise 1.5 Warping

To get point correspondence for the texture model, we need to use a reference image and compute the relationship between points in the reference image and in the shape image. The relationship is described as a warping transformation function. We are going to use the thin plate splines method [**?**] to do the warping transformation.

The example on how to create mask is given in Appendix A.3. Follow the similar steps as in Appendix A.3 using the default option and the mean shape, that we have generated from the previous exercise. Plot the mask together with the mean shape.

The warping transformation is described in Appendix A.6. Select one particular training shape, e.g. warp the mask into the 5th training shape. Plot the warped mask together with the training shape.

The grid points in the target image becomes irregular. During the sampling, we have to make rounding to the nearest image indices.

We can actually make the mask from the training shape, perform sampling there, where we still have regular grid points, and then warp the mask into the mean shape. Can you explain why this is not a good practice?

Exercise 1.6 Intensity sampling

For each training shape, we perform the intensity sampling, to get the texture vectors. Load the list of training shape filenames:

```
>> fnames = importdata('mri_sa.txt');
```

Iterate through the list of filenames to perform the intensity sampling.

```
>> G = [];
>> for i=1:length(fnames)
P = thin_plate_splines(shape_pca.mean, S(:,i), mask);
G(:,i) = intensity_sampling(fnames{i},P);
disp(sprintf('Sampling from %s',fnames{i})); end
```

To plot a texture vector, use function `plot_texture` (see Appendix A.5).

Exercise 1.7 Texture normalization

Perform the normalization of texture vectors (`photometric_normalization` function) and save it into variable `Gn`. Select one texture vector, e.g. the 10th texture vector from `G` and `Gn`, and plot both texture vectors in different figures. Can you see the difference?

Exercise 1.8 PCA model of texture

With the similar way of the PCA of shape model, we can perform the PCA onto the texture vectors. Use also 98% dimensionality reduction. Notice how many texture vector dimension you get after PCA. Plot the mean texture vector.

Reconstruct one texture vector from the PCA model and compare it with the original texture vector. The mode of texture variation can also be visualized using the `texture_viewer` function, with the same functionality as with the `shape_viewer` function. Can you explain what do the first 2 mode of texture variations represent?

1.3 Combined Appeareance Modeling

The shape and texture parameters can be combined to build the appearance model. Hence we are going to use PCA again to create the combined appearance model (see again Eq. 1.1, 1.3 and 1.4).

In the combined appearance model, the observed data is the combined shape ($\boldsymbol{b}_\mathrm{s}$) and texture ($\boldsymbol{b}_\mathrm{g}$) parameters.

$$\boldsymbol{b} = \begin{pmatrix} \boldsymbol{W}_\mathrm{s}\boldsymbol{b}_\mathrm{s} \\ \boldsymbol{b}_\mathrm{g} \end{pmatrix} \tag{1.5}$$

where $\boldsymbol{W}_\mathrm{s}$ is a weighting matrix for the shape parameters to make the shape and texture parameters comparable. Usually in practice the choice of $\boldsymbol{W}_\mathrm{s}$ is relatively insensitive. A simple approach is by taking $\boldsymbol{W}_\mathrm{s} = r\boldsymbol{I}$, where r^2 is the ratio of the total intensity variation to the total shape variation in the normalized frames and $\boldsymbol{I}$ is just an identity matrix.

Thus PCA on the combined appearance model is taken on $\boldsymbol{b}$ and since $\boldsymbol{b}$ is normalized, then the model is given as in Eq. 1.4.

Exercise 1.9

Reformulate the statistical shape model in Eq. 1.1 and the texture model in Eq. 1.3 that takes into account the appearance parameter $\boldsymbol{c}$ (see Eq. 1.4). Hint: fill this following blank forms as a help.

Let $\boldsymbol{\Phi}_\mathrm{c} = (\boldsymbol{\Phi}_\mathrm{cs}\boldsymbol{\Phi}_\mathrm{cg})^T$, which means that the appearance components $\boldsymbol{\Phi}_\mathrm{c}$ consists of components from the shape $\boldsymbol{\Phi}_\mathrm{cs}$ on top and components from the texture $\boldsymbol{\Phi}_\mathrm{cg}$ at the bottom. From Eq. 1.5, $\boldsymbol{\Phi}_\mathrm{cs}$ and $\boldsymbol{\Phi}_\mathrm{cg}$, we can derive:

$\boldsymbol{b}_{\mathrm{s}} =$ _____ $\boldsymbol{c}$

$\boldsymbol{b}_{\mathrm{g}} =$ _____ $\boldsymbol{c}$

Hence

$\boldsymbol{x} = \bar{\boldsymbol{x}} + \boldsymbol{\Phi}_{\mathrm{s}}\boldsymbol{b}_{\mathrm{s}} = \bar{\boldsymbol{x}} +$ _____

$\boldsymbol{g} = \bar{\boldsymbol{g}} + \boldsymbol{\Phi}_{\mathrm{g}}\boldsymbol{b}_{\mathrm{g}} = \bar{\boldsymbol{g}} +$ _____

Exercise 1.10

Calculate $\boldsymbol{W}_{\mathrm{s}}$ and compute PCA of $\boldsymbol{b}$ but without the mean removal. Plot the mean appearance with `plot_appearance` function. Reconstruct one of the training shapes from this appearance model using the equation that you have just derived from the previous exercise. Plot both the reconstructed shape from the appearance model and the original shape in the same axes. Compare the two shapes. Can you see the difference?

For you who are interested further, you can analyze how well the combined appearance model describes the shape and texture variation by calculating the reconstruction error both in shape and texture.

Exercise 1.11

Use the `appearance_viewer` function to play with its mode of variations. Can you see the difference between the appearance model and the texture model? Play with different combination of principal components to understand the mode of variations, that are given by the appearance model.

Before you leave, save the workspace to a file. The shape, texture and appearance model are going to be used during the matching in the next chapter.

Chapter 2

AAM Matching

In this chapter, we are going to use the appearance model that we have built in the previous chapter to match a new unseen image. The goal in AAM matching is to minimize the difference between the texture generated by the model ($\boldsymbol{I}_m$) with the texture from the new image ($\boldsymbol{I}_i$), which is located under the shape generated by the model.

$$\delta \boldsymbol{I} = \boldsymbol{I}_i - \boldsymbol{I}_m \tag{2.1}$$

Minimizing $|\delta_{\boldsymbol{I}}|^2$ is an optimization problem and we are going to find the minimum value by using a 'guidance' that is learned from the model.

2.1 Learning to Correct Pose and Model Parameters

To update model and pose parameters during the matching process, the AAM model needs to be trained, such that it can predict the correct way towards the solution. We used a priori knowledge, i.e. training shapes, that are perturbed its pose ($\delta \boldsymbol{t}$) and/or its model ($\delta \boldsymbol{c}$) parameters, and calculate the difference with the original generated texture

$$\delta \boldsymbol{g} = \boldsymbol{g}_{\mathrm{s}} - \boldsymbol{g}_{\mathrm{m}} \tag{2.2}$$

where $\boldsymbol{g}_{\mathrm{s}}$ and $\boldsymbol{g}_{\mathrm{m}}$ are the texture vector from the displaced and the original generated from the model.

The linear relationship between $\delta \boldsymbol{t}$ and $\delta \boldsymbol{c}$ are given as follows

$$\delta \boldsymbol{t} = R_t \, \delta \boldsymbol{g} \tag{2.3}$$

$$\delta \boldsymbol{c} = R_c \, \delta \boldsymbol{g} \tag{2.4}$$

The AAM model is thus trained to get the matrix R_t and R_c.

Exercise 2.1 Preparation

If you saved your workspace during the previous assignment, then you can load them back. Don't forget to call the `startup` script to set the path. If you didn't save the last assignment workspace, then perform steps described in Appendix B to prepare the appearance model.

Exercise 2.2 Learning pose parameter prediction

You can use the `pose_prediction` function to learn the pose parameter prediction during the matching. The pose parameter with position (t_x,t_y), scaling s and angle (θ) is defined as follows

$$\boldsymbol{t} = [s_x \ s_y \ t_x \ t_y]^T \tag{2.5}$$

where $s_x = s \ \cos(\theta) - 1$ and $s_y = s \ \sin(\theta)$.

Read Appendix A.7 on how to give arguments to call the `pose_prediction` function. Use only 5 training shapes to learn the pose parameters, otherwise it will take a long time to compute. Look at the variable `Rt`. The rows are associated with rows in Eq. 2.5.

Plot the texture (using `plot_texture`) of each row of `Rt`. Can you see the resemblance with gradient images, especially for the third (t_x) and the last (t_y) rows?

Exercise 2.3 Learning model parameter prediction

The `params_prediction` function is used to learn the appearance model prediction (R_c). It is similar way to call this function with `pose_prediction`. Since this function makes a regression on all appearance parameters and each parameter, by default, takes 4 times regressions (see Appendix A.8), then it will take longer time to compute R_c, compared with R_t. To visualize the R_c matrix, try to compute with the first 3 training shapes. Look at the variable `Rc`. Each row is associated with each mode in the appearance model (the $\boldsymbol{b}$ vector in Eq. 1.3).

Plot the first and the second row of `Rc`, with the `plot_texture` function, which represent the model prediction if we change the first and the second mode of variations of the appearance model. If you don't remember how these first two mode of variations then call again the `appearance_viewer` to visualize these modes.

Exercise 2.4 Learning pose and model parameter prediction from all training shapes

Fortunately, this approach is performed only once during the training. Thus

we can pre-compute pose and model parameter predictions before the AAM matching.

Load file `Rt50.mat` and `Rc50.mat` from the current directory and these two files contain the pose and model parameter predictions from all 50 training shapes. Try to look at the textures of these two variables and compare it with the previous exercises.

> **Important note:** These two variables (`Rt50.mat` and `Rc50.mat`) depend on many parameters you used during the model building. If you use **exactly** equal steps as described in Appendix B, then you can use these variables. Otherwise you need to train the pose and model parameter yourself. In this case, use around 20 training shapes for this training to reduce the computation time.

2.2 Iterative Model Optimization

One basic AAM search is called the iterative model optimization [**?**]. We are going to implement this basic search in the following exercises. Before that, make sure that you already have these following *ingredients* for the AAM matching process:

Iterative model optimization algorithm:

1: Initialize damping vector $k = [1.5, 0.5, 0.25, 0.125, 0.0625]^T$
2: Initialize pose parameter ($\boldsymbol{P}_0$) and model parameter $\boldsymbol{c} = \boldsymbol{0}$.
3: Initialize shape generated model $\boldsymbol{x}_{\mathrm{m}}$.
4: $\boldsymbol{x}_{\mathrm{match}} \leftarrow$ transformed shape of $\boldsymbol{x}_{\mathrm{m}}$ with $\boldsymbol{P}_0$.
5: **while** not converge **do**
6: $\quad \boldsymbol{g}_{\mathrm{m}} \leftarrow$ generated texture model with parameter $\boldsymbol{c}$.
7: $\quad \boldsymbol{g}_{\mathrm{im}} \leftarrow$ texture that are sampled from the image using $\boldsymbol{x}_{\mathrm{match}}$.
8: $\quad$ Normalize $\boldsymbol{g}_{\mathrm{im}}$ into $\boldsymbol{g}_s$.
9: $\quad$ Evaluate error vector $\delta \boldsymbol{g}_0 = \boldsymbol{g}_{\mathrm{s}} - \boldsymbol{g}_{\mathrm{m}}$.
10: $\quad$ Evaluate $E_0 = |\delta \boldsymbol{g}_0|$.
11: $\quad$ Predict pose displacement $\delta \boldsymbol{t}$.
12: $\quad$ Predict model parameter displacement $\delta \boldsymbol{c}$.
13: $\quad$ **for all** $k_i \in k$ **do**
14: $\quad\quad$ Update $\boldsymbol{c} = \boldsymbol{c} - k_i \delta \boldsymbol{c}$.
15: $\quad\quad$ Invert transform $\boldsymbol{x}_{\mathrm{match}}$ and transform it back by taking into account $\delta \boldsymbol{t}$.
16: $\quad\quad$ $\boldsymbol{g}_{\mathrm{m}} \leftarrow$ generated texture model with parameter $\boldsymbol{c}$.
17: $\quad\quad$ $\boldsymbol{g}_{\mathrm{im}} \leftarrow$ texture that are sampled from the image using $\boldsymbol{x}_{\mathrm{match}}$.
18: $\quad\quad$ Normalize $\boldsymbol{g}_{\mathrm{im}}$ into $\boldsymbol{g}_i$.
19: $\quad\quad$ Evaluate error vector $\delta \boldsymbol{g}_i = \boldsymbol{g}_{\mathrm{i}} - \boldsymbol{g}_{\mathrm{m}}$.
20: $\quad\quad$ Evaluate $E_i = |\delta \boldsymbol{g}_i|$.
21: $\quad\quad$ **if** $E_i \leq E_0$ **then**

22: Break from this loop.
23: **end if**
24: **end for**
25: $E = E_i$
26: Define convergence if E does not change from the previous iteration.
27: **end while**

Exercise 2.5

The iterative model optimization is implemented in the script `aam_matching.m`. Since the script will run on global workspace, you need to replace all variable names to match with your own variable names. Otherwise the script will raise an error message. Try to understand the script by comparing it with the above algorithm, while you replace variable names.

There are 5 images that have been prepared for testing AAM search. They are listed in the `mri_sa_tests.txt` files and their contours in `shortaxes_test.mat` file. This script will ask which test image you want to use.

Note that at the beginning, this script will call the `initial_pose` function to define the initial pose parameter. Click and drag the mouse to define t_x, t_y, scale and angle θ. After you are satisfied with the initial position, click the `get` button to continue with the AAM search.

At the end of the AAM search, the matched shape is plotted as green lines, together with the image's correct contour (yellow). A simple point-to-point average error is also calculated and displayed in the main window.

Exercise 2.6

Play with different initialization position. What do you think about the affect of initial pose parameter with the AAM search result? What are other kind of drawbacks from this AAM basic search that you can think of?

Appendix A

Predefined Functions

There are predefined functions available throughout this practical assignment in the current directory. In the following sections, each of the predefined function is explained briefly with some examples. Not all predefined functions are explained in this appendix. Some of them are clearly explained using the `help` function. For example,

```
>> help intensity_sampling
```

A.1 `align_all_shapes`

Description:
Perform the Procrustes shape alignment on a training set.
Usage:

```
Sa = align_all_shapes(S);
Sa = align_all_shapes(S,'arg1',val1,'arg2',val2,...);
```

Input:

1. `S` is a $2N \times M$ shape matrix, where N is the number of landmark points and M is the number of shapes. Each shape vector $\boldsymbol{v}$ (column in `S`) must have the following format:

$$\boldsymbol{v} = [x_1, y_1, x_2, y_2, \ldots, x_N, y_N]^T$$

Output:

1. `Sa` is the aligned $2N \times M$ shape matrix.

Optional arguments:

1. `'init_mean', 0 <= integer <= M`. Default is `1`.
   This value determines which shape of `S` that is used for the initial mean shape. The minimum number is 0 (randomly assigned) and the maximum number is the number of shapes, i.e. M. Note that this value determines the final pose of the aligned shapes.

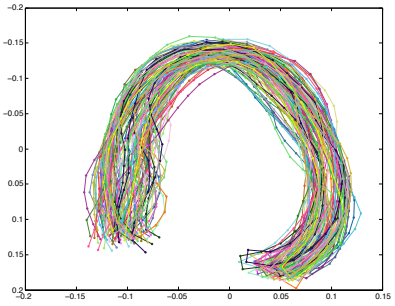
2. `'tangent_projection', 0 | 1`. Default is `1`.
   If the value is `1`, then the landmark points after alignment in each step are projected to its tangent space. Otherwise no projection into the tangent space. Projection into the tangent space might be needed to make the distribution of the aligned shapes is linear.

3. `'limit', numeric`. Default is `1e-5`. This value determines the limit value for the stopping criteria, i.e. convergence.

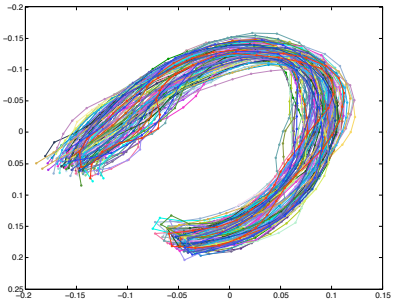
Examples:

The variable `S` in this example is the same shape set of contours from 4 chamber view MRI that is used in Appendix A.4. The following are examples how to use the `align_all_shapes` function.

```
>> Sa = align_all_shapes(S);
>> plot_shapes(Sa,'close',0,'unique_color',1);
```



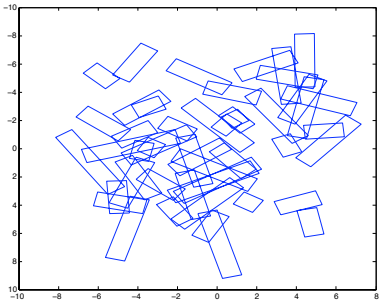
```
>> Sb = align_all_shapes(S,'init_mean',0);
>> plot_shapes(Sb,'close',0,'unique_color',1);
```



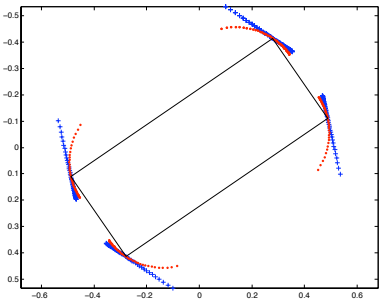
Notice how the final pose is different.

In the next example, `S` consists of rectangles that their aspect ratio are varied. The next example shows the different of shape distribution after the alignment with and without the projection to the tangent space.

```
>> S = gen_rectangles(50);
>> plot_shapes(S,'Marker','none');
```



```
>> Sa = align_all_shapes(S);
>> Sb = align_all_shapes(S,'tangent_projection',0);
>> Sm = mean(Sa,2);
>> plot_shapes(Sa,'+b');
>> hold on;
>> plot_shapes(Sb,'.r');
>> plot_shapes(Sm,'-k');
```



A.2 `compute_pca`

Description:

Perform the Principal Component Analysis. The PCA linear generative model is given as follows

$$\boldsymbol{x} = \bar{\boldsymbol{x}} + \boldsymbol{\Phi}\boldsymbol{b}$$

where $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$ are the observed data and the mean vector of the aligned training set. The $\boldsymbol{\Phi}$ is a matrix where each column is a principal component and $\boldsymbol{b}$ is the weighting parameter vector.

Usage:

```
pca = compute_pca(S);
pca = compute_pca(S,'arg1',val1,'arg2',val2,...);
```

Input:

1. `S` is the aligned training set matrix. Each training sample is a vector (column in `S`).

Output:

1. `pca` is a structure with the following fields:
   - `pca.phi` is the $\mathbf{\Phi}$ matrix,
   - `pca.b` is a matrix of $\boldsymbol{b}$ vectors,
   - `pca.var` is variances for each principal component, and
   - `pca.mean` is the mean vector.

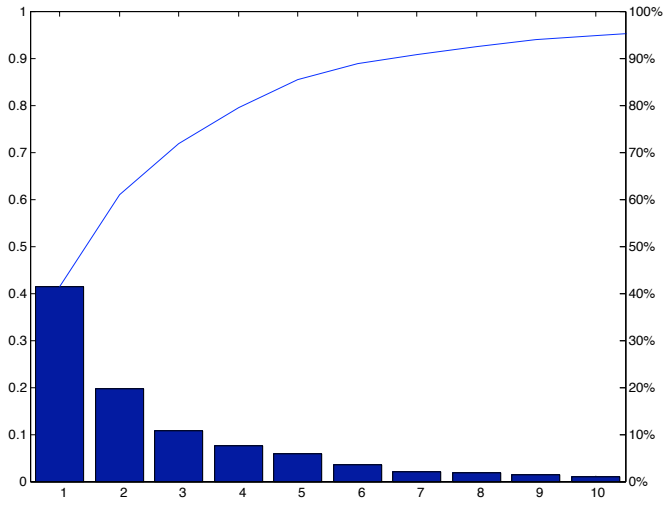
Optional arguments:

1. `'remove_mean', 0 | 1`. Default is `1`.
   This value determines whether removing mean from the input `S` is performed before calculating PCA. If the value is 1, then the mean removal is performed. Otherwise `S` is used as is.

2. `'reduce', 0<numeric<=1`. Default is `1`.
   PCA can be used to reduce the dimension of the data. This option determines how to reduce the dimensionality of `S`. If the value is 1, then all principal components are used. If this option is set, for example, to 0.98, then the dimension is reduced such that the model captures 98% of the total variance.

Examples: Let `S` be a set of 2D planar shapes, taken from 4 chambers view of cardiac MR images.

```
>> S = importdata('4chambers.mat')
>> Sa = align_all_shapes(S);
>> pca = compute_pca(Sa)
pca =
      phi: [128x128 double]
        b: [128x56 double]
      var: [128x1 double]
   mean: [128x1 double]
>> pareto(pca.var/sum(pca.var));
```



The above picture shows the variance explained in the cumulative plot for each component, starting from the first principal component.

A.3 `create_mask`

Description:

Create 2D grid points ($N \times 2$ matrix) that represents a mask, created from a shape vector.

Usage:

```
pts = create_mask(x);
pts = create_mask(x,opt1,val1,opt2,val2,...);
```

Input:

1. `x` is a valid shape vector.

Output:

1. `pts` is $N \times 2$ grid points.

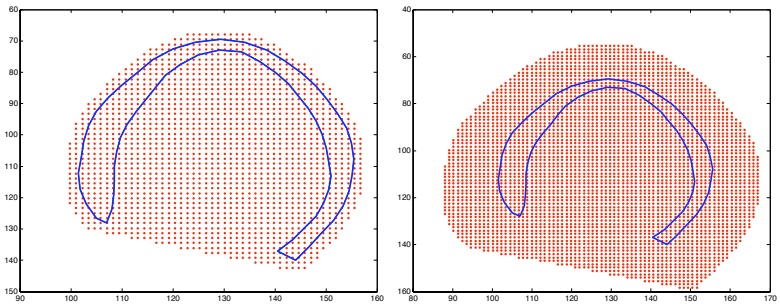
Optional arguments:

1. `'extent', number`. Default is `1.1`.
   This value determines how much extent of the mask from the outermost contour of the shape. Sometimes it is necessary to expand the mask some few pixels to make sure that all pixels are covered by the mask.

2. `'density', array of 2 elements`. Default is `[50 50]`.
   This value determines how many sampling points per dimension.

Examples: Let variable `x` be a shape vector and make two masks, one with the default arguments and the other with 1.5 extent and 75 by 75 grid points.

```
>> mask1 = create_mask(x);
>> plot(mask1(:,1),mask1(:,2),'r.');
>> hold on;
>> plot_shapes(x,'-b');
>> mask2 = create_mask(x,'extent',1.5,'density',[75,75]);
>> figure;
>> plot(mask1(:,1),mask1(:,2),'r.');
>> hold on;
>> plot_shapes(x,'-b');
```



A.4 plot_shapes

Description:

Plot 2D shape matrix as the first argument into the current figure.

Usage:

```
h = plot_shapes(S);
h = plot_shapes(S,LineSpec,'arg1',val1,'arg2',val2,...);
h = plot_shapes(S,'arg1',val1,'arg2',val2,...);
```

Input:

1. `S` is a $2N \times M$ shape matrix, where N is the number of landmark points and M is the number of shapes. Each shape vector $\boldsymbol{v}$ (column in `S`) must have the following format:

$$\boldsymbol{v} = [x_1, y_1, x_2, y_2, \ldots, x_N, y_N]^T$$

Output:

1. `h` is an array of graphic object handles. Each element points to each shape in the figure.

Optional arguments:

1. `LineSpec` is similar with `LineSpec` in the built-in Matlab's `plot` command. The default style for each shape is `'-b.'`.

2. Any valid optional arguments for the built-in Matlab's `plot` command.

3. `'img_format', 0 | 1`. Default is `1`.
   If the value is `1`, then image coordinate system is used, i.e. origin is at the top left corner. Otherwise it uses the default coordinate system.

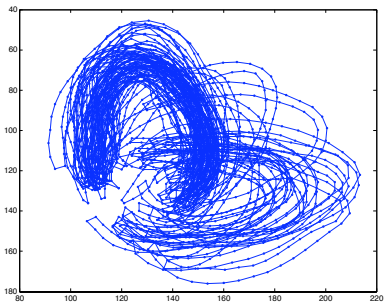
4. `'close', 0 | 1`. Default is `1`.
   If the value is `1`, then the shape is drawn as a closed curve. Otherwise it is an open curve.

5. `'unique_color', 0 | 1`. Default is `0`. If the value is `1`, then each shape uses randomly assigned different color.

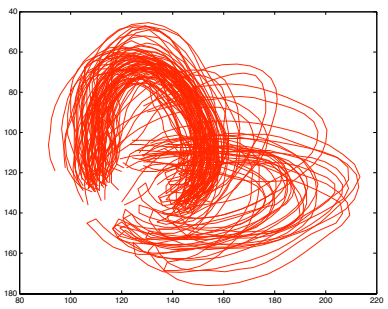
Examples:

The variable `S` in this example is a shape set myocardial borders from MRI images in four chamber view. The following are examples how to use the `plot_shapes` function.

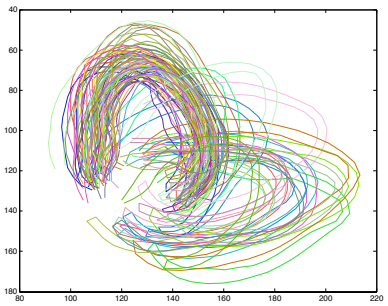
```
>> S = importdata('4chambers.mat');
>> plot_shapes(S);
```




```
>> plot_shapes(S,'-r','close',0);
```



```
>> plot_shapes(S,'close',0,'Marker','none','unique_color',1);
```



A.5 plot_texture

Description:
Plot a texture (or grey-level or intensity) vector.
Usage:

```
h = plot_texture(patch,g);
```

Input:

1. `patch` is a $N \times 2$ matrix that describes a 2D patch, which gives location of each vertices where a texture vector is drawn onto. (*see* `create_mask` *function to create a patch based on a shape vector*)

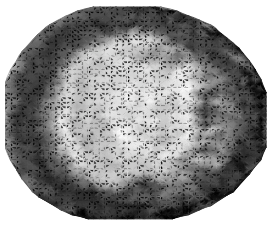
2. `g` is an N length of texture vector.

Output:

1. `h` is an array of object handle to the patch.

Examples: Let `xmean` and `g` be a mean shape and a texture vector.

```
>> mask = create_mask(xmean);
>> plot_texture(mask,g);
```



A.6 thin_plate_spline

Description:

Perform 2D warping using the thin plate splines described in [**?**].

Usage:

```
Pt = thin_plate_spline(xs,xt,Ps);
```

Input:

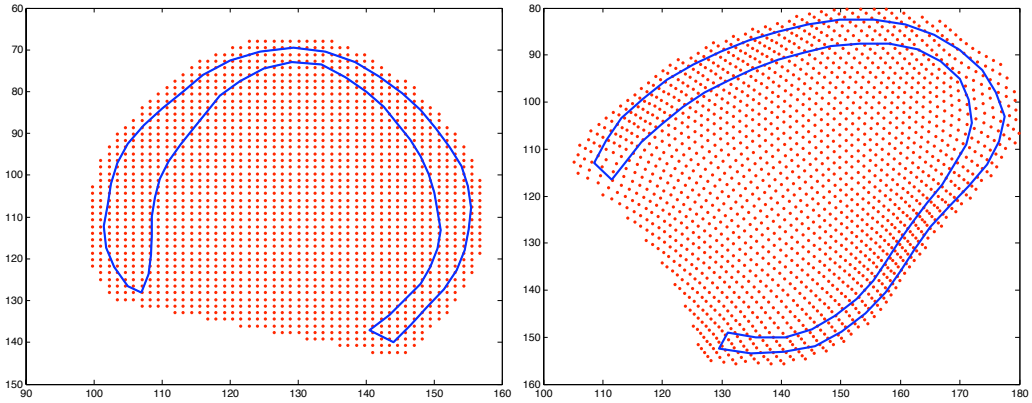
1. `xs` is a shape vector acts as source control points.
2. `xt` is a shape vector acts as target control points.
3. `Ps` is a $N \times 2$ points that are going to be warped.

Output:

1. `Pt` is $N \times 2$ warped points.

Examples: Let variable `xs` and `mask` are source shape vector and mask that are created in Appendix A.3. We warp `mask` by using another shape `xt` as shown here:

```
>> mask = create_mask(xs);
>> Pt = thin_plate_spline(xs,xt,mask);
>> plot(mask(:,1),mask(:,2),'r.'); hold on;
>> plot_shapes(xs,'-b');
>> figure;
>> plot(Pt(:,1),Pt(:,2),'r.'); hold on;
>> plot_shapes(xt,'-b');
```



A.7 pose_prediction

Description:

Performs the Fixed Jacobian Matrix Estimation to build the pose parameter predictors that are needed during the AAM matching.

Usage:

```
Rt = pose_prediction(SM,TM,AM,Ws,files,shapes,mask);
Rt = pose_prediction(SM,TM,AM,Ws,files,shapes,mask,opt1,val1,...);
```

Input:

1. SM is the shape PCA model,
2. TM is the texture PCA model,
3. AM is the appearance PCA model,
4. Ws is the weighting matrix used to scale shape parameters in the appearance modeling (see Eq. 1.4),
5. files is a cell string contains training shapes' files,
6. mask is a 2D grid points.

Output:

1. `Rt` is $4 \times M$ matrix of pose parameter predictions, where M is the number of elements in the texture vector.

Optional arguments:

1. `'x_disp', array`. Default is `[-6 -3 -1 1 3 6]`.
   This value determines the displacement in the x direction.
2. `'y_disp', array`. Default is `[-6 -3 -1 1 3 6]`.
   This value determines the displacement in the y direction.
3. `'scale_disp', array`. Default is `[0.95 0.97 0.99 1.01 1.03 1.05]`.
   This value determines the displacement of scale.
4. `'angle_disp', array`. Default is `[-5 -3 -1 1 3 5]`.
   This value determines the displacement of angle (rotation) in degrees.
5. `'select', indices`. Default is to use all training shapes.
   Select which training shapes to include during the prediction training.

Examples: Learn pose prediction from the first 5 training shapes:

```
>> Rt = pose_prediction(shape_m,texture_m,app_m,Ws,fnames,...
            S,mask,'select',1:5);
```

(*note that the triple dots are used for line breaking only*)

A.8 params_prediction

Description:
Performs the Fixed Jacobian Matrix Estimation to build the appearance parameter predictors that are needed during the AAM matching.
Usage:

```
Rc = params_prediction(SM,TM,AM,Ws,files,shapes,mask);
Rc = params_prediction(SM,TM,AM,Ws,files,shapes,mask,opt1,val1,...);
```

Input:

1. SM is the shape PCA model,
2. TM is the texture PCA model,
3. AM is the appearance PCA model,
4. Ws is the weighting matrix used to scale shape parameters in the appearance modeling (see Eq. 1.4),
5. files is a cell string contains training shapes' files,
6. mask is a 2D grid points.

Output:

1. `Rc` is $N \times M$ matrix of appearance parameter predictions, where N is the number of appearance parameters (modes) in the appearance model and M is the number of elements in the texture vector.

Optional arguments:

1. `'model_disp', array`. Default is `[-0.5 -0.25 0.25 0.5]` times variance.
   This value determines the displacement of each of the appearance parameter.
2. `'select', indices`. Default is to all training shapes.
   Select which training shapes to include during the prediction training.

Example: Learn appearance parameter prediction from the first 5 training shapes:

```
>> Rc = params_prediction(shape_m,texture_m,app_m,Ws,fnames,...
            S,mask,'select',1:3);
```

(*note that the triple dots are used for line breaking only*)

Appendix B

Preparing Appearance Model

1. Call the `startup` script.
2. Load the short-axis MR shapes: `S=importdata('shortaxes.mat');`
3. Shape alignment: `Sa = align_all_shapes(S);`
4. Compute PCA shape model:
   `shape_pca = compute_pca(Sa,'reduce',0.98);`
5. Load the list of short-axis MRI filenames:
   `fnames = importdata('mri_sa.txt');`
6. Intensity sampling:
   `[G,mask] = generate_texture_samples(shape_pca.mean,S,fnames);`
7. Texture normalization: `Gn = photometric_normalization(G);`
8. Compute PCA texture model:
   `texture_pca = compute_pca(Gn,'reduce',0.98);`
9. Calculate weighting matrix W_s:
   `Ws = sqrt(sum(texture_pca.var) / sum(shape_pca.var));`
10. Compute PCA appearance model:
    ```
    app_pca = compute_pca([Ws*shape_pca.b; texture_pca.b],...
    'remove_mean',0, 'reduce',0.98);
    ```