THE USE OF MULTI RESOLUTION ACTIVE SHAPE MODELS FOR FACE DETECTION

KONSTANTINOS AMPLIANITIS

February 2012

Technische Universität Berlin
Faculty of Planning, Construction and Environment
Institute for Geodesy and Geoinformation Science

THE USE OF MULTI RESOLUTION ACTIVE SHAPE MODELS FOR FACE DETECTION

MASTER THESIS

Submitted to the Institute for Geodesy and Geoinformation Science
at the University of Technology Berlin
in partial fulfillment of the requirements
for the Degree of

Master of Science

by

KONSTANTINOS AMPLIANITIS
[Matriculation Number: 329232]

February 2012

Supervisor: Prof. Dr - Ing, Olaf Hellwich
Department of Computer Vision & Remote Sensing

Technische Universität Berlin
Faculty of Planning, Construction and Environment
Institute for Geodesy and Geoinformation Science

DECLARATION OF AUTHORSHIP

I hereby declare that the work presented here is original and the result of my own independent investigations, except where otherwise acknowledged. All content and ideas drawn directly or indirectly from external sources are cited as such. This master's thesis has not been submitted, either in part or whole, to any other university, institution or examining body and has not been published.

February 2012 Konstantinos Amplianitis

Berlin, Germany

ACKNOWLEDGMENTS

Thanks are due first to my supervisor, Professor Olaf Hellwich for his cooperation, perspectives, advice and guidance whenever possible.

I am also greatly thankful to my Professor in Photogrammetry, Elli Petsa from the Technological Educational Institution of Athens, who encouraged and supported me to pursue my studies in Germany, particularly at Berlin Institute of Technology.

Finally, my sincerest gratitudes and appreciation go to my parents for their faith and unlimited support to me.

Contents

ABSTRACT

This master thesis examines the use of a multi resolution Active Shape Model (ASM) applied on facial features, utilizing the Viola/Jones face detector.

The method, initially introduced by Cootes, et. al, requires good initial pose parameter values for placing a face model from its local system to the image's system. This is one of the most critical parts of the process from which the convergence of the method depends on. For this reason, the Viola/Jones detector kicked in, to initially detect the face and subsequently estimate the initial pose parameters for positioning the face model in the search image. The testing of the face detector as well as the quality of the model's initial position was executed on face images provided by the Milborrow University of Cape Town (MUCT) online database.

For building a face model, a set of training images provided by Cootes was used and the search images were chosen randomly from the same training set.

Experiments made initially on some frontal upright images, showed that the face detector succeeded in all images and the placement of the face model was quite accurate in most cases. Subsequently, the quality of the model fit using the multi resolution active shape model approach, showed that the method converged quite well for the inner part of the face but in the outer part, in some cases, was not that precise.

1

Introduction

This thesis will describe in detail the implementation of the Active Shape Model method, running in a multi resolution approach, on gray scale images, with the initial pose parameters of the face model estimated by the Viola/Jones face detector. Generally, good mathematical skills are required (especially linear algebra and statistics) and additionally some fundamental knowledge in digital image processing. The reader should not limit himself to the mathematical material presented within this work but use it as a reference for concurring more solid knowledge in the near future.

1.1 Motivation

According to the method, when an instance of an object model in a local normalised system is given, initial pose parameter values are requested for placing the model into the image and begin the searching/convergence process. Nevertheless, given some approximate values for translation, rotation and scaling, the process might or might not converge. This means that the method itself is very sensitive and dependent from the given initial values. Having faces as the object; two things are examined within this thesis work:

- Expanding the method to a multi resolution approach. This could improve the efficiency and robustness of the existing algorithm. It starts by searching for face features in a coarse image and when convergence at the current level it's rescaled and placed as an initial position on the next image level. This leads to a faster algorithm and the probability of failing to detect the correct facial features, is much less.

- Estimating the initial pose parameter values needed for the placement of a face model in the image, using the Viola/Jones algorithm.

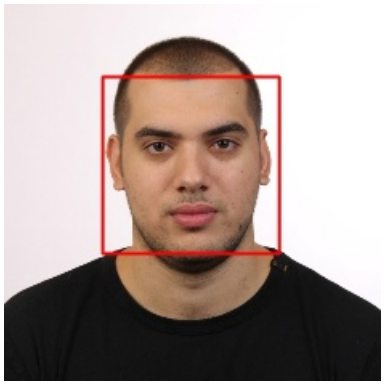


Figure 1.1: *Face detector*

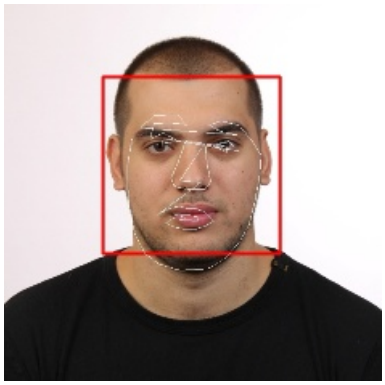


Figure 1.2: *Initial position of the face model*

There are several issues which are not investigated in this thesis work:

1. Developing a system that automatically measures corresponding landmark points from a set of training images. This can be considered as an autonomous project, since more essential research still needs to be done in this area, albeit nothing to-date fully reliable.

2. Face recognition. This thesis deals with the detection of a face and not the actual identity of the person in the image.

3. Working with 2D profiles (in this work only 1D profiles are considered).

4. Working with PCA of the gray level appearance of the landmark points. It seems that it works better only on RGB images.

5. Restriction to 2D information and not 3D.

6. Not using 64bit images (RGB) but 8bit images (gray scale images).

7. Working with real time (RT) ASM.

8. Working with Active Appearance Models (AAMs)

1.2 Layout of the thesis

In the second chapter, some basics in statistics are introduced. In the third chapter, a very detailed description of the Active Shape Model method is given, firstly explaining the classical approach and then extending it to a multi resolution approach. In the last part of this chapter the Viola/Jones face detector is mentioned. Subsequently in the fourth chapter, the software developed for this thesis is presented. It includes the most important classes and also defines - explains all the functions written for this thesis. In the fifth chapter, results on real face data are demonstrated together with comments about the method used. Finally, the last chapter contains discussions and conclusions with respect to the output results of the method applied and recommendations for improvements.

2

Background Statistics

This chapter begins by introducing some basics in statistics, with the intention of understanding the Principal Component Analysis approach described later on in this chapter. Principal Component Analysis is a mathematical procedure used in the statistical part of the Active Shape Model method for building (in general) the object model. The reader of this chapter should not restrict himself to the material provided here but consider it as a way of understanding the basic idea behind it and thus be aware how the statistical part of the Active Shape Model works.

2.1 Introduction to Statistics

2.1.1 Mean

The *arithmetic* mean, or the "standard" average of a population, x is equal too:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{n} x_i \tag{2.1}$$

where $\bar{x}$ is the mean value, N is the total number of the population and x_i are the values in the population.

2.1.2 Standard Deviation

Standard deviation is a widely used method for measuring the variability in statistics and probability theory. It shows how much variation or "dispersion" the data have from their average (also known as mean or expected value). A low standard deviation indicates that the data points tend to be very close to the mean, whereas high standard deviation shows that the data are spread out over a range of values.
Consider having a random variable $X = [x_1, x_2, \ldots, x_n]$, with each value having the same probability. Then the standard deviation, σ, is defined as:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{2.2}$$

where **μ** is the mean and **N** the total number of population.

If, instead of having equal probabilities, the values of the random variable have different probabilities, then the standard deviation will be:

$$\sigma = \sqrt{\frac{1}{N}p_i\sum_{i=1}^{N}(x_i - \mu)^2} \tag{2.3}$$

where,

$$\mu = \frac{1}{N}\sum_{i=1}^{N}p_i x_i \tag{2.4}$$

Have in mind that trying to estimate the ***standard deviation of the sample***, is different from estimating the ***sample standard deviation***.

In the first case the standard deviation is computed by the formula [2.2], whereas in the second case it would be divided with *N-1* samples.

2.1.3 Variance

In probability theory and statistics, the variance is a measure of how far a set of numbers are spread out from each other. It is one of the several descriptors of a probability distribution, describing how far the numbers lie from the mean (expected value). Mathematically, it is calculated by taking the standard deviation and rising it to the power of 2.

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu) \tag{2.5}$$

2.1.4 Covariance

In probability theory and statistics, covariance is a measure of how much two variables change together. Variance is been considered being a special case of the covariance when the two variables are identical. The formula for computing the covariance matrix of two variables X and Y is:

$$COV(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{x})(Y_i - \bar{y})}{N} \tag{2.6}$$

where $\bar{x}$, $\bar{y}$ are the mean of X and Y respectively and N is the size of the population. Again, there may be times where the above formula is divided by N-1 as said previously.

In case of more than two populations (Lets say *p* size) the covariance matrix S, would look like:

$$S = \begin{bmatrix} s_1^2 & s_{12} & \cdots & s_{1p} \\ s_{21} & s_2^2 & \cdots & s_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p1} & s_{p2} & \cdots & s_p^2 \end{bmatrix} \tag{2.7}$$

where on the main diagonal are the variances of every variable population and on the off diagonal the covariance between them.

2.2 Principal Component Analysis

2.2.1 What is Principal Component Analysis?

Principal Component Analysis (PCA), also known as Hotelling transform is a method that reduces the dimensions of the data by computing the covariance matrix between the data. The first people that started working on this method where Pearson (1901) and Hotelling (1933). Pearson was involved in trying to find lines and planes that best fit a set of points in a *n* dimensional space. On the other hand, Hotelling tried to increase his "components", that is the variance in the original variables also known as "principal components". Both Pearson and Hotelling, came across with the *eigenvalue problem* (described later on in this chapter), which was hard to solve for an order higher than four and a computer system was needed to process all this information. Today PCA, with the help of powerful computer systems, is a method widely used and established in different fields of applications.

2.2.2 The Basic Principle

As in other transformations (e.g Helmert Transform), PCA tries to transform data from one system to another, where a new set of basis vectors are used. However, in the PCA case, the basis vectors do not remain constant but they vary based on the data being transformed.
PCA is a linear transformation and the new basis vectors, denoted as e_i are orthogonal between them:

$$e_i^T e_j = \delta_{ij} = \begin{cases} 1 \ if \ i = j \\ 0 \ if \ i \neq j \end{cases} \tag{2.8}$$

where δ_{ij} is the Dirac's delta function.

Since PCA is a linear transformation, it means is has translation and rotation parameters. Thus, if ***x*** are the input data and ***y*** the transformed data, the transformation is:

$$y = A(x - \mu_x) \tag{2.9}$$

where A contains the new basis vectors and thus $A = [e_1 \, e_2, \ldots, e_n]^T$ and μ_x is the mean of the data set.

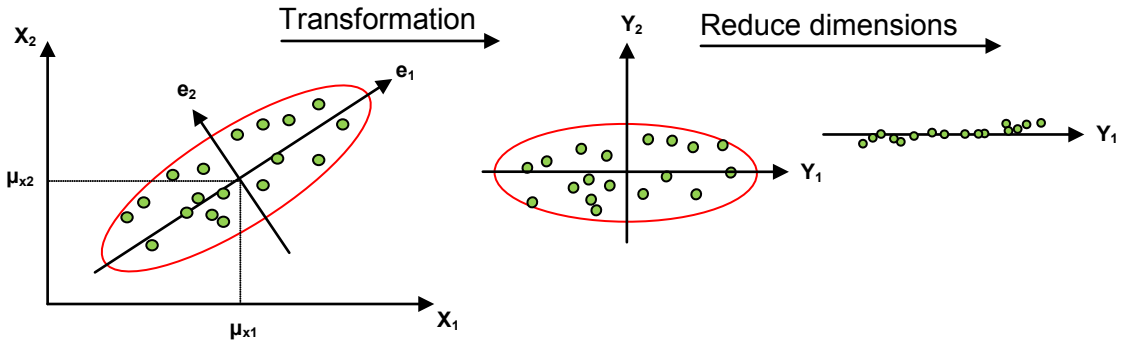


Figure 2.1: *Basic Principle of PCA*

The first figure shows the input data where each sample of the data is denoted as $x_i = [x_{1i}\ x_{2i}]^T$. Second figure illustrates the transformed data where each sample is denoted as $y_i = [y_{1i}\ y_{2i}]^T$ and is calculated using equation [2.9]. The first two sub figures of figure [2.1] show the transformation from one system to another with the main variance expressed in the first two variables Y_1 and Y_2. Ignoring the second variable, the main variance of the data is kept. In this case the information is presented in a more compact form (see last sub figure).
In figure [2.1], ignoring one variable might not make any difference. Nevertheless, if the dimensionality of the data is 30 (random number) and their variability could be expressed by only two variables, then a compact representation could be achieved. Therefore, PCA is applied in cases where high dimensionality of data is present.
Although PCA is used to reduce the dimensions of the data, one of its main properties is that new data set could be created, similar to the initial ones using only the new variables. In figure [2.1] the data are uncorrelated and remain as is in lower dimensions too.

2.2.3 Mathematics Behind PCA

As already mentioned, the transformation is given by equation [2.9]. To find A and subsequently apply the transformation, the following steps have to be done:

Equation [2.9] could be written as $y = Ax'$ where $x' = x - \mu_x$ and the inverse transformation is equal to $x' = A^{-1}y$. Considering that ***A*** is an orthogonal matrix where $A^{-1} = A^T$, then $x' = A^T y$ can be re written as:

$$x' = [e_1\ e_2\ \ \dots\ e_3] \cdot y = \sum_{i=1}^{n} y_i e_i \tag{2.10}$$

If *m* components are used $(m < n)$, then some information could be lost during the inverse transformation and that could lead to implausible x' shapes. Thus, an estimate $\hat{x}'$ is defined:

$$\hat{x}' = \sum_{i=1}^{m} y_i e_i \tag{2.11}$$

As mentioned in the beginning of this section, main purpose is to find ***A*** so that the difference between x' and $\hat{x}'$ is:

$$\alpha = E\{(x' - \hat{x}')^T \ (x' - \hat{x}')\} \tag{2.12}$$

Putting equations [2.10] and [2.11] together results:

$$\alpha = E\left\{\left(\sum_{i=m+1}^{n} y_i e_i\right)^T \quad \left(\sum_{i=m+1}^{n} y_i e_i\right)\right\} \tag{2.13}$$

Because of the orthonormality introduced in [2.8], equation [2.13] becomes:

$$\alpha = E\left\{\sum_{i=m+1}^{n} {y_i}^2\right\} \tag{2.14}$$

From equation [2.9] it comes out that $y_i = e_i^T x'$ giving:

$$\alpha = E\left\{\sum_{i=m+1}^{n} \left(e_i^T x'\right)^2\right\} = E\left\{\sum_{i=m+1}^{n} \left(e_i^T x'\right)\left(e_i^T x'\right)\right\} \tag{2.15}$$

Since $e_i^T x' = {x'}^T e_i$, equation [2.15] becomes:

$$\alpha = E\left\{\sum_{i=m+1}^{n} \left(e_i^T x'\right)\left({x'}^T e_i\right)\right\} = E\left\{\sum_{i=m+1}^{n} e_i^T x' {x'}^T e_i\right\} \tag{2.16}$$

Changing the order of the summation (because e_i is deterministic) it exists:

$$\alpha = \sum_{i=m+1}^{n} e_i^T E\{x' {x'}^T\} e_i \tag{2.17}$$

Setting as $C_x = x' {x'}^T$ then:

$$\alpha = \sum_{i=m+1}^{n} e_i^T C_x \, e_i \tag{2.18}$$

At this point, in order to find the best/optimal e_i the square error function should be minimised. This is done by defining a function using Lagrange multiplier:

$$g(e_i) = e_i^T C_x e_i - \lambda\left(e_i^T e_i - 1\right) \tag{2.19}$$

Applying partial first order derivative to the previous function it becomes:

$$\bigvee g(e_i) = 0 => \bigvee g(e_i) = C_x e_i + C_x^T e_i - \lambda 2 e_i = 0 \tag{2.20}$$

Since $C_x = C_x^T$, equation [2.20] becomes:

$$C_x e_i - \lambda e_i = 0 => \tag{2.21}$$

$$(C_x - \lambda I) \cdot e_i = 0 \tag{2.22}$$

where ***I*** is the identity matrix.

2.2.4 The Eigenvalue Problem

Equation [2.22] is known as the *eigenvalue problem* and it's a problem which is seen in other applications besides PCA. To solve this problem, the determinant of equation [2.22] is taken. Due to the fact that equation [2.22] is a homogeneous system of the form $Ax = 0$ and has no trivial solution, the determinant of the coefficient matrix is zero:

$$determinant(C_x - \lambda I) = 0 \tag{2.23}$$

Finding the polynomials of the above equation and getting the square root of them, these are the eigenvalues λ_i of C_x. Every eigenvalue λ_i corresponds to the i'th eigenvector $\mathrm{e_i}$ and also $\lambda_i \geq \lambda_{i+1}$.
After finding the eigenvectors, they have to be arranged as row vectors in the *A* matrix and then equation [2.9] is ready to be used.

2.2.5 Choosing which components to ignore

Since the eigenvectors have been computed, the input data could be transformed. The only problem is that the dimensions of the input data haven't been reduced. Thus, the task now is to reduce the dimensions of the data without losing much information about the initial data.
There are several existing methods for this purpose. Nevertheless, the one that is worth describing is the *m – method*. The main goal of all methods is to maintain as much variation of the initial data as possible in the smallest space possible.
According to the m – method, since the i'th eigenvalue is equal to the variance of the i'th variable and given that $\lambda_i \geq \lambda_{i+1}$, the amount of variability kept is defined as:

$$I_k = \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \cdot 100\% \tag{2.24}$$

where λ_i is the i'th eigenvalue, *m* is the number of eigenvectors used, *n* is the dimensions of the input data and I_k is the percentage of variance kept in the transformation.

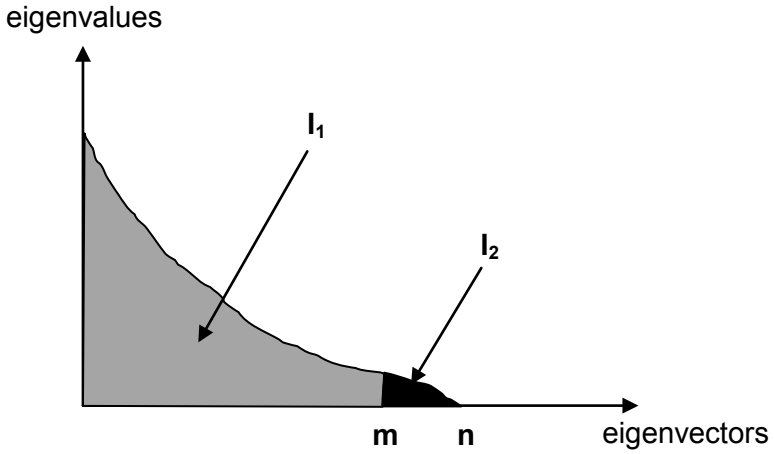


Figure 2.2: *The preserved and removed information*

The figure above shows the eigenvectors in the X axis with their associated eigenvalues in the Y axis.
The value *m* is used as a threshold value between the selected and ignored eigenvectors, I_1 represents the variation kept and I_2 the ignored one. This technique is known as the *m – method*.

2.2.6 How to apply PCA

To apply PCA five steps have to be done. The first step is to acquire the data. In this case the more data used the better the method will work. The second step is to compute the covariance matrix C_x of the input data. This could be calculated from $C_x = VV^T$ where $V = [x_1', x_2', \dots, x_n']$ and $x_i' = x_i - \mu_x$. Third step is to derive the eigenvectors and eigenvalues from equation [2.23]. Subsequently, when the eigenvalue problem has been solved, it has to be determined which eigenvalues to keep. For this purpose the *m – method* could be used. Last step concerns mapping the data in a lower dimension using equation [2.9].

3

Active Shape Models

This chapter describes in detail the Active Shape Model method. It begins with the labeling of the object in training images followed by the alignment of the data sets. It continues with the extraction of some statistical information about the training shapes. Subsequently, a description of the method for getting the gray level appearance information of each model point. Then, the placing of the face model in the testing image to detect the said object. Furthermore, an improved multi resolution approach is described. Last but not least, some basic information of the Viola Jones algorithm is given.

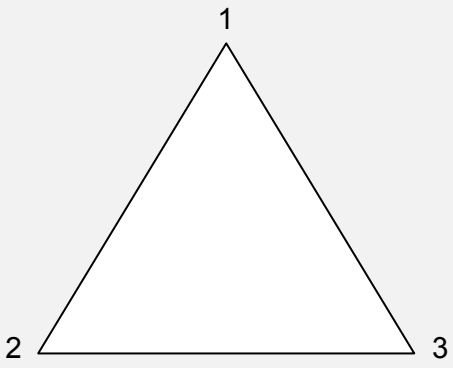
3.1 Introduction

Active Shape Models is a method that was developed by T.F. Cootes et.al for detecting known objects in images. Till now, building rigid models of objects for image understanding was well achieved. However, there are cases where objects of the same class are not identical, thus rigid models wouldn't work, for example the shape of a heart, where it is one object represented from different shapes and sizes. With the method explained in this chapter, new models could be produced from images representing the same object with different shape/ size variations. In addition, Cootes tried to create models that although vary but still preserve the structures of the object class they belong to.
For the method to work, some points are needed that represent the shape of the object within different training examples. These sets are then aligned in order to minimize the variation between equivalent points. From these aligned data sets, a "Point Distribution Model" is created, which gives a mean shape of the aligned shapes and some model parameters that express the different variations within the training set.
Given this model and an image that contains this object, an iterative scheme could be applied that would find the appropriate pose and model parameters which best fit the model in the object.

In order for the process to start, some approximate pose parameters have to be given to place the model within the image. That involves finding the best translation, rotation and scaling parameters which will best fit the model in the image. This method is also very similar with the Active Contour Models of Kass et.al. The main difference is found in the global shape constraints, hence to differentiate from the Active Contour Models it was named Active Shape Models. The main advantage of this method is that the model can only deform in ways which are similar to those in the training set.

3.2 Shapes

A shape, by definition in [15] is "***all the geometrical information that remains when location, scale and rotational effects are filtered out from an object***". That means that it remains invariant to Euclidean similarity transformations. A shape is described by a set of points which in this thesis are expressed in the form seen in Eq. [3.1]. An example of a shape is shown below, together with its points (in this example arbitrary coordinates) that define it.



Number of Points	x	y
1	2.0	2.0
2	0.0	0.0
3	4.0	0.0

$$x = \begin{bmatrix} 2.0 \\ 2.0 \\ 0.0 \\ 0.0 \\ 4.0 \\ 0.0 \end{bmatrix}$$

Figure 3.1: Left *a simple shape, in a triangular form, defined by three points.* ***Middle*** *is the same shape defined as an array.* ***Right*** *is the same shape described as a vector (Self illustrated).*

The points from the above figure move in some invariant way. If the shape moves (translated), then the shape remains the same. If it's rotated or scaled, it still remains the same shape. The edges are not considered part of the shape but are used to get the relationship between the points.
The distance between two points is the Euclidean distance between the points. The distance between two shapes, according to Procrustes in [2] is equal to the sum of the distances between their corresponding points. The Procrustes distance between two shapes x_1 and x_2 is the root mean square distance between the shape points after alignment $\sqrt{(x_1 - x_2) \cdot (x_1 - x_2)}$. The *centroid* $\bar{x}$ (also known as the position of the shape), is the mean of the point positions. The *size* of the shape is defined as the root mean square distance between the shape points and the centroid.

3.3 Point Distribution Model

3.3.1 Labeling the training set

Every shape is expressed by a number of landmark points, also known as "landmark points". According to Bookstein [25], every landmark point is described and categorized according to their usefulness. Thus, they can be expressed in three different categories:

- Points that represent a particular part of the object, such as the centre of an eye or sharp boundaries.

- Points that are placed at the highest part of the object with a particular orientation or curvature extrema.

- Points which are being interpolated from other points of types 1 and 2. In this case the points are equally spaced.

The figure below indicates the annotation of a face that contains 79 points. In general, as per Bookstein, points of type 1 are preferable to those of type 2, since they are much easier to identify. Nevertheless, points of type 2 and 3 are always necessary due to the fact that they are used to describe the shape of the object with much more detail.

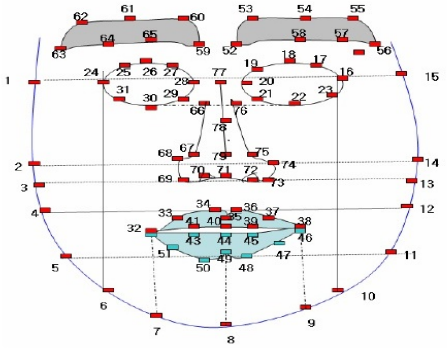


Figure 3.2: *Seventeen nine landmark points describing the shape of a face (Marios Savvides, et.al, 2009).*

The simplest method for choosing landmark points for each training image, is usually achieved by an expert (manually). Though, this can be very time consuming especially when the number of the training images is very large. In practice, automatic and semi – automatic methods have been developed to do this annotation as automated as possible.
As mentioned previously, every shape **x**, is described in 2D by *n* points placed in a vector of the form:

$$x = (x_1, y_1, \dots, x_n, y_n)^T \tag{3.1}$$

3.3.2 Aligning the Training Set

In order to be able to compare equivalent points from different shapes in the training set, they should be aligned within a common coordinate frame. There is considerable literature concerning the alignment of shapes. Although the most frequently used method is based on Procrustes Analysis [2], Cootes in [25] explains a modification of this method. This is done by scaling, rotating and translating the shapes so that the weighted sum of squares of distances between equivalent points is minimized (energy function):

$$E_j = \left(x_i - M(s_j, \theta_j)[x_j] - t_j\right)^T W \left(x_i - M(s_j, \theta_j)[x_j] - t_j\right) \tag{3.2}$$

where:

$(\boldsymbol{x_i}, \boldsymbol{x_j})$ are the coordinates of a pair of shapes
$\boldsymbol{M(s_j, \theta_j)[x_j]}$ is a rotation by theta and scaling by s analyzed as:

$$M(s,\theta)\begin{bmatrix} x_{jk} \\ y_{jk} \end{bmatrix} = \begin{bmatrix} s \cdot cos\theta & s \cdot sin\theta \\ -s \cdot sin\theta & s \cdot cos\theta \end{bmatrix}\begin{bmatrix} x_{jk} \\ y_{jk} \end{bmatrix} \tag{3.3}$$

where $\boldsymbol{t_j}$ is the translation of the second shape:

$$t_j = [t_{xj}, t_{yj}, \dots, t_{xj}, t_{yj}]^T \tag{3.4}$$

and **W** is a diagonal matrix of weights for each point.
The meaning of the weights is to provide the amount of significance of each point, that is to show which points are more stable compare to others. So if R_{kl} is the distance between points k, l and $V_{R_{kl}}$ is the variance of this distance over the training set, then the weight w_k for the k^{th} point is equal too:

$$w_k = \left(\sum_{t=0}^{n-1} V_{R_{kl}}\right)^{-1} \tag{3.5}$$

If a point tends to move more, compared to the other points in the training set, then the variance is large, otherwise is small.
The alignment procedure requires normalization of the shapes. This involves setting the current mean shape at some suitable defaults for translation, rotation and scaling. For the translation phase that is to offset the shape to the origin so its center of gravity (C.o.G) be at the point zero of the coordinate system. This is achieved by removing the translation from the center of the shape with respect to the systems origin from all its points. This will result an automatic shift of the shape to the origin. The next step is to normalize the scale factor by scaling the mean distance to the origin for each component to 1 (The mean Euclidean distance will then be $\sqrt{2}$). The last step of the normalization is the rotation part, where according to Cootes [25], it should be a rotation where a specific part of the object is always on top. A solution to this could be a rotation which is equal to the mean value of the arctangent of all landmark points. In mathematical notation it's equivalent to:

$$\overline{angle} = \frac{\sum_{t=0}^{n-1} atan2\left({}^{Y_t}/_{X_t}\right)}{n} \tag{3.6}$$

The final rotation is obtained by subtracting the mean angle from every $atan2\left({}^{Y_t}/_{X_t}\right)$.

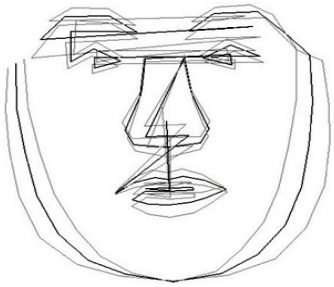
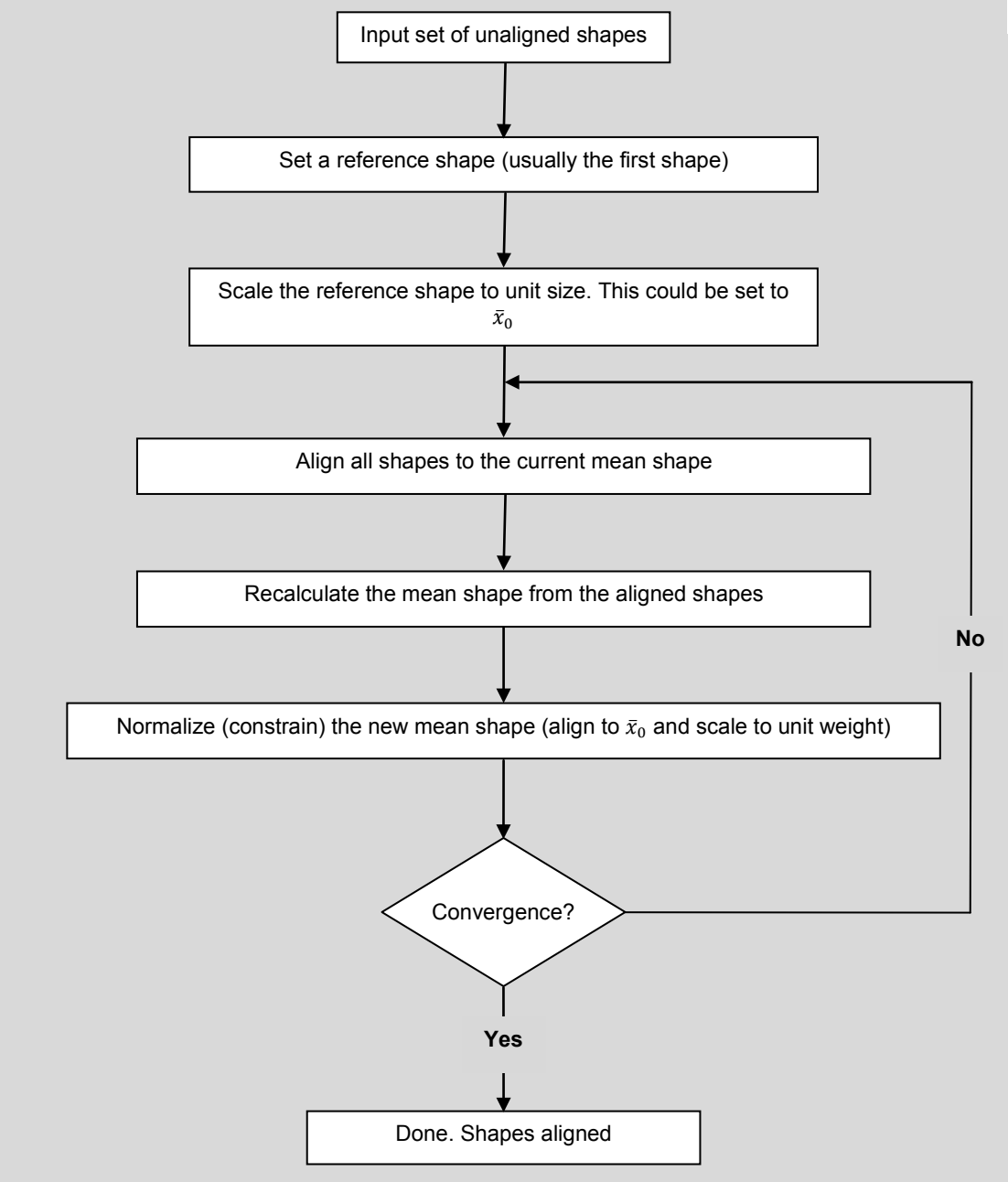


Figure 3.3. *Alignment of a set of faces with the mean shape drawn on top with a thicker line (Milborrow, 2007).*

Another solution , which is the one implemented within this thesis is to always rotate a shape according to the rotation that the reference shape has (usually the first shape in the training set).
Normalizing the mean shape to a default scale and pose (translation and rotation) ensures that the algorithm will converge. Experiments have shown that the alignment converges with very few iterations. Note, that normalizing the mean shape and then align all others to it, is not the same as normalizing every shape individually. If every shape was constrained to have a normalized scale equal to 1, a distortion of the model could occur during the alignment face. On the other hand, if all shapes have to align to the current mean shape, then they will have a scale similar to that of the mean. Result from the alignment of a set of training faces is shown above in Fig. [3.3] together with the mean shape drawn on top. A simple alignment algorithm proposed for face alignment is as follows:



Graph 3.1: *Alignment algorithm for faces (Self illustrated).*

3.3.3 Capturing the statistics of the aligned shapes

Having a set of aligned shapes, their statistic could be derived. All aligned shapes form a distribution within a n^d dimensional space. From this distribution, new examples could be generated similar to the ones coming from the aligned shapes. After the alignment of a training set, some clouds are created around each landmark point. These clouds could be either dense or defused. In the case of dense, the variability is less and if defused, much greater. The Point Distribution Model in this case tries to create a model of this variability for each one of these clouds.
Have in mind that the position of each landmark is not independent but dependent from all other points. Having every aligned shape represented by a point in a $2n$ dimensional space, then in case of N number of aligned shapes, there exists N points in the $2n$ space. All these N points are assumed that they lie within what is called **Allowable Shape Domain** (ASD). The size and shape of the ASD depends upon the points distribution within this ASD. Every $2n$ point within this ASD can create shapes similar to the ones in the training set. According to Cootes [25], an assumption has been made that the shape of this domain is approximately Ellipsoidal with the center of the ellipse being the mean shape and the major axis is the variation of the mean shape.

Given a set of N aligned shapes x_i, the mean shape $\bar{x}$ (the center of the ASM) could be calculated from:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.7}$$

Then the $2n \times 2n$ covariance matrix S can be calculated using:

$$S = \frac{1}{N}\sum_{i=1}^{N} dx_i \, dx_i^T \tag{3.8}$$

The principal axes of the ellipsoid which describe the variation of a shape are expressed by p_k $(k = 1, \dots, 2n)$, unit eigenvectors of S, so that:

$$Sp_k = \lambda_k p_k \tag{3.9}$$

where λ_k is the eigenvalue of S and $p_k^T p_k = 1$.

According to the theory of Principal Component Analysis, the eigenvectors of the covariance matrix S corresponding to the largest eigenvalues, describe or capture most of the variation of the shapes statistics (they represent the largest axes of the ellipsoid). Thus, the 2^d ellipsoid could be approximated by an ellipsoid of lower dimensions, let's say ***t***.
One way of calculating the smallest number of modes ***t*** that describe the largest part of the variation of λ_T where,

$$\lambda_T = \sum_{k=1}^{2n} \lambda_k \tag{3.10}$$

is the total variance of all the variables:

$$\sum_{i=1}^{t} \lambda_i \geq f_u \lambda_T \tag{3.11}$$

where f_u is the part of variance to be explained by the shape model and also defines the number of modes (experiments made, say it should have a value of 0.98 with alignment or 0.995 without alignment).
Any point in the ASD can be reached - expressed by taking the mean corresponding point and adding a linear combination of the eigenvectors. Thus, every shape in the training set can be approximated by taking the mean shape and a combination of the deviations obtained from the first ***t*** modes:

$$x = \bar{x} + Pb \tag{3.12}$$

where,

$\bar{x}$ is the mean shape (of the ASD),
$P = (p_1, p_2, \dots, p_t)$ is the matrix with the first ***t*** eigenvectors and
$b = (b_1, b_2, \dots, b_t)^T$ is a vector of weights (model parameters).

By varying the vector of weights *b* within suitable limits, new shapes could be generated similar to those in the training set. The limits for b_k are derived by examining the distribution of the parameter values used to derive the training set. Thus, the limits applied are:

$$-3\sqrt{\lambda_k} \leq b_k \leq 3\sqrt{\lambda_k} \tag{3.13}$$

which is three standard deviations from the mean, where most of the variation is expressed.

Another method for choosing model parameters b_k is to compute the Mahalanobis distance D_m from the mean and if the difference is less than a D_{max} (where Cootes in most papers sets it to 3) then the value for the specific element of the model parameter vector is maintained. In a mathematical notation that is:

$$D_m^2 = \sum_{k=1}^{t} \left(\frac{b_k^2}{\lambda_k} \right) \leq D_{max}^2 \tag{3.14}$$

3.3.4 Understanding the Shape Model

It is easier to use rectangles to describe the shape model rather than complicated face shapes. The figure below shows a number of rectangles, symmetrical around the origin. To specify the four points of any of these rectangles, eight numbers are required: four, x and y coordinates. Taking into account what was said about the symmetry, the rectangles could actually be described by just two parameters: its width and height.

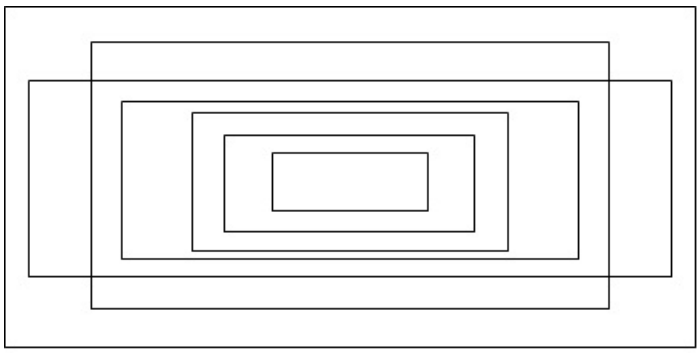


Figure 3.4: *Rectangular shapes (Self illustrated).*

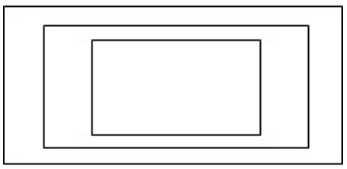


Figure 3.5
Variation of the first component (Self illustrated).

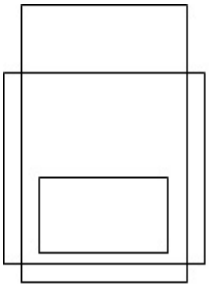


Figure 3.6
Variation of the second component (Self illustrated).

Using Eq. [3.12] to generate a shape model (in this case a rectangle) the form it takes is:

$$\hat{x} = \begin{bmatrix} 23 & 12 \\ -23 & 12 \\ -23 & -12 \\ 23 & -12 \end{bmatrix} + b_0 \begin{bmatrix} 12 & 4 \\ -12 & 4 \\ -12 & -4 \\ 12 & -4 \end{bmatrix} + b_1 \begin{bmatrix} -4 & 12 \\ 4 & 12 \\ 4 & -12 \\ -4 & -12 \end{bmatrix} + \cdots \tag{3.15}$$

The eigenvalues of the covariance matrix S sorted by descending order are 3778, 444, 2, 0.1 etc. There are eight eigenvalues altogether, two for every landmark point, that is one for x and y element. From these eight eigenvalues, only the first two remain and that's because they represent most of the shapes variation (>98%). Thus, the shapes can be parameterized by just two parameters, b_0 and b_1. The first parameter varies the first eigenvector and in this case it changes the size of the generated rectangle. The second parameter varies the second eigenvector, which in this example adjusts the aspect ratio of the shape.

3.3.5 An Example of a Shape Model

Figure (3.7) shows example shapes from a training set of 300 labeled faces. Each image is annotated with 133 landmarks.

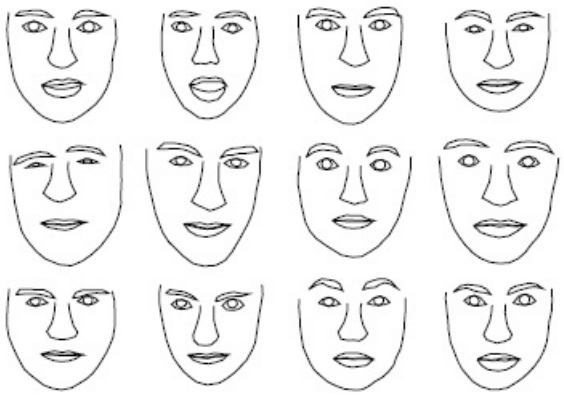


Figure 3.7: *Example shapes from training set of shapes* (*T.F. Cootes & C.J.Taylor, 2004*).

The shape model has 36 modes which correspond to approximately 98% of the total variance of the landmark points. The figure below shows the result of varying only the first three shape parameters within ± 3 standard deviations from the mean shape and setting the other shape parameters to zero.

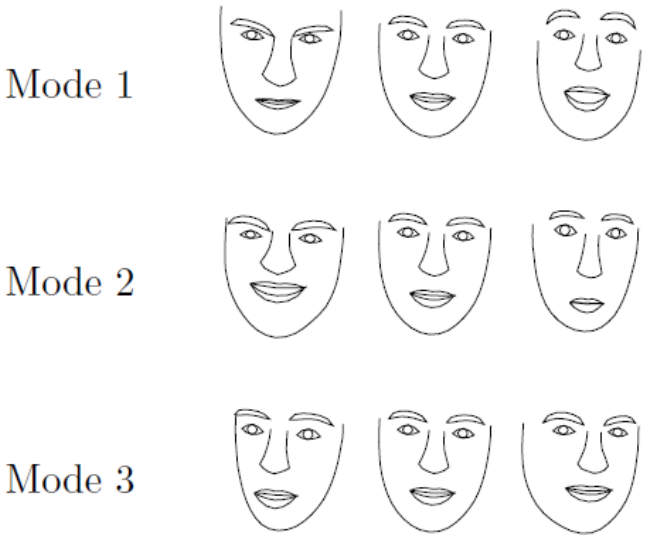


Figure 3.8: *Varying each of the first three shape parameters within ±3 s.d (T.F. Cootes & C.J.Taylor, 2004).*

3.4 Modeling Gray Level Appearance

In order to detect an instant of a model within new images, not only the shape but also gray-level appearance is important. This is done by examining the statistics of the gray level appearance around the neighbor of each landmark point. Since every point corresponds to a particular part of an object, the gray level appearance of that point in different example images will not be identical but quite similar. Cootes in [28] uses the gray level information to compute the movement of the points in a different position and finally detect the contour of the object as accurate as possible.

3.4.1 Computing Normal to Boundary

Although the gray level information around the area of each point is considered being 2D, within this thesis the limitation will be done only on 1D information.
In this case, the gray level information is derived from 1D profiles, normal to the boundary passing through the points. This profile normal, also known as “whisker” is defined from the following equations:

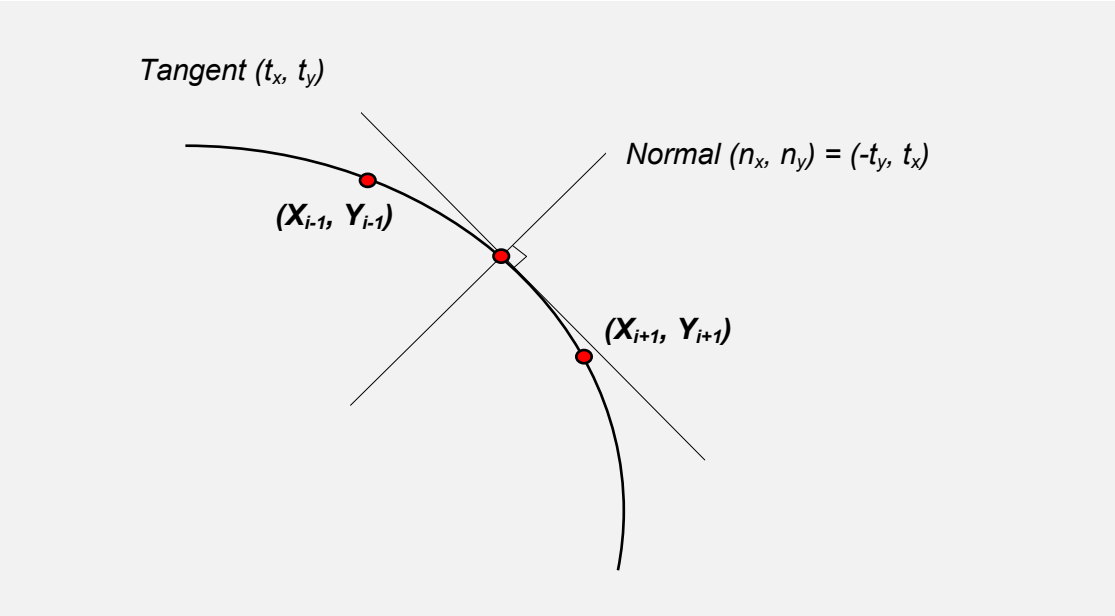


Figure 3.9: *Normal to a landmark point (Self illustrated).*

$$(t_x, t_y) \approx \frac{(d_x, d_y)}{\sqrt{d_x^2 + d_y^2}} \tag{3.16}$$

where $d_x = X_{i+1} - X_{i-1}$ and $d_y = Y_{i+1} - Y_{i-1}$

The normal to the boundary is created by computing firstly the tangent to the current point and then rotating it clockwise by 90 degrees. The normal is a unit vector and has a length of one.

3.4.2 Sampling along profiles

If the profile runs from p_{istart} to p_{iend} and has length of n_p pixels, then every interval point on this profile normal is computed from:

$$y_{ik} = p_{start} + \frac{k-1}{n_p - 1}(p_{iend} - p_{istart}) \tag{3.17}$$

where y_{ik} is the k[th] point on the profile normal *i* (see Fig. [3.10]).

Usually the distance between each interval point is equal to 1 pixel.

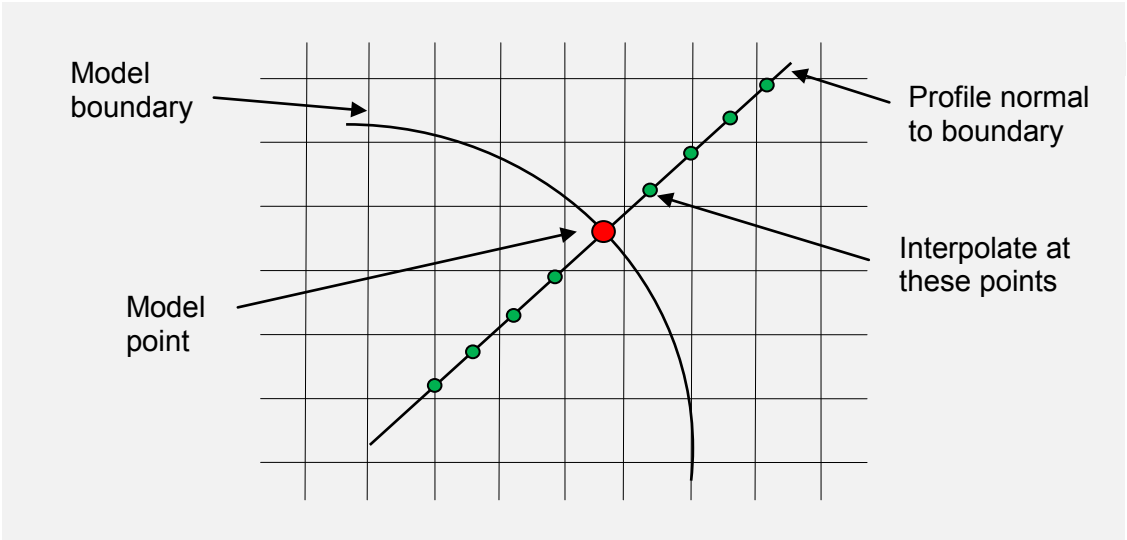


Figure 3.10: *Interpolated points on the whisker (Self illustrated).*

3.4.3 Forming a profile

Having computed the normal at each model point together with the linear intervals on these lines, the last step is to form the gray level appearance of each point. This is achieved by setting each element (interpolated or interval point) on the profile vector (whisker) of a landmark point, to the gray level intensity (0 - 255) of its image below it (see Fig. [3.11]).

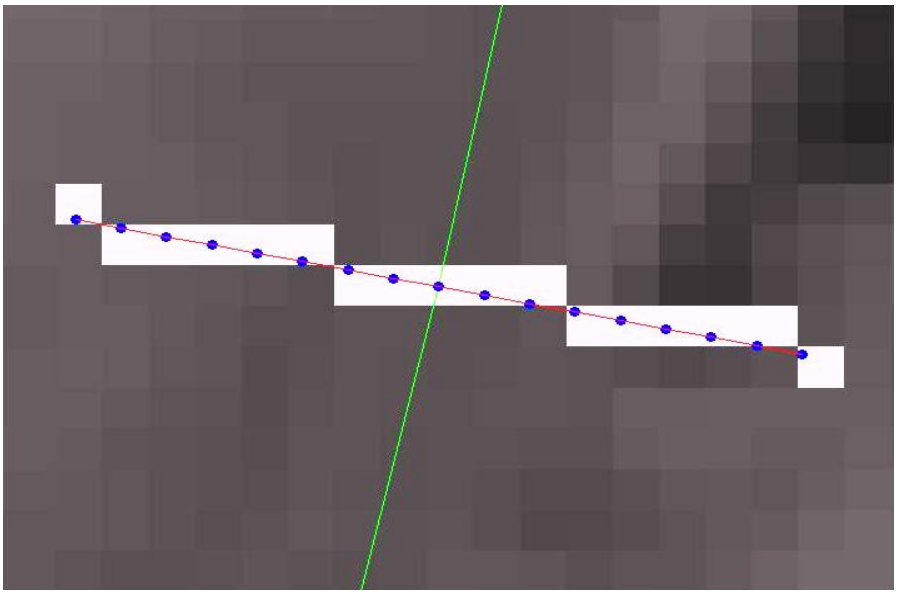


Figure 3.11: *Profile normal of a point (Self illustrated).*

The above figure shows interpolated points (blue color) placed on a profile normal (red colour) of a model point (point at which the red and green lines intersect) with the corresponding pixels that they belong to, marked with white colour. Sometimes you may have two interpolated points placed in one pixel, so in that case they will be assigned with the intensity of that pixel.

The next step is to replace each profile element by the intensity gradient.

This is done by setting the difference between the intensity of the next point and the previous point, divided by two (in some literature the first derivative might be found without the deviation by 2 but only with the intensity differences, which is also considered correct):

$$g_{ijk} = \frac{\left(I_j\left(y_{i(k+1)}\right) - I_j\left(y_{i(k+1)}\right)\right)}{2} \tag{3.18}$$

Divide each element of the resulting vector with the sum of all absolute elements of that vector (normalizing the profile):

$$g'_{ij} = \frac{g_{ij}}{\sum_{k=1}^{n_p} \left|g'_{ijk}\right|} \tag{3.19}$$

The purpose of normalization is to reduce the effect of image lighting and contrast.
For each point *i*, calculate the mean normalized profile:

$$\bar{g}_i = \frac{1}{N_S} \sum_{j=1}^{N_S} g'_{ij} \tag{3.20}$$

Last but not least, calculate the $n_p \times n_p$ covariance matrix, S_{gi} that will give a statistical description for the profile of the current label point.
Given a simple numerical example, suppose a profile normal consists of (2·8+1) interpolated points (that is 8 points at each side of the profile normal plus one for the current model point position). The mean normalized derivative profile $\bar{g}_i$ will then have a size of 17x1 and a covariance matrix of size 17x17.

3.5 Applying shape model

Having generated acceptable shape models of an object within ± 3 standard deviations and the grey level of appearance around each point, this information could be used to find examples of the modeled structures within an image. In general, this procedure involves two mandatory stages:

- Number of hypothesis made, giving approximate locations of the model points (iteration process)
- From all hypotheses, the best one will be chosen.

The initial hypothesis involves finding and placing the model shape from its local coordinate system to the image system. This is done by applying the “appropriate” initial pose (translation, rotation and scaling) and shape parameters, assuming that the position of the object within the searching image is known.
An instance of a model, X, for a set of pose and shape parameters are given by:

$$X = M(s, \theta)[x] + X_c \tag{3.21}$$

where,

$M(s, \theta)$ is a rotation by $\boldsymbol{\theta}$ and a scaling by $\boldsymbol{s}$

x is the position of the model shape within the local coordinate frame
X_c is the position of the center of the model in the image frame.

Cootes in [28] suggests an iterative method for refining the shape and pose parameters so as to give a better match between a model instance and the structures in the image. The algorithm is as follows:

1. Initialize the shape parameters ***b*** to zero
2. Generate the model instance $x = \bar{x} + Pb$. This means that the model instance will be equal to the mean shape.
3. Find the pose parameters to place the model shape into the image
4. Compute the proposed adjustments to the points
5. Find the pose parameters that align the current shape to the updated shape.
6. Project back to the local coordinate system the updated shape and compute the coordinate difference from the previous state of the shape.
7. Updated the shape parameters ***b***
8. Apply constraints on ***b***
9. If not converged[1], return to step 2.

3.5.1 Calculating a suggested movement for each model point

Given an initial estimate of the model points position within the image, new suggested points have to be calculated which will move each model point to a better position. Making the assumption that model points represent the boundary of an object, their movement to a better position (adjustment) will shift them towards the edges of the image object. Having the gray scale appearance of each model point from training, the adjustment involves finding the position/region which better matches the sampled model.
For an arbitrary point of the shape model placed in the image, a derivative (or sample) profile g is extracted of some length $l(> n_p)$. The profile model is then shifted to each position of the sample profile till it finds the position where it best fits\matches.

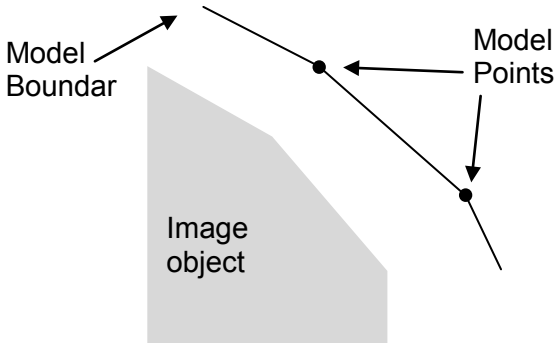


Figure 3.12: *Part of an object boundary approximating the edge of an image object (Self illustrated).*

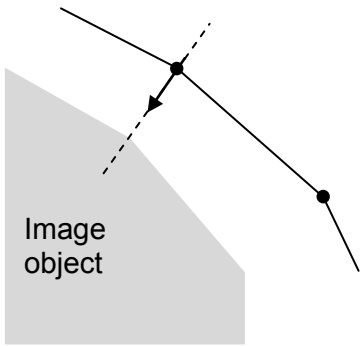


Figure 3.13: *Suggested movement of a point along the normal boundary, towards the direction where the profile model best fits the sampled profile (Self illustrated).*

[1] Convergence means applying an iteration which will not produce any significant change in the pose nor the shape parameters. The following chapters will provide a more detailed explanation of the complete process.

Mathematically speaking, the process of trying to find the position in the sample profile where the profile model best fits is expressed by the square of the *Mahalanobis distance*:

$$f(d) = (h(d) - \bar{g})^T S_g^{-1} (h(d) - \bar{g}) \quad (3.22)$$

where,

$h(d)$ is a sub – interval of g of length n_p, centered at point d within the sample profile
$\bar{g}$ is the normalized mean gray level appearance of that model point and
S_g^{-1} is the inverse of the covariance matrix of the model point

The value of $f(d)$ decreases as the fit improves. Thus, the point of best fit is the one for which $f(d)$ is minimum.

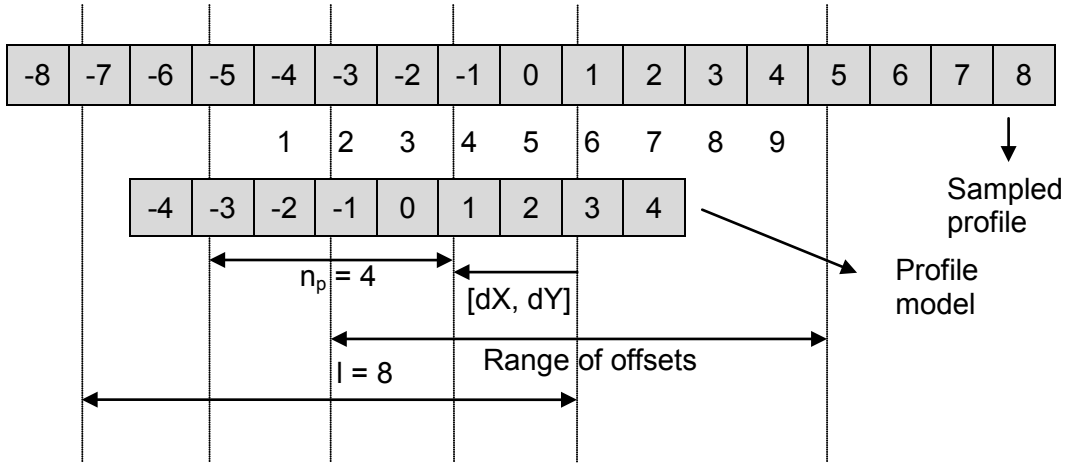


Figure 3.14: *Movement of the profile model along the sampled profile (Self illustrated).*

The above figure is used for the reader to have a better "visualized" representation of how the quality of fit is found between the sample and model profile.
According to this, the sampled profile has a length equal to 17 (2·l+1) pixels and the model profile of 9 (2·n_p+1) pixels.
The model profile, as mentioned previously should always be smaller than the sample profile. Subsequently, the model profile runs from left to right at 9 different locations and the quality of measure is calculated from Eq. (3.22). The number of offsets are calculated from:

$$Range\ of\ offsets = (2l + 1) - \left(2n_p\right) \quad (3.23)$$

As it can be seen, the minimum position (best fit) is found at the location 3 out of 9 but because all distances start from the center point of the samples profile (that is 0) it's actually minimum at the location -2 of the sampled profile. The position of the minimum, with the starting point being the center of the sampled profile, is calculated from:

$$movement = (minPos - 1) - ((Range\ of\ offsets - 1)/2) \quad (3.24)$$

The displacement vector from the current location the model point is placed on the image, to the location of best fit, is calculated by:

$$dX = n_x \cdot movement \tag{3.25}$$

$$dY = n_y \cdot movement \tag{3.26}$$

By adding the corrections to the current model point, the position of the suggested new point is been calculated.

3.5.2 Computing changes in pose and shape parameters

After adjusting the model shape from its current position X, to a new suggested position [X + dX], the pose and shape changes that connect the current with the new shape have to be calculated. If the current estimate of the model is centred at (X_c, Y_c) with orientation θ and scale s, a new translation (dX_c, dY_c), rotation $d\theta$ and scaling factor $(1 + ds)$ are desirable. These pose corrections can be calculated by using Cootes alignment algorithm, as per Appendix A.
Having adjusted the pose parameters, there remain residual adjustments which can only be computed by deforming the shape of the model. Firstly, the adjustments **dx** are calculated, in the local coordinate frame, that cause the points **X** to move by **dX** when they are combined with some new pose parameters.
If the initial position of the model points in the image frame is given by Eq. [3.21], the new residual adjustments ***dx***, in the local coordinate frame are given from:

$$M\big(s(1 + ds), (\theta + d\theta)\big)[x + dx] + (X_c + dX_c) = (X + dX) \tag{3.27}$$

By doing some matrix operations on Eq. [3.27] it becomes:

$$M\big(s(1 + ds), (\theta + d\theta)\big)[x + dx] = (M(s, \theta)[x] + dX) - (X_c + dX_c) \tag{3.28}$$

and since,

$$M^{-1}(s, \theta)[\quad] = M(s^{-1}, -\theta)[\quad] \tag{3.29}$$

dx is equal to:

$$dx = M\big(s(1 + ds)^{-1}, -(\theta + d\theta)\big)[y] - x \tag{3.30}$$

where,

$$y = M(s, \theta)[x] + dX - dX_c \tag{3.31}$$

Although, Eq. [3.31] calculates the suggested movements of the points ***x*** in the local coordinate system, these movements are not consistent with the shape model. Thus, ***dx*** has to be transformed from its local system into the model parameter space and the corrections are then notated with ***db***.

Adding in Eq.[3.11] the shape corrections ***db***, it becomes:

$$x + dx \approx \bar{x} + P(b + db) \tag{3.32}$$

and by subtracting 3.11 from 3.31 it gives:

$$dx = P(db) \tag{3.33}$$

Solving for ***db***, the final corrections for the shape model are given from:

$$db = P^T dx \tag{3.34}$$

since $P^T = P^{-1}$ due to orthogonality and unit length.

3.5.3 Updating the pose and shape parameters

The equations in the previous section were used to compute the corrections of the pose and shape parameters between the current and updated – suggested shape. The goal now is to apply them in an iterative scheme until the process converges:

$$X_c \rightarrow X_c + w_t dX_c \tag{3.35}$$

$$Y_c \rightarrow Y_c + w_t dY_c \tag{3.36}$$

$$\theta \rightarrow \theta + w_\theta d\theta \tag{3.37}$$

$$s \rightarrow s(1 + w_s ds) \tag{3.38}$$

$$b \rightarrow b + W_b db \tag{3.39}$$

where w_t, w_θ, w_s are scalar weights and W_b is a diagonal matrix of weights, one for each mode. In this thesis, no weights are given and the weights for the alignment algorithm in the searching part are set to identity.
One of the main advantages of the active shape models is that applying a model to an image, it will deform until it converges, still though preserving a shape consistent with the training set. But in order to do so, limits have to be applied to the values of b_k. Cootes in [28] suggests that a shape can be considered acceptable if the Mahalanobis distance D_m is less than a suitable constant, D_{max}, say 3. This limit according to Cootes, satisfies almost all training examples when applied in Eq. [3.14]. As mentioned previously, **b** should lie within a hyperellipsoid about the origin. When **b** is updated in every iteration, plausible shapes may occur $(D_m > D_{max})$ so then, the point lies outside the ellipsoid. To correct this, **b** should be rescaled so that it lies somewhere around an allowed ellipsoidal volume using the equation:

$$b_k \rightarrow b_k \cdot \left(\frac{D_{max}}{D_m}\right) \ (k = 1, \dots, t) \tag{3.40}$$

3.6 Multi – Resolution Active Shape Models

Multi resolution active shape models are used to improve the efficiency and robustness of the classical ASM algorithm. This method begins by initialy searching the object in a coarse image and subsequently refining the location in a series of finer resolution images. As a result, the algorithm tends to be much faster and less likely to not converge.
For every training image, a Gaussian pyramid is built. The base image (level 0) is the original image. Then, the image in the next level is created by applying firstly a 5x5 Gaussian filter for the reduction of noise and after that doing sub sampling to obtain an image which has half the size of the original. Cootes in [30] suggests a filter that results from a convolution of two linear filters with values 1-5-8-5-1. Common methods used for resampling are bilinear or bicubic interpolation. Figure 3.15 shows a resampling of a face image in four levels.

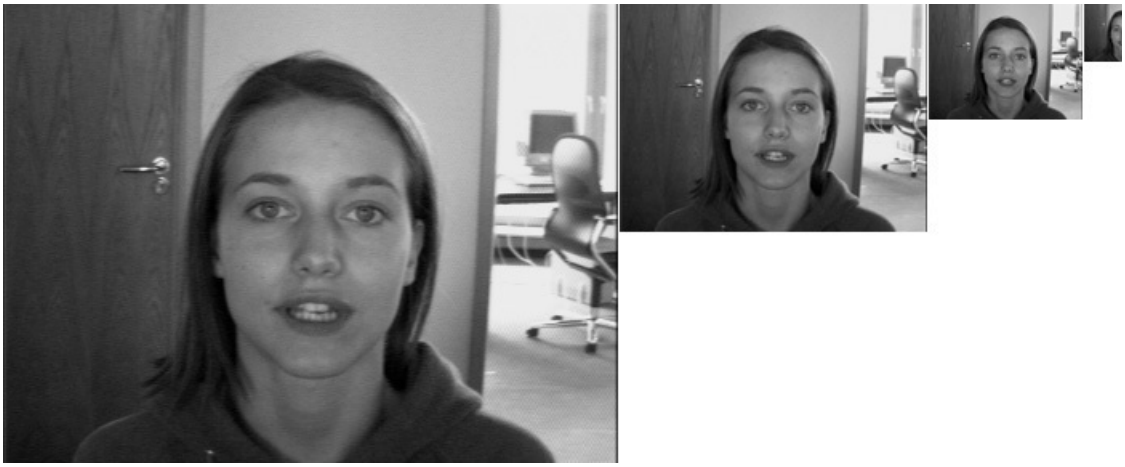


Figure 3.15: *Image Pyramid with 4 levels. The current level is equal to half of the image dimensions of the previous level (Milborrow, 2007).*

At the training stage, the statistical model of the gray level appearance is created along the normal to the points, for each of the image levels. Usually, the size of the profile normal is the same for each of the levels, so it makes sense that in the coarse image, information deduced is much more than the initial image (See Fig. [3.16]). In the coarse image, the movements of the points into a better position are much larger than the searching process in the initial image. Thus, reaching towards the search in the initial image, the movements are much smaller and the convergence time much less.

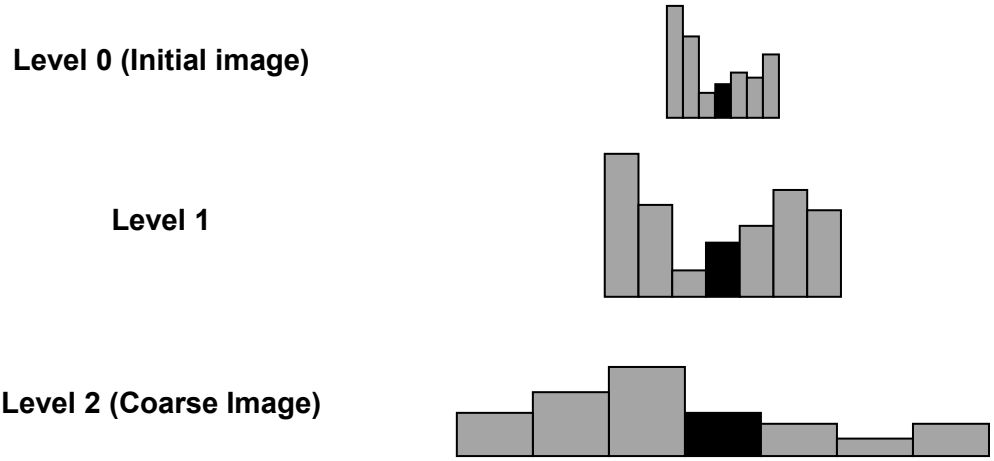


Figure 3.16: *Gray level appearance for a landmark point at different levels of the Gaussian pyramid (Self illustrated).*

One of the issues that emerge during the search process is to decide when the algorithm should converge for the existing level and move onto the next one. Cootes in [30] suggests that this could be achieved by recording the number of times the gray model has found its minimum (50%) within the central region of the profile. When a sufficient number of points (e.g. > 90%) produce a best fit within this central region (50%), then the algorithm is declared to have converged to that level (See Fig.[3.17]). Then, the current model is projected into the next image level and runs to converge again. When the convergence process is reached at the finest resolution, the search stops (See Fig. [3.18]).

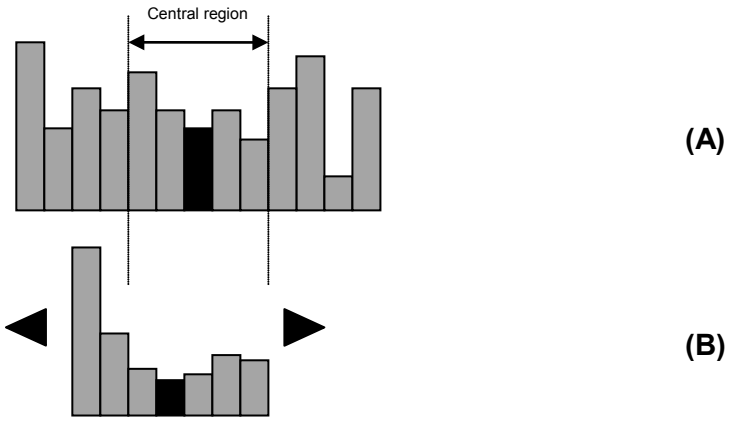


Figure 3.17: ***(A)*** *shows the profile sampled normal of the current pyramid level together with the central 50% of the positions* ***(B)*** *Grey level model of the current landmark point for the current pyramid level (Self illustrated).*

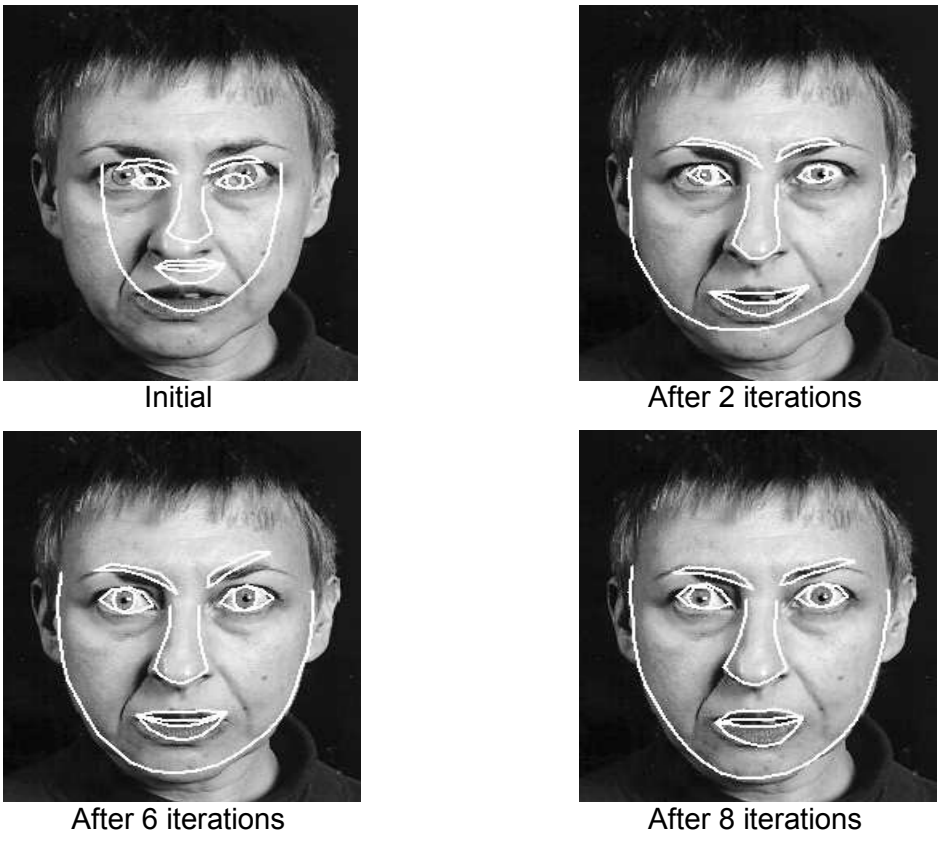


Figure 3.18: *Multi resolution ASM search on a face (T.F.Cootes & C.J.Taylor, 2004).*

3.7 The Viola - Jones detector

The Viola Jones algorithm is the first object detection algorithm proposed in 2001 by Paul Viola and Michael Jones [18]. Although it can be used (through training) to detect several objects, it was primary used to solve the problem of face detection.

The method is based on simple features (see Fig. [3.19]) which are applied not on the original image but on what is called derivative image. One of the main reasons of using this method is that the feature based system operates much master than a pixel – based system.
The Viola – Jones algorithm is restricted to three different kind of features. The first one is a *two - rectangle* feature whose value is the difference between the sum of pixels of two rectangular features. The regions have the same size and shape and are horizontally or vertically adjacent. Then, a *three – rectangle* feature, computes the sum within two outside rectangles, subtracted from the sum of the centre rectangle. Last but not least, a *four – rectangle* feature calculates the difference between the diagonal parts of the rectangles. According to the figure below, every feature is enclosed within a 24x24 detector, which means that a very large set of rectangles could be used, around 180,000.

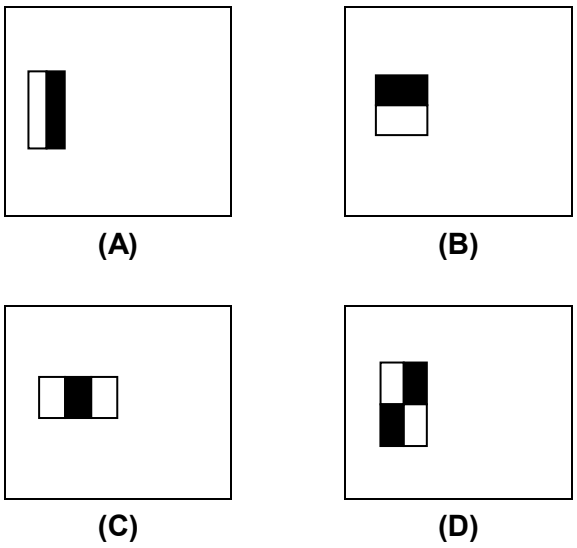


Figure 3.19: *Example rectangle features shown relative to the detecting window. Two rectangle features are shown in ((A),(B)), (C) shows a three rectangle feature and (D) a four rectangle feature (Viola & Jones, 2001).*

Coming back to the definition of an integral image, any location x, y is equal to the sum of the pixels above and to the left of x, y and its expressed mathematically as:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{3.41}$$

where $ii(x,y)$ is the integral image and $i(x', y')$ the original image.

The above formula could be explained with the following example: Consider four rectangles A,B,C,D. The sum of pixels within rectangle D can be computed by using all four points (Fig. [3.20]). The value that the integral image will have at location 1 is the sum of pixels in rectangle A. Then the value at point 2 is equal to the sum of pixels in rectangles A and B. Subsequently, the value at point 3 is equal to the sum of pixels in A, B and C. Last but not least the value at point 4 is equal to the sum of pixels in A, B, C and D.

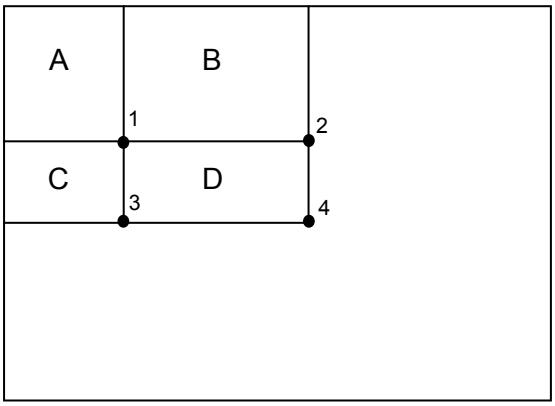


Figure 3.20: *The sum of the pixels within rectangle D is computed using all three rectangles (A,B,C) (Viola & Jones, 2001).*

The classifier is created by using a machine learning algorithm known as Adaptive Boost algorithm (AdaBoost) that selects from a set of features and a training set of positive and negative images, a small number of features used to train the classifier.
Originally, this AdaBoost algorithm was used increase or to boost the classification performance of a weak learning algorithm. As mentioned previously, there are over than 180,000 rectangle features applied within a sub window. Thus, although calculating every feature is doable and quite effective, the same process for all features is quite expensive. So the problem is how to find these features.
AdaBoost combines a collection of weak classifiers to create a stronger classifier. A weak classifier in this algorithm is considered being a feature.

A weak classifier $h_j(x)$ consists of a feature f_j, a threshold $\theta_j$and a parity p_j indicating the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1, & if\ p_j f_j < p_j \theta_j \\ 0, & otherwise \end{cases} \tag{3.42}$$

with x being a 24×24 pixel sub – window.

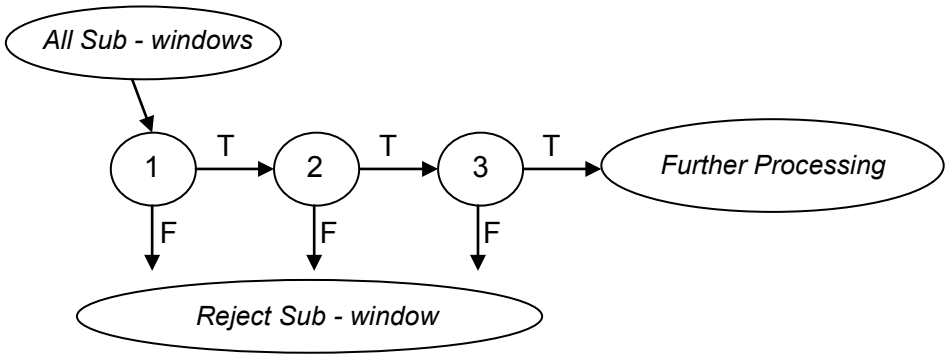


Figure 3.21: *Schematic description of a detection cascade (Viola & Jones, 2001).*

Furthermore Viola and Jones introduced an *Attentional cascade* which creates a cascade of classifiers for increasing detection performance and reducing computation time. It works by creating smaller classifiers which when applied, reject most of the negative sub – windows and keeps all positive. The meaning of the word "cascade" indicates that the process of detection has a tree search structure.

A positive result from the first classifier kicks in the second classifier and so on. A negative result would lead to rejection. The process is done by training classifiers using the AdaBoost algorithm and then setting a threshold to minimize the false negatives.
Stages in the cascade are created by training the classifiers using AdaBoost algorithm and then setting the threshold to minimize false negatives (different threshold in every scale). Generally by default, AdaBoost algorithm generates a low rate error on the training data. The image below contains front faces of many students during a school trip to Italy. Applying the Viola – Jones algorithm on the image all of the faces that have a front view are detected.

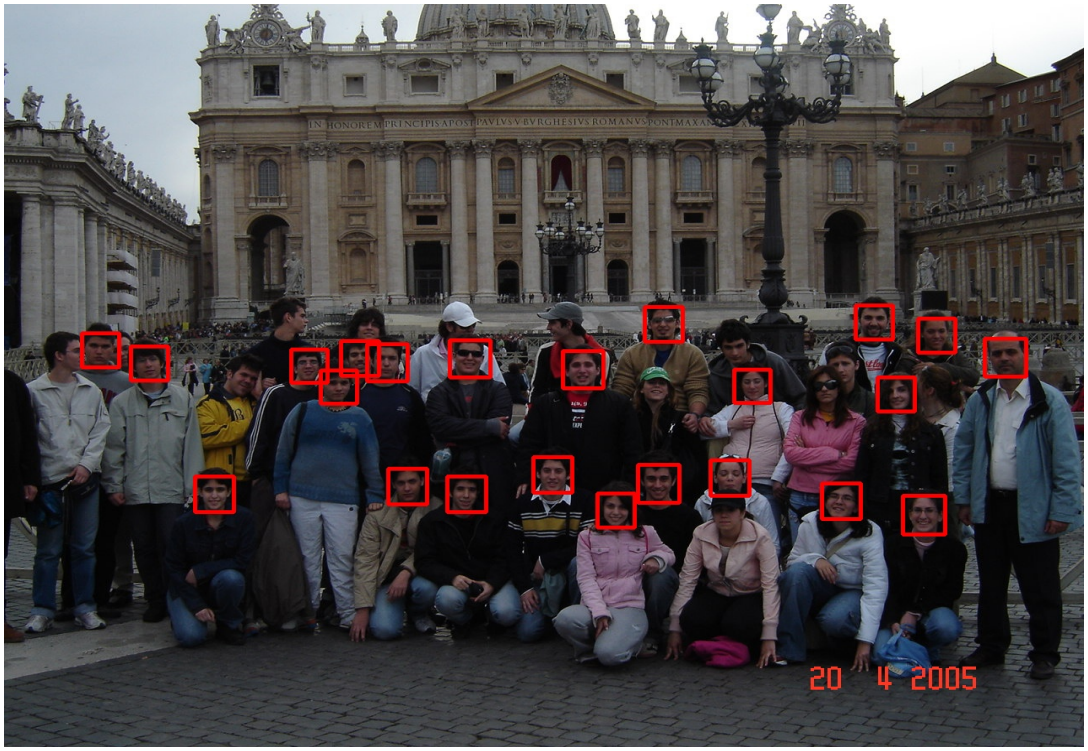


Figure 3.22: *Result of the Viola Jones algorithm on an image with multiple faces. The faces detected are only frontal faces but not profile faces. That's because the classifier was trained from front face images (Personal picture library, School Trip to Rome, Italy, 2005).*

4

Software

This chapter provides a detailed description of the functions written for the purpose of this master thesis. Within this framework, the implementation of the Viola / Jones algorithm for providing initial position to the face model is discussed. The input parameters, structures and classes are explained in Appendix A. The implementation was written in C++, using Microsoft Visual C++ 2010 Express compiler and OpenCV library. The source code was developed on a SonyVaio, VPC - Z1 model, i5 CPU at 2.40 GHz, 64 bit system with 4 GB RAM.

4.1 Functions definition

- *CvScalar **aligning_pair_of_shape** (CvMat *mat1, CvMat *mat2, vector<double> *weights, CvMat *aligned_data);*

Description: This function aligns a face shape to another reference shape based on the algorithm described in paragraph 3.3.2, graph 3.1.

Input:

mat1 : The coordinates of the first shape
mat2 : The coordinates of the second shape
weights : The weights for each one of the points

Output:

aligned_data : The aligned shape

- *void **alignTrainingData** (vector <CvMat*> *training_data, CvMat *mean_alighned_shape, vector <CvMat*> *aligned_set);*

Description: This function aligns a number of training images.

Input:

training_data : A vector containing the landmark coordinates of all training images.

Output:

mean_aligned_shape : The mean aligned shape or the Allowable Mean Domain.
aligned_set : The aligned normalised shapes.

- *void **mean_shape** (vector<CvMat*> *aligned_shapes, CvMat *mean_aligned_shape);*

Description: This function computes the mean shape from a set of aligned shapes.

Input:

aligned_shapes : A vector containing the landmark coordinates points of all training images.

Output:

mean_aligned_shape : The mean shape.

- *void **centerText** (char* s);*

Description: This function centres and prints out some words, sentences or data that the user will give as an input argument.

Input \ Output:

aligned_shapes : A string given as an input in the function, printed in the system's output window.

- *void **print_modes** (CvMat *EVal, CvScalar *eigenval_sum, vector<double> *modes);*

Description: This function creates a print table with three columns and indicates the most significant eigenvalues. First column contains the indices of the eigenvalues, second column their values and last column the percentage from the total variation each eigenvalue covers.

Input \ Output:

EVal : The eigenvalues of the covariance matrix derived from the aligned data.
eigenval_sum : The sum of the eigenvalues.
modes : Number of modes of variation and the equivalent most significant eigenvalues.

- *void **GetGrayValue** (IplImage *Image, CvMat *interval_X, CvMat *interval_Y, CvMat *Intensities);*

Description: This function gets the intensities of the interpolated points on the profile normal.

Input:

Image : The imported image

interval_X : The linear interval in x – direction across the normal
interval_Y : The linear interval in y – direction across the normal

Output:
Intensities : The gray intensities for each of the interpolated points

- *void* ***check_signed_zeros*** *(CvMat *M);*

Description: In C++ the zero value could be either positive or negative. Therefore, in order to avoid having problems with some calculations, this function checks the matrix for any negative zeros and sets them to positive zero.

Input / Output : The matrix to be checked. The function returns the same matrix but with just positive zeros

- *void* ***get_gray_level_appearance*** *(int nScales, vector<CvMat*> *matrices, vector<string>*dir_images,CvScalar nsamples, PyramidLevel *LabelPointsAppearance);*

Description: This function builds the gray level appearance around each landmark point and for each image level.

Input:
nScales : The number of image levels
matrices : The landmark coordinates of the non aligned training images
dir_images : The path where the images are kept.
nsamples : Number of interpolated points defined on the profile normal (only for the one side of the profile normal).

Output:
LabelPointsAppearance : The gray level appearance of every landmark point, in each image level, defined on the profile normal.

- *void* ***GetInitialPosition*** *(IplImage *image, CvMat *meanASD, CvScalar *InitialPosition, CvMat *InitialPos);*

Description: This function initially runs the Viola/Jones algorithm to detect the face in the search image and then tries to compute the initial pose parameters that will place the model within the image.
Based on figure 4.1, two different cascades are used to approximate the initial position. The first one is for frontal upright faces and the other for the left eye. The left eye detector is used to shift the t_y parameter downwards, because taking as translation parameters the centre of the frontal detector, it would place the face model quite high. Therefore, when attempting to bring the face model much closer to where the search face is, the centre of the eye detector was used. Based on figure 4.1, the formulas derived for the approximation of the translation parameters are:

For the $\mathbf{t_x}$ direction:

$$t_x = X_2 - X_1 \tag{4.1}$$

the $\mathbf{t_y}$ is equal too:

$$t_y = Y_1 + \frac{(Y_4 - Y_1)}{2} + \frac{(Y_{4'} - Y_{1'})}{2} \tag{4.2}$$

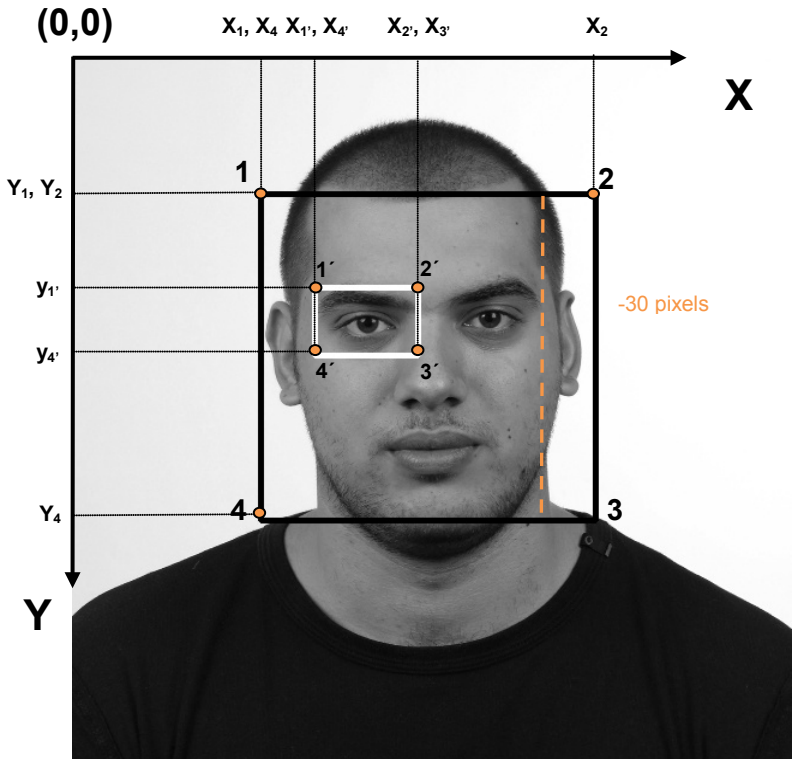


Figure 4.1: *Mathematically extracting the initial position*

The rotation $\boldsymbol{\theta}$ was set to a fix value of 2^o because most of the faces have approximately the same orientation.

For the scaling factor ***s*** the following simplified algorithm was used:

Input: *t_x, t_y, θ, coords_local_system, coords_image_system;*
Output: *scale (s)*
Initialise: *s = 30, scale increment = 0.01;*

for (infinite loop)
    *coords_image_system = T(t_x, t_y, θ, s) * coords_local_system;*
    Get_X_values = coords_image_system(:,1); // (n x 1) dimensions
    If *(Get_X_values > X_2 – 30)* ***break****;*
    s = s + scale_increment;
end

Algorithm 4.1: *Scale definition*

Based on figure 4.2 and algorithm 4.1, the face model is produced initially in a scale of 30 and then increased by a 0.01 step till the x value of the shape gets larger then X_2 – 30pixels. The offset of the detector window inwards by 30 pixels was defined empirically using different images. In most cases, the rectangle drawn outside the face was approximately around 30 to 50 pixels away. Figure 4.2 shows the result of the face model position and the initial pose parameters are:

TABLE 4.1
Approximate Initial Values

X_0	Y_0	Theta (θ)	Scale (s)
296.0000	283.0000	2	56.4959

Figure 4.2: *Initial position of the face model*

Input:

Image : The searching image
meanASD : The mean shape of the Allowable Shape Domain
InitialPosition : The pose parameters used to take the model from its local coordinate system to the image system

***Output*:**

InitialPos : The initial position of the model in the image coordinate system

- *void* ***get_position*** *(CvMat *model_shape, CvScalar *pose_shape_param, CvMat *pos_coords);*

Description: This function places the face model from its local normalised coordinate system to the images system (i, j).

Input:

model_shape : The face model in its local normalised coordinate system
pose_shape_param : The pose parameters (translation, rotation and scaling)

Output:

pos_coords : The model's position in the image system

- *CvMat* ***get_model_parameters*** *(CvMat *shape_coords, CvMat *adjustments, CvScalar *pose_parameters, CvScalar *updated_pose_parameters, CvMat *P);*

Description: This function computes the shape parameters $b = (b_1, b_2, \dots, b_t)^T$ of the first t modes of variation, as a result of deforming the face model during the update of its points towards a better position in the image.

Input:

shape_coords : Coordinates of the face shape model in its local normalised coordinate system
adjustments : The adjustments of the current model shape to a new updated shape
pose_parameters : The pose parameters before the shape is updated
updated_pose_parameters : The new updated pose parameters after approximating the current shape to a new updated shape
P : The first *t* eigenvectors

Output: The function returns the shape parameter vector

- *void* ***check_shape_parameters*** *(CvMat *eigen_values, CvMat *b);*

Description: This function checks if the shape parameters are within the Allowable Shape Domain and if not, hard limits are applied to place the parameters on the border of the domain.

Input:

Eigen_values : The total number of eigenvalues
b : The model parameter vector

Output:

b : Same matrix *b* is given as an output but corrected whenever is needed

- *CvMat* ***Apply_Model*** *(IplImage *searching_image, int nScales, CvMat *x_current, CvMat *model_position, PCAParam *Statistics, CvScalar *num_model_samples, CvScalar *pose_parameters, int TImage_samples, PyramidLevel *LabelPointsAppearance);*

Description: This function applies the face model on the image and starts the iteration process.

Input:

searching_image : The search image
nScales : number of image scales used
x_current : The coordinates of the face model within its local coordinate system
model_position : The coordinates of the shape model in the image system
Statistics : It is a variable name of type *PCAParam* and it contains all the statistical information of the aligned training data. That is the eigenvalues, eigenvectors, mean aligned shape, number of modes, generated new shapes
num_model_samples : number of interpolated points on the profile normal of the model points placed in the searching image (the number concerns only one side of the profile)
pose_parameteres : Initial pose parameters used to place the face model in the image system
TImage_Samples : Number of interpolated points on the sample profile
LabelPointsAppearance : The gray level appearance of the landmark points in every image level

Output: A matrix containing the final coordinates of the face model, fitted as best as possible on the face in the image

- *void* ***read_txt_face_parts*** *(string path, vector<string> *parts_name, vector<string> *parts, vector<int> *minVal, vector<int> *maxVal);*

Description: This function reads the text file with the name ***face_parts.txt***. The figure below shows the content of this file:

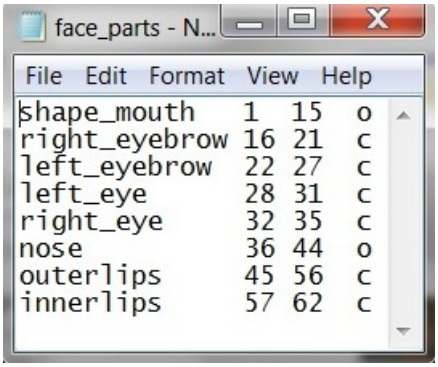


Figure 4.3: *text file of the parts of the face*

The file name (face_parts.txt) as well as the format of the file (fixed length) must not be altered. The reason for this is that the program classifies for every line the name of the face part, the minimum and maximum index of that part and also specifies if it's a close contour of not. Nevertheless, it makes sense to use this function when the range and index of landmark points per face part is known.
If a new data set of landmark points is given, where the user does not know which range of points corresponds to which part of the face, then it is not recommended to use this function. It is used to fix the normals to the face contour.

Input:

path : The path where the file *face_parts.txt* is located

Output:

parts_name : A vector containing the names of the face parts
parts : A vector containing the last column of the above text file which determines if the face part is close or open contour
nsamples : Number of interpolated points taken on the profile normal
minVal : Vector containing the starting index value of each parts of the face
maxVal : Vector containing the end index value of each part of the face

- *void* ***plot_aligned_data*** *(vector<CvMat*> *aligned_data, CvMat *mean_aligned_shape);*

Description: This function plots the aligned shapes and the mean shape on top of them.

Input:

aligned_data : A vector containing the coordinates of the aligned shapes
mean_aligned_shape : The mean normalised aligned shape

- *void* ***plot_variations*** *(vector<CvMat*> *variations);*

Description: This function plots different shapes produced by varying the most significant eigenvalues.

Input:

variations : A vector containing the landmark coordinates of the produced shapes

- *void* ***plot_object*** *(CvMat* variations, IplImage *search_image);*

Description: This function plots the final face shape model on top of the search image

Input:

variations : A matrix containing the final face model coordinates

Output:

search_image : The searching image with the face model fitted on top

- *void* ***InvMatMahalanobis*** *(CvMat *Covariance, CvMat *InvCov);*

Description: This function finds the inverse of the covariance matrix used in the computation of the Mahalanobis distance. Due to strong linearities between the interpolated points on the profile normal of a point, the covariance matrix is not positive definite and not full rank (at least most of the times). This could lead to inaccurate results given by the Mahalanobis distance and a zero value can only mean that the profile match is perfect.
This, according to some properties of the inversion of a matrix is not acceptable. Thus, a method used to solve this problem is as follows:

1. Assume that A is a problematic covariance matrix. Perform a spectral decomposition so that $A = Q\Lambda Q^T$ where Λ is a diagonal matrix containing the eigenvalues

2. Set all the small or negative elements of the Λ matrix to a very small positive number (in the project it was set to DBL_EPSILON) and create a new diagonal matrix Λ'

3. Reconstruct A from $A = Q\Lambda'\mathrm{Q}^{\mathrm{T}}$

The new covariance matrix A is now full rank and positive definite.

Input:

Covariance : The covariance matrix of the current landmark point

Output:

InvCov : The inverted covariance matrix

- *void* ***Calc_Covar_mean_mat*** *(vector<CvMat*> *obs_vector, int nsamples, CvMat *Covariance, CvMat *mean);*

Description: This function calculates the covariance matrix of every landmark point and also the mean normalised gray level intensity of the point taken as an average from all the training examples.

Input:

obs_vector : A vector of matrices containing the normalised intensities of a landmark point throughout all images

nsamples : Number of training examples

Output:

Covariance : The covariance matrix of a point
mean : A column matrix containing the mean normalised intensity of a point

- *void* ***PCA*** *(vector<CvMat*> *aligned_shapes, CvMat *aligned_mean_shape , PCAParam *ShapeStat);*

Description: This function applies Principal Component Analysis on the aligned training data.

Input:

aligned_shapes : A vector containing the aligned shapes
aligned_mean_shape : The mean aligned shape

Output:

ShapeStat : This is a variable of the class type ***PCAParam*** that contains all the statistical information concerning the aligned training data. These are the eigenvectors and eigenvalues of the covariance matrix derived from the aligned data, mean aligned shape, number of modes, most significant eigenvalues and new approximated shapes

- *void* ***convert_N2_to_N1*** *(CvMat *input_mat, CvMat *output_mat);*

Description: Converts the coordinates layout from a $n \times 2$ order to a $2n \times 1$ form.

Input:

input_mat : The given $n \times 2$ matrix

Output:

output_mat : The resulting $2n \times 1$ matrix

- *void* ***convert_N1_to_N2*** *(CvMat *input_mat, CvMat *output_mat);*

Description: Converts the coordinates layout from a $2n \times 1$ order to a $n \times 2$ form.

Input:

input_mat : The given $2n \times 1$ matrix

Output:

output_mat : The resulting $n \times 2$ matrix

- *void* ***convert*** *(string input_file, CvMat *ma);*

Description: This function converts the initial coordinates in text format to a *CvMat* form.

Input:

input_file : A string specifying the path to the text file where the coordinates of the current example image are saved.

Output:

ma : The coordinates of the training image converted into a *CvMat* form.

- *void* ***dist*** *(CvMat *m, CvMat *output)*

Description: This function computes the Euclidean distances of a point with respect to all other points in the training image.

Input:

m : A matrix containing the landmark coordinates of the shape in a $n \times 2$ form

Output:

output : A matrix $(n-1) \times n$ dimensions, where each column corresponds to the distances of a point to all other points

- *double* ***var_of_pts*** *(CvMat *pts)*

Description: This function computes the variance of every point using the formula in Eq. [3.5].

Input:

pts : A row vector containing the distance of the current point to its next point in all training images

Output: The variance of this distance over the training set

- *void* ***compute_Weights*** *(vector <CvMat*> *training_data, vector <double> *weights)*

Description: This function computes the weights of each landmark point in the training set using Eq. [3.5].

Input:

training_data : A vector containing the landmark coordinates for each one of the training images

Output:

weights : A vector with the weights of each landmark point

- *CvMat* ***DiagMatrixWeights*** *(CvMat *arg)*

Description: This function creates a diagonal matrix $(2 \times n,\ 2 \times n)$ with the diagonal elements being the point weights.

Input:

arg : A matrix $(n \times 1)$ containing the weights of the points

Output: A weighted diagonal matrix

- *void* ***COG*** *(vector<CvMat*> *matrices, vector<CvMat*> *cog)*

Description: This function translates all the training images to their centre of gravity. This is considered being the first part of the normalization process.

Input:

matrices : A vector containing the landmark coordinates for each one of the training set

Output:

cog : A vector containing the translated landmark coordinates of each training images

---

- *void* ***scaling*** *(CvMat *InitialMeanShape, CvMat *MeanScaleShape)*

Description: This function scales the reference shape so that the mean distance between the centre (COG) of the shape and all other points is equal to $\sqrt{2}$ (second step of the normalization).

Input:

InitialMeanShape : The reference translated shape to be normalised

Output:

MeanScaleShape : The scaled reference shape

- *void* ***GrayLevelAppearance::GetContourNormals*** *(string type, CvMat *Contour_Points, CvMat *ContourNormals)*

Description: This function calculates the contour normals of the face shape. It is a general function which could also be used for calculating the normals of close or open contour objects, regardless from faces. The next function calls the current described function for calculating the normals for each part of the face separately.

Input:

type : A given character which is either “c” for close contour or “o” for open contour. The computational difference between these two is that in the close contour case, the direction of the whisker of the first point is defined from the difference of the second point and the previous last point of the contour. On the other hand, in case of an open contour, the whisker direction is defined only from the next point and the current point.

Output:

ContourNormals : A $n \times 2$ matrix containing the normals to the points in the current shape

---

- *void* ***GrayLevelAppearance::GetContourNormalsFace*** *(string path, CvMat *Contour_Points, CvMat *ContourNormals)*

Description: This function calculates the normals to the face’s contour. In is often the case where the range of points corresponding to each part of the face is unknown. The next point minus the previous point technique is not that “valid” for example in cases where the normal on the first point of the eyebrow is calculated using the last point of the outer face (compare results between figures [4.4a] and [4.4b]).

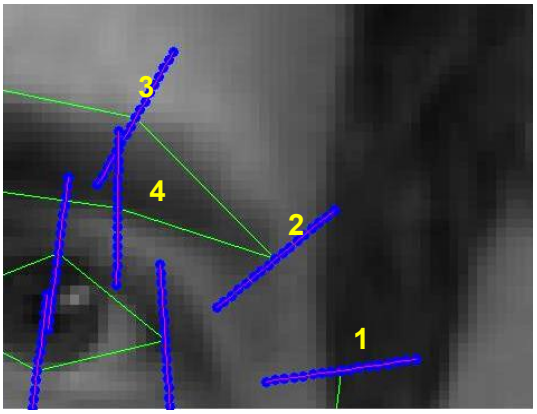


Figure 4.4a: *Point 2 whisker direction by using points 1 and 3*

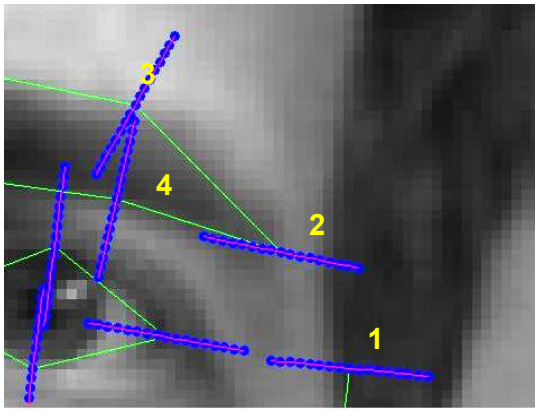


Figure 4.4b: *Point 2 whisker correction by using the points 3 and 4*

The above figures show how the direction of the whisker at point 2 is affected from the previous and next point. As you can see, the right eyebrow is considered being a close polygon, thus the next point of the current point 2 should be 3 and the previous 4. Taking this into account, the "corrected" direction of the whisker can be seen in Fig. [4.4b].

Input:

path : The path directory to the text file ***face_parts.txt***
Contour_Points : The landmark coordinates of the current image

Output:

ContourNormals : A $n \times 2$ matrix containing the normals of the face in the current image

- *void* ***GrayLevelAppearance::linear_intervals*** *(CvMat *Contour_Points, CvMat *ContourNormals, CvMat *intervalx, CvMat *intervaly, CvScalar *samples)*

Description: This function computes the linear intervals on the profile normal (whisker) with a step equal to one pixel.

Input:

Contour_Points : The landmark points in the current image
ContouNormals : The normals to the landmark points
samples : Numbers of samples taken on the whisker. The number of points should involve only one direction of the whisker. Thus, the total number of interpolated points will be equal to $2 \times n + 1$

Output:

intervalx : The linear intervals in the x - direction
intervaly : The linear intervals in the y - direction

- *void* ***GrayLevelAppearance::GetDerivatives*** *(IplImage *src_image, CvMat *intervalx, CvMat *intervaly, CvMat *derivatives_norm, double samples)*

Description: This function calculates the derivatives and normalised derivatives of the gray level appearance around each landmark point based on chapter 3, paragraph 3.4.3.

Input:

src_image : The input image from which the gray intensities of the profile normals will be derived
intervalx : The linear intervals in the x - direction
intervaly : The linear intervals in the y - direction
samples : Numbers of samples taken on the whisker. The number of points should involve only one direction of the whisker. Thus, the total number of interpolated points will be equal to $2 \times n + 1$

Output:
derivatives_norm : A $(2 \times n + 1) \times number\ of\ label\ points$ matrix containing the normalised derivatives. Each column corresponds to one landmark point

- *void* ***CheckifImageGrayScale*** *(vector <string> *training_images)*

Description: This function converts the 24bit training images to 8bit gray scale images in case they were not inserted in this form initially.

Input / Output:
training_images : A vector containing the names of the training images. The output of this function is the convention of the RGB images into gray scale images.

- *double* ***me17*** *(CvMat *LandmarkPoints, CvMat *searchPoints)*

Description: This function computes the me17 error described in paragraph 5.1.

Input:
LandmarkPoints : A $n \times 2$ matrix containing the fixed landmark coordinates of the face

searchPoints : A $n \times 2$ matrix containing the resulting coordinates of the face model emanated either from the initial placement of the face model in the image or after the convergence of the model.

Output:
The me17 value of the face fit (in pixels) between the fix landmark points and the points coming from the face detector or the ASM search result.

- *vector<string>* ***GetPyramidImages*** *(int nScales, vector<string> *TrainingImages)*

Description: This function produces the image pyramid of the training set.

Input:
nScales : Number of pyramid scales to be created
searchPoints : A vector containing pointers to the training images in their initial dimensions.

Output: A vector containing pointers to the pyramid images.

5

Results

This chapter presents the experimental part of this master thesis. It starts by introducing the method used to measure the quality of the model fit in the image, known as ***me17***. Then, the initial placement of a face model on a set of images selected randomly from the MUCT dataset, applying the Viola / Jones face detector and the quality of fit is seen. Subsequently, using Cootes online data set (training set) provided in sub section 5.4.1, the process of alignment, capturing the statistics of the aligned data and applying the face model on a coarse search image is explained. Moreover, a comparison will be shown of the different ***me17*** fits using face models derived from unmodified and modified raining images (adding Gaussian noise and dislocating the landmark points).

5.1 The me17 measure

Christinacce in [4] introduced the ***me17***[2] measurement, designating the quality of fit of a face model in an image, described by the following formulation:

$$m_e = \frac{1}{ns}\sum_{i=1}^{i=n} d_i \tag{5.1}$$

where:

d_i are the point to point Euclidian distances,
s is the distance between the eye pupils and
n is the number of landmark points used to describe this error. The result is in pixel units.

[2] The reason it is given the name ***me17*** is because 17 points are used to estimate the overall fit.

Keep in mind that this measurement is ambiguous due to the fact that it covers some of the inner part and none of the outer part of the face. This, in some cases could lead to a somehow unreliable result, because the inner part might not have converged as well as the outer part and vice versa. Figure [5.1] illustrates the relation between the *me17* points of the face model respectively to the equivalent ones in the face image.

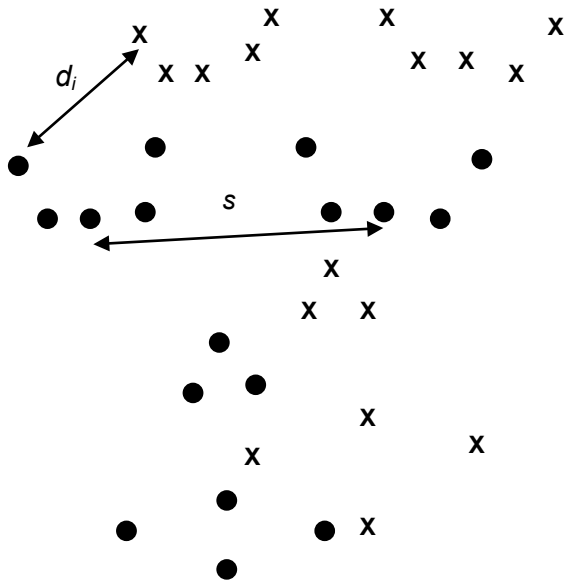


Figure 5.1: *The symbol "●" indicates true location and "**x**" predicted location*

Figure [5.2a] shows the position of the 17 points in the image and figure [5.2b] the index of these points.

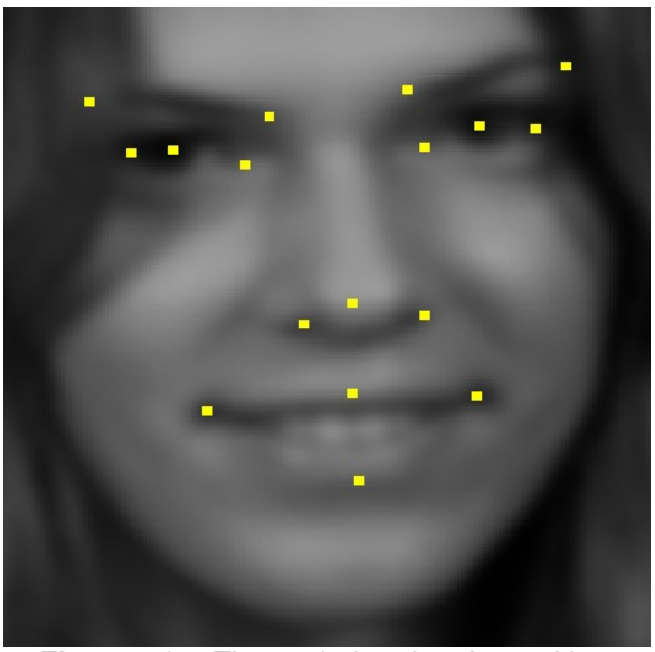


Figure 5.2a: *The me17 Landmarks position [Milborrow]*

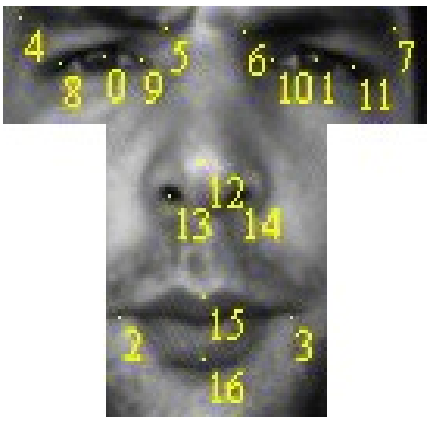


Figure 5.2b: *The me17 Landmark index [BioID set]*

5.2 Face Configuration of the MUCT dataset

The face images, provided by the online database, were captured from five different cameras with a configuration shown below:

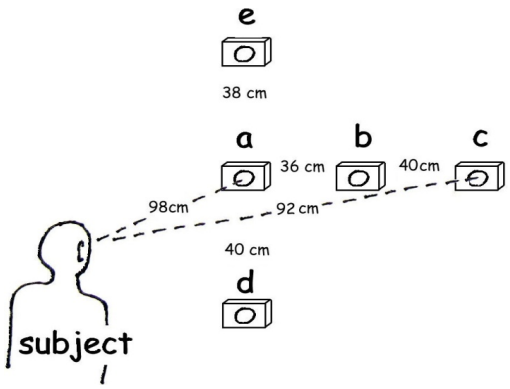


Figure 5.3: *The five cameras and their position with respect to the face*

The utilization of the training data emanated only from camera **a**. Specifically, cameras **b** and **c** were not good enough to be used due to obscured landmarks, thus they were excluded. The images have a resolution of 640 x 480. As you can see from the above figure, camera was not located to the left since it could be approximated by mirroring the image from the right. In total, 76 landmark points where measured manually in each image from a human landmarker and then checked from a third person. The image filenames have the form "***i000qa-fn.jpg***" where:

- ***i*** is a prefix in all images
- ***000*** is the ID increment
- ***q*** is the lighting set (lookup table in http://www.milbo.org/muct/muct-details.html)
- ***a*** is the camera view (see Fig. [5.1])
- ***f*** stands for female and ***m*** for male
- ***n*** is for no glasses and ***g*** for glasses

For example, image "***i007qa-fn.jpg***" is the 7th image in the dataset, with lighting set *q*, captured from position *a,* gender female and wears no glasses.
For more technical information concerning the lighting configuration and the advantages of this dataset over some other datasets refer to [21].

5.3 Face detection on MUCT images

The Viola – Jones algorithm is robust enough to detect faces but it is also important to see how it could be used to approximate the position of a face model in the search image. This was established and explained in the previous chapter and here some experimental results are provided.
Six upright frontal faces where chosen randomly from the MUCT database, each one with different external characteristics. These different characteristics could be distinguished between people with or without hair, wearing glasses or not, having a beard or not etc. In Fig. [5.4] the detector has situated the face model in a very good initial position.
All parts of the face model are placed very close to the edges of the object. On the other hand, the position of the eyebrow, eye and nose in Fig. [5.5] are placed quite close to the face image, but the outline of the face is further away from it.

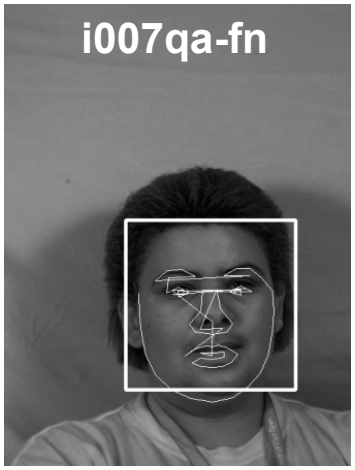


Figure 5.4

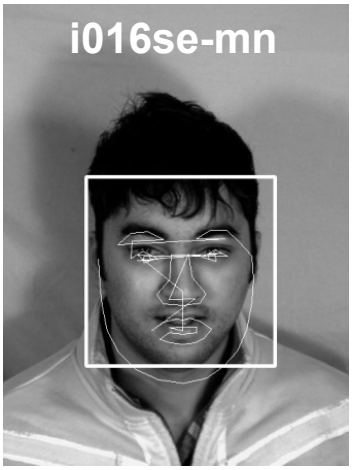


Figure 5.5

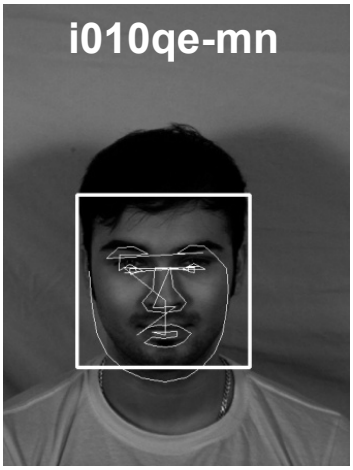


Figure 5.6

In figure [5.6] the face model is been placed quite well, similar to figure [5.4]. Nevertheless, using this image as a search image could result a bad fit because of the constant gray intensities around the beard. However, training the gray level appearance using images with similar complexities and beard presence, the face model could most probably converge. Despite this comment, the initial approximation is very good.

Continuing with Fig. [5.7], the face model is well placed on the outer part of the face, nose and lips but the eyebrows and eyes are situated further down from where they should be. Regardless of this problem, the particular face couldn't be used as a search image since the gray level appearance of the training faces does not contain any headscarf information (or very limited) and secondly because the area around the face has a constant intensity, hence no better positions could be estimated.

In figure [5.8] the face detector works quite well but the eye detector fails to provide a good initial position of the face model. As a result, the eye detector recognises the mouth part which automatically shifts the face model further down. Figure [5.9] illustrates this.

Last but not least, in Fig. [5.10] the face model is placed quite well in most parts except the eye part.

However, that wouldn't be a problem because the ASM method runs in a multiresolution approach which means that for a suitable profile length the eye part could be shifted to a better position.

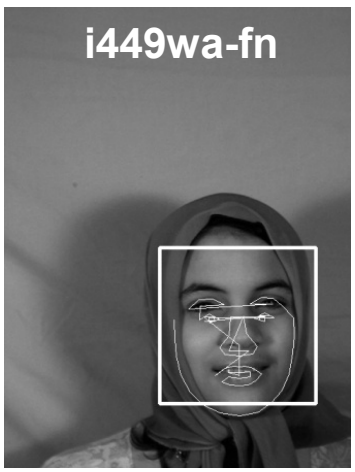


Figure 5.7

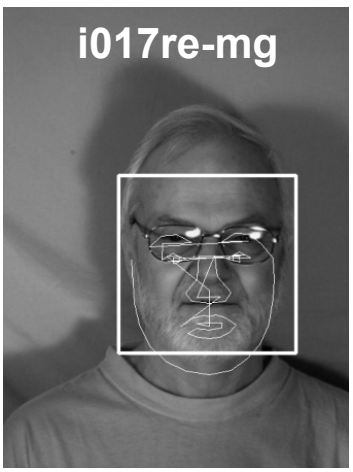


Figure 5.8

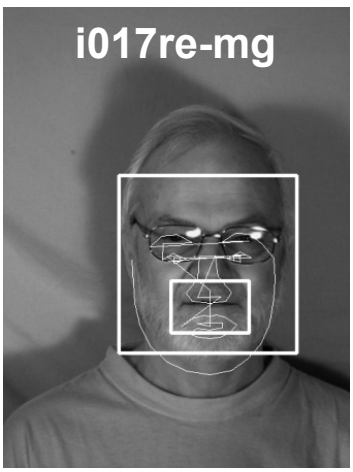


Figure 5.9

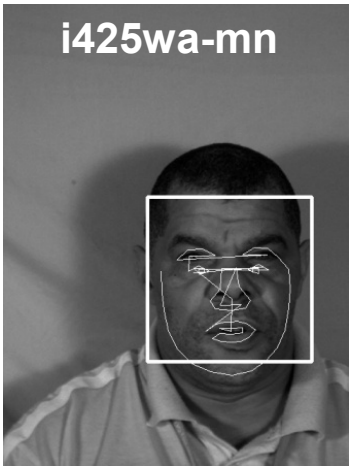


Figure 5.10

The chart below shows the ***me17*** measurement for all six images. It is clear that the best initial position is located in the third image with a ***me17*** of 0.084. Keep in mind that the ***me17*** result in all cases is affected from the position of the eye pupils. This means that if the rest of the model is placed quite well but the eye pupils are far away from the correct position, then the ***me17*** measure will give an ambiguous result. Taking a close look at the ***me17*** result of the *i017re-mg* image, the error is quite high due to large offset the eyes of the face model have from the real eyes position. At the same time, having this large offset in the inner part of the face, the outer part fit is quite good.

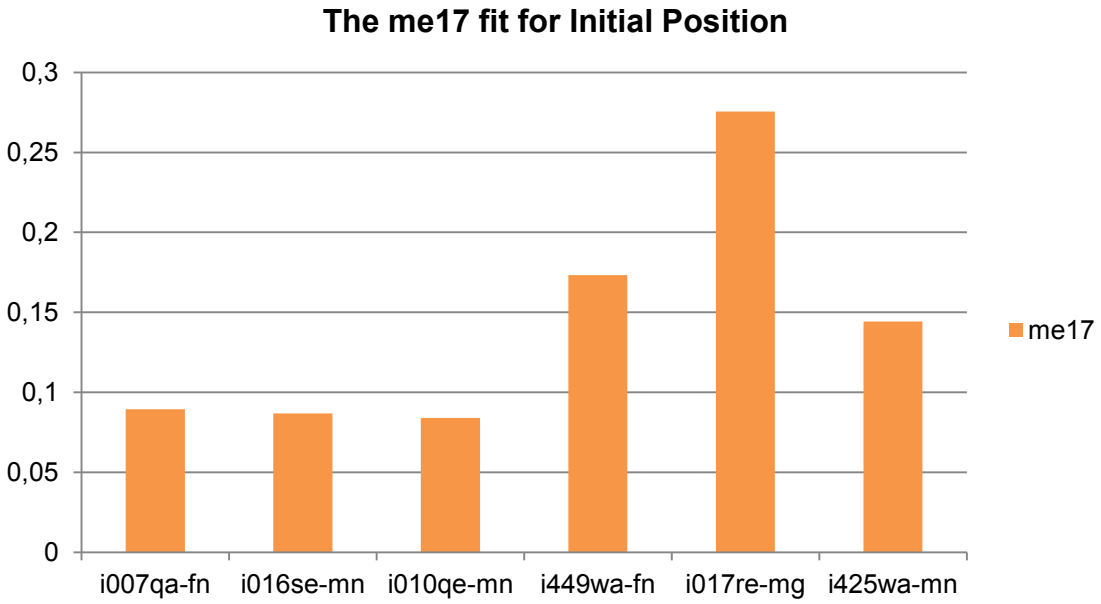


Figure 5.11: *The me17 fit for initial face model position*

On the other hand, the ***me17*** fit measure is quite good in the *i016se-mn* image due to the eye position, but the external part does not fit as well.
Overall, in the first three images, the error is kept in low levels because of the very good eye position and at the same time the outer part is not that good. On the other hand, the error is large in the rest of the images due to bad eye position but the outer part is good. The table below gives the initial pose parameters and the ***me17*** quality fit for each of the aforementioned images.

Table 5.1
Initial Pose Parameters and the me17 fit

Images	Initial Pose Parameters				me17 fit
	X_t (pix)	Y_t (pix)	theta (deg)	scale	
i007qa-fn	285	440	2	45.3478	0.0895
i016se-mn	246	398	2	52.0467	0.0869
i010qe-mn	221	410	2	46.3776	0.0840
i449wa-fn	323	468	2	40.1987	0.1733
i017re-mg	283	392	2	47.9274	0.2756
i425wa-mn	314	408	2	43.2882	0.1442

5.4 Applying the Active Shape Model on Cootes data

5.4.1 The training data

Cootes online data set contains 24 images in different moods with 68 landmark points measured per image. The dimension of the images is 640x480 as shown below.

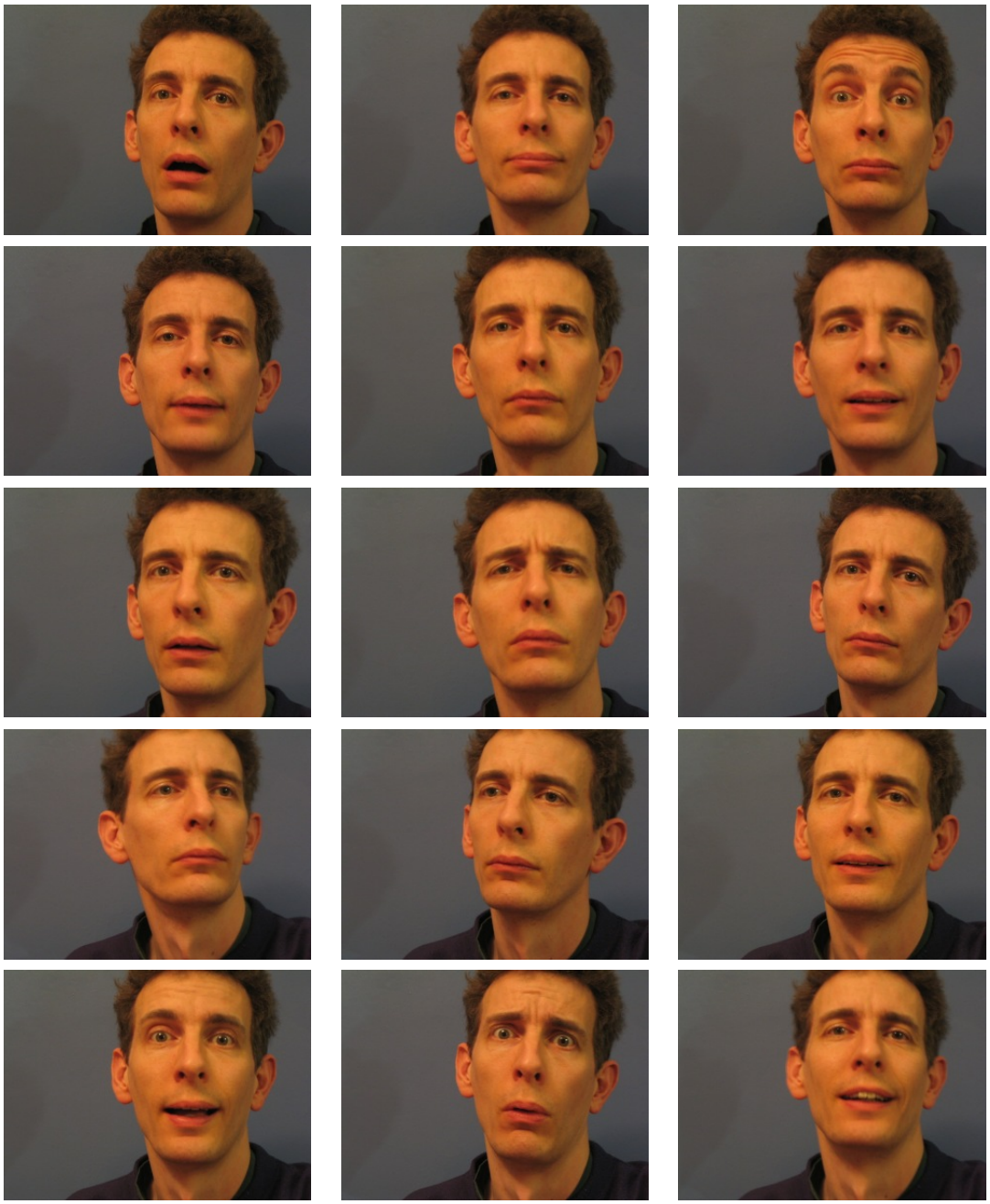


Figure 5.12: *Training set containing 24 images*

5.4.2 Aligning the training set

First part of the process involves the alignment of the training faces. Figure [5.13a] shows all the contour faces misaligned and then their alignment in figure [5.13b]. The algorithm used for the alignment process is referred in graph 3.1.
Figure [5.13a] shows a “cloud” of contour faces coming directly from the training set and represented with a green line. Figure [5.13b] on the other hand illustrates the aligned faces, with a red line on top representing the mean face. The alignment procedure converged after ***3 iterations***.

Figure 5.13a: *The training set before alignment*

Figure 5.13b: *The training set after the alignment*

5.4.3 Capturing the statistics of the aligned data

After the alignment, PCA was applied on the aligned faces and the results are given in the table below. For the calculation of the covariance matrix, corresponding eigenvectors and eigenvalues the algorithm in Appendix C was used. The reason for this is because the number of training samples is less than the coordinates (2*68points = 136points > 24 images).
Table 5.2 shows the most significant eigenvectors listed in descending order, covering 97% of the total variation of the training set. From all seven eigenvalues, only the first three express the greater part of the variation and the rest a significantly smaller part.

TABLE 5.2
Most significant Eigenvalues of the Covariance matrix derived from Cootes training set

Eigenvalue index	Eigenvalue	Total variance (%)
λ_1	0.4304	65.01
λ_2	0.0918	13.86
λ_3	0.0507	7.66
λ_4	0.0326	4.92
λ_5	0.0232	3.51
λ_6	0.0124	1.87
λ_7	0.0042	0.63

Figure [5.14] shows the model parameters, defined as $\sqrt{\lambda_i}$ respectively to their x square distribution values. It is clear that most of the variations are indicated in the first 7 model parameters and the rest are concentrated very close to each other.

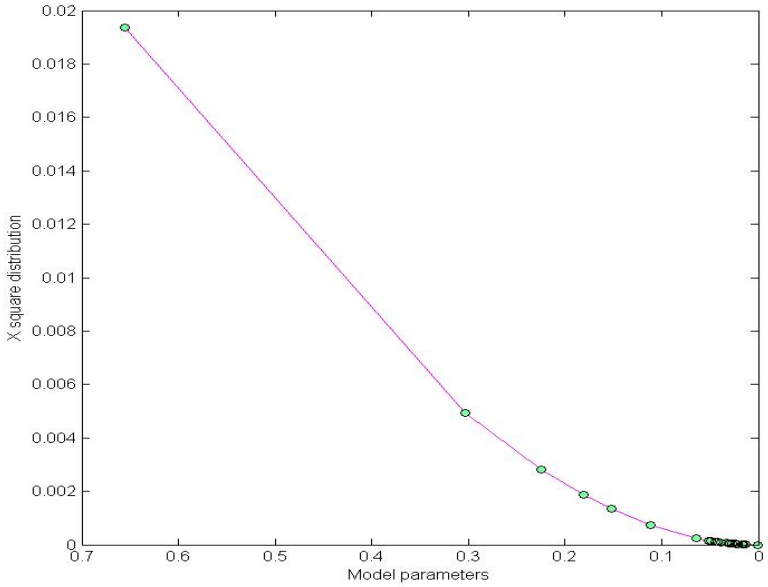


Figure 5.14: *The chi square distribution of the model parameters*

Figure [5.15] shows the first two principal components b_1, b_2 together with the aligned training examples, each represented with a single point. All these points lie, in what was defined in the previous chapter, as Allowable Shape Domain and the centre of this domain is the origin of the two most significant principal components. What is shown is that these two parameters could be considered being linearly independent due to the way the points are distributed within the domain. Dependency between these two parameters could result the generation of "illegal" shapes.

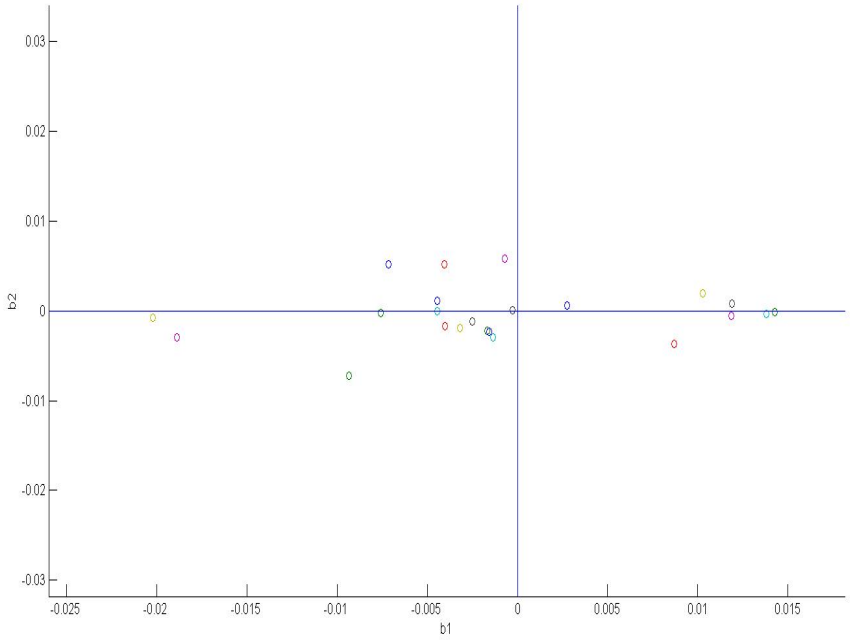
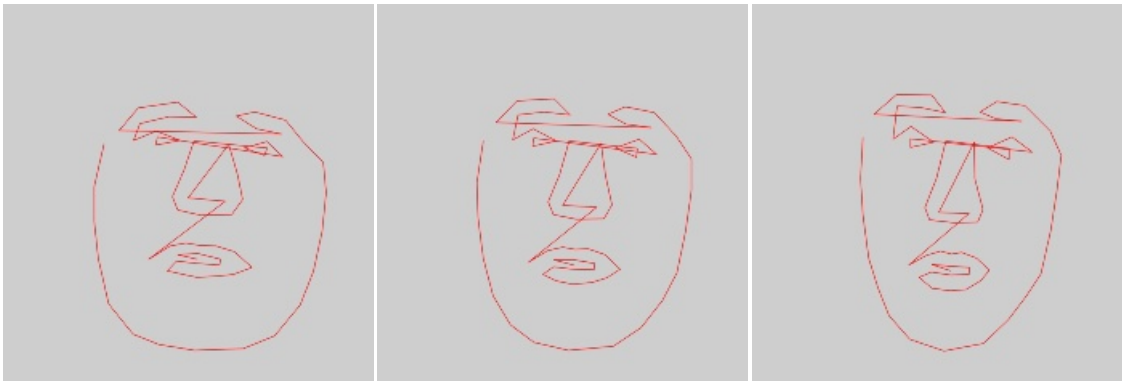


Figure 5.15: *Plotting b_1 against b_2*

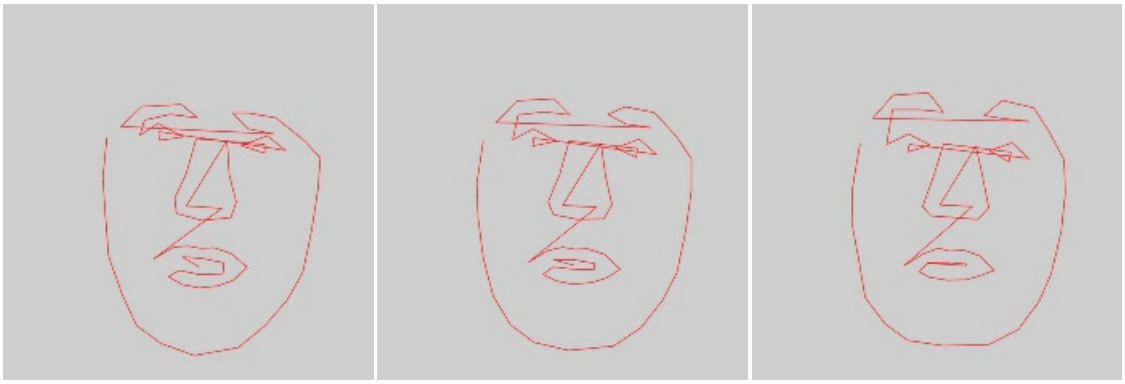
New shapes produced from Eq. [3.12] and [3.13] are shown below:

$-3\sqrt{0.4304}$ ⟵ 0 ⟶ $3\sqrt{0.4304}$



$-3\sqrt{0.0918}$ ⟵ 0 ⟶ $3\sqrt{0.0918}$

$-3\sqrt{0.0507}$ ⟵ 0 ⟶ $3\sqrt{0.0507}$



$-3\sqrt{0.0326}$ ⟵ 0 ⟶ $3\sqrt{0.0326}$

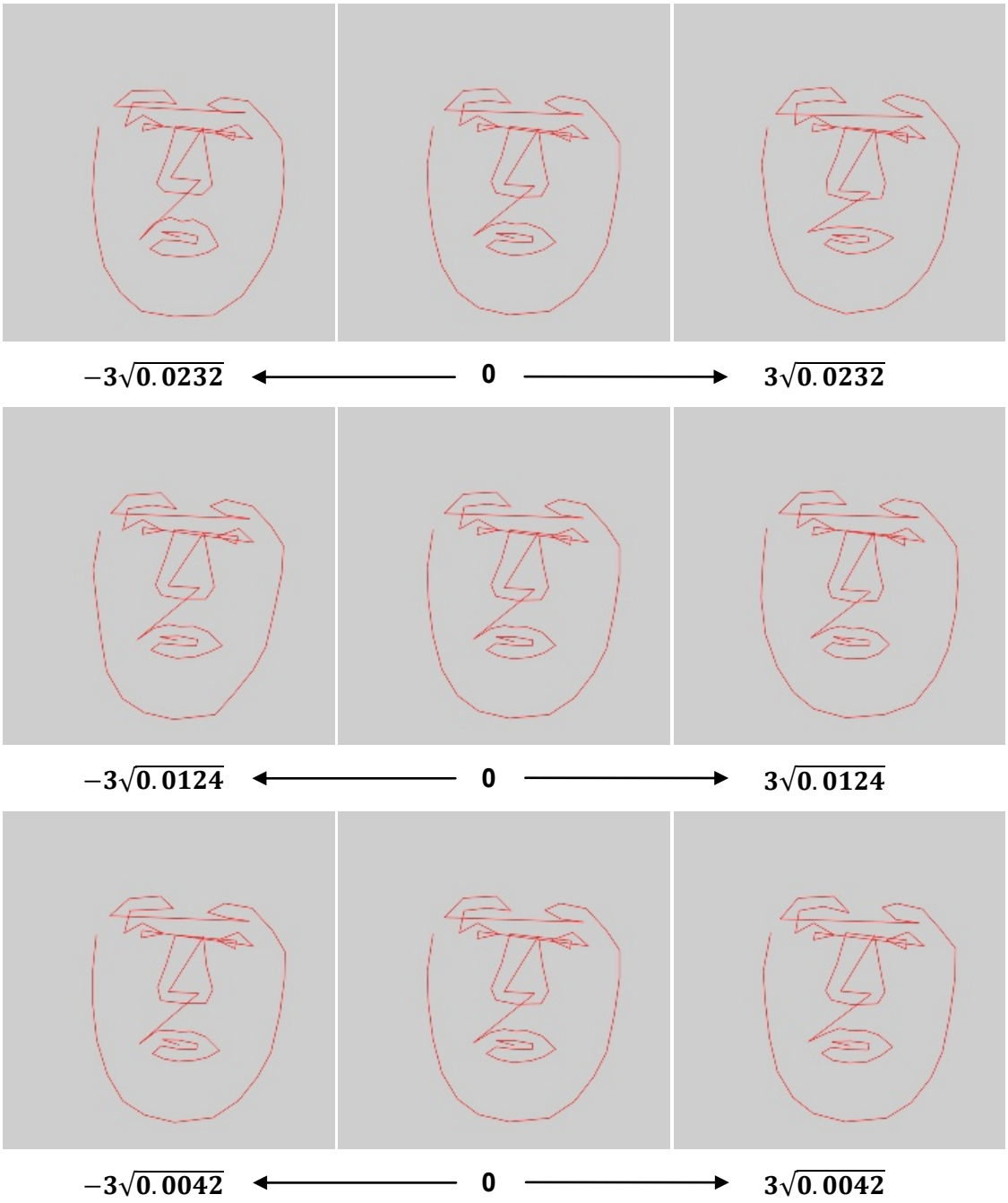


***Figure 5.16**: New face models build by using the most significan eigenvalues*

For each variation mode, three different shapes were created with a maximum variability of $\pm\lambda_i$. For the first mode, the generated shapes have a side rotation, capturing the variability of faces with a side view. The second mode of variation creates an elongated face, stretched outwards from its upper and lower part. It is clear that this mode captures the variation effecting the outer part of the face. Third and fourth modes vary in the mouth part, affecting the opening and closing of the mouth. The fifth mode stretches the mouth from both sides as it tries to create a flat smile. The sixth mode creates small movements of the jaw from left to right direction. Last but not least, the seventh mode effects the size of the lips by a small amount.

5.4.4 Detection and Searching

Last step of the procedure involves placing the derived face model from its local coordinate system to the image's system in order to start the iteration process towards face convergence. The images on the left column (below) show the result of the Viola – Jones face detector together with the initial placement of the face model. The results of the Active Shape Model approach are seen on the corresponding right column.

Image 1a

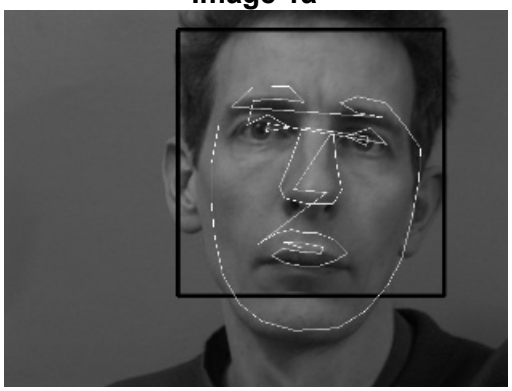


Image 1b

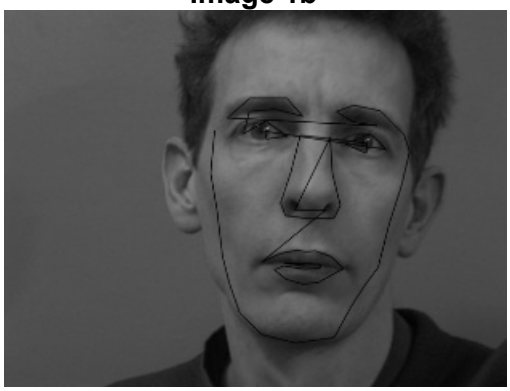


Image 2a

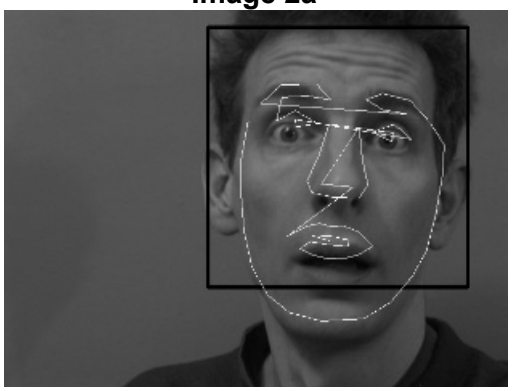


Image 2b

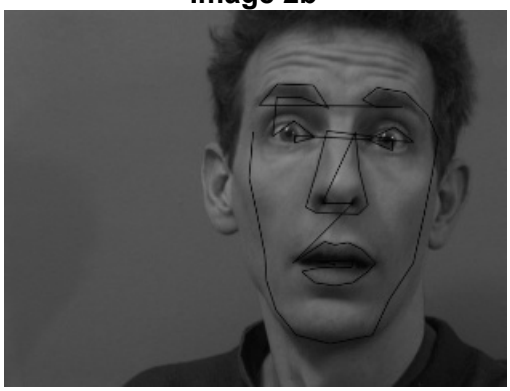


Image 3a

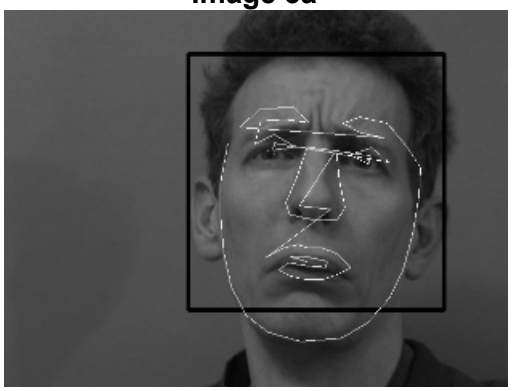


Image 3b

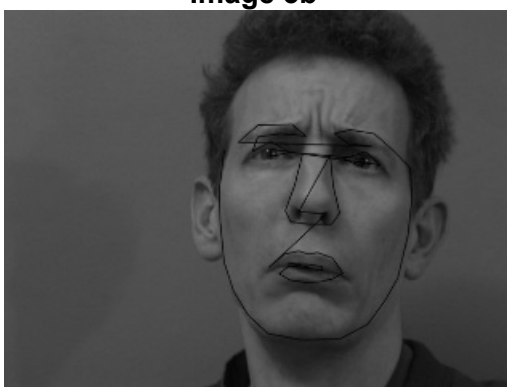


Image 4a

Image 4b

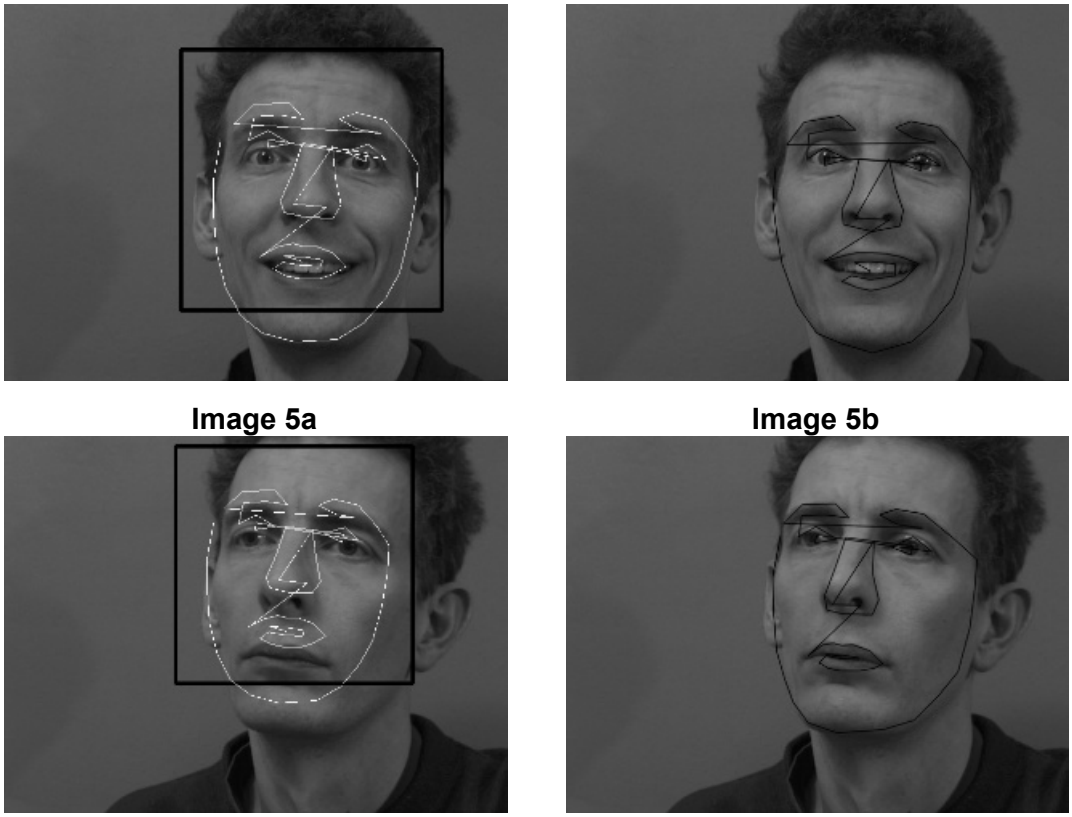


Image 5a **Image 5b**

Figure 5.17: *The images on the left show the initial position of the face model using the Viola Jones algorithm and on the right, the equivalent model fit.*

The input parameters required for the initialization searching process, were defined through testing and the values given are: model profile = 5; search profile = 9; 97% maximum described variation; 2 image scales. Theoretically, the initial position of the face model in all images is good enough not to require many iterations. Because 2 pyramid levels are used, when the model converges from the coarse image it should then be rescaled to fit the face in the initial image. The model would change image scale when 95% of all points lie within 50% around their central position. In case the model has not converged, a number of 400 iterations are set in the coarse level. The size of the Allowable Shape Domain, D_{max} was set to 0.8.

The results for the first and second image are quite good, regarding the inner part of the face. The outer part showed a lot of fluctuation and couldn't remain steady. What is intriguing is the fact that after numerous iterations the inner part wouldn't change, though the position of the points in the outer part fluctuated. Therefore, in order to check if there is enough room for improvements, all 400 iterations where executed.

For the second image, the model fit would improve until 180 iterations but after that it would start deviating rather than improving. Moreover, images 3, 4 and 5 have visually the best fit from the rest although the ***me17*** contradicts this view.

According to table 5.3, images 2 and 4 have the best fit, though this could be visually true for image 4 but not for image 2. Furthermore, while the inner part of images 1 and 2 has converged quite well, the outer part hasn't, though the me17 results are quite good. Generally, if no hard limits where applied on the size of the Allowable shape domain, the movement of the points would be very ambiguous and the degree of freedoms quite high. For image 3, the size of the domain had to be changed because with the value of 0.8 the model would fail to converge. Given the value 1, the model would successfully fit.

It is still under thoughts why would that specific image need a larger domain for the updated shapes while all other images, coming from the same dataset converged using the same domain size.

TABLE 5.3
ASM Results

Image	Pose Parameters				Iterations coarse image	D_{max}	Initial me17	Final me17
	X_t	Y_t	Theta (θ)	Scale (s)				
1	391	230	2	71.9064	400	0.8	0.1350	0.0317
2	423	223	2	68.8257	180	0.8	0.1590	0.0251
3	395	256	2	69.2854	150	1	0.0893	0.0354
4	390	257	2	69.8559	200	0.8	0.1293	0.0196
5	366	194	2	62.1450	200	0.8	0.1471	0.0442

The graph below shows the ***me17*** fit of the initial face drawn with a blue line and the ***me17*** fit after convergence (red line). The blue line, as it is expected, has large values and also a large variation between images. After convergence, all ***me17*** fits are more or less on the same level and obviously have smaller values compared to the initial results.

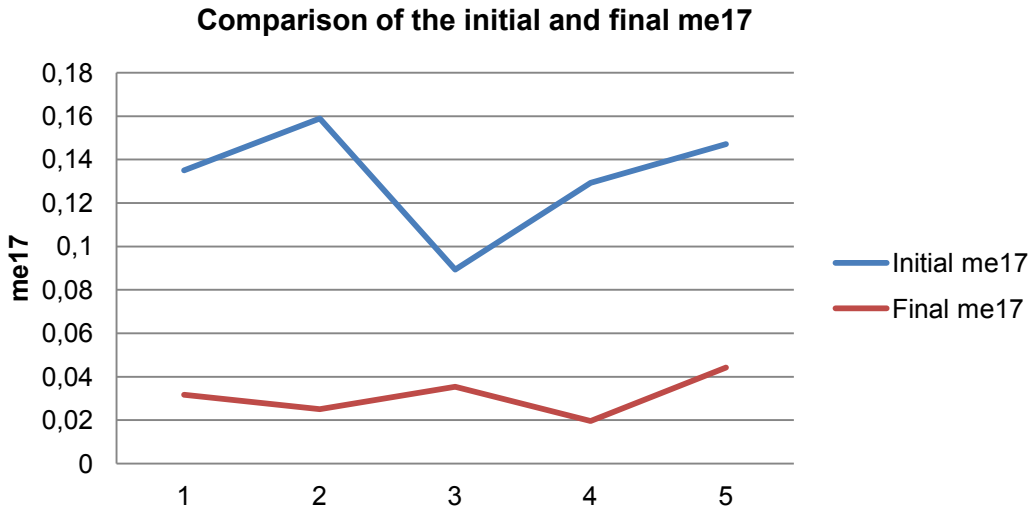


Figure 5.18

5.5 Adding noise during training

Data provided by Cootes contained training images exempted from noise and other disturbances as well as landmark points measured with great accuracy. It would be interesting to see how a model could be built and applied on a face image using perturbed images or data. This means adding noise to the training images or modifying the position of the landmark points by a small random number. As a result, the gray level appearance around each landmark point and also the variability of the data will be disturbed. Both cases will be examined.
The noise added to the training images was Gaussian noise with a mean value of zero and a variance of 0.0008. Experimenting with larger variance started to create images with no more strong edges and therefore pointless to try and build the gray level appearance around each landmark point. The fitting results can be seen from table 5.4. It is clear there is a large difference from the results in table 5.3.

First of all, the number of iterations in both cases is the same except in the first and last image. Secondly, the size of the allowable shape domain had to be modified for every image, otherwise the process would fail.
The explanation for this is still under thought so no clear answer can be given. What is interesting to mention is that these D_{max} values defined experimentally are specific. Increasing or decreasing the size of the predefined domain will lead to a failed search process. The ideal situation would be to have a fixed size of the shape domain and achieve a good result for every image based on this fixed size. The initial thought was to use the same number of iterations and domain sizes as mentioned in table 5.3 to compare the fit quality of a model trained from the initial images and then from the noisy ones.

TABLE 5.4
ASM Results using Noisy Training Images

Image	Iterations course image	D_{max}	Final me17
1	148	0.8	0.0202
2	180	0.5	0.0384
3	150	1.8	0.0508
4	200	1.5	0.0556
5	400	0.8	0.0498

The second modification was to maintain the initial training images unchanged and simply adjust the displacement of the landmark points so as to increase the variability of the face models. This was achieved by adding a random number to the existing landmark coordinates, within a range of 0-1. Table 5.5 gives the searching results for each image. In this case the number of iterations and domain sizes remain the same as in table 5.3.

TABLE 5.5
ASM Results from randomly modified Landmark points

Image	Iterations course image	D_{max}	Final me17
1	400	0.8	0.0647
2	180	0.8	0.3845
3	150	1	0.0323
4	200	0.8	0.0233
5	200	0.8	0.0699

The graph below illustrates the quality of fit between the initial results shown in table 5.2 and the modified results from tables 5.3 and 5.4. As it can be seen, training images forced by Gaussian noise provide results with a quality of fit very similar to the unmodified data.

Unfortunately, the number of iterations and also the size of the allowable shape domain had to be changed in the case of the Gaussian noise, otherwise the search would fail. Therefore, the comparison between these two approaches wouldn't work using the same property values. The reason why the noisy training images would force the allowable shape domain to fluctuate that much, is still under thoughts. Even increasing the maximum number of iterations and maintaining fixed the size of the domain, wouldn't make any difference. Consequently, the convergence of a face model, built from noisy images is pretty much dependant on the size of the domain rather than the quality of the gray level appearance around each landmark point.
The ***me17*** measurement of a face model, created firstly from the initial images and then by modified landmark points, is very close for images 1 and 5, almost coinciding for images 3 and 4 and very large for image 2. It is clear that even changing the displacement of the landmark points by a random number, the results are comparable, although in image 2 these small changes caused a steep change of the quality of the fit.

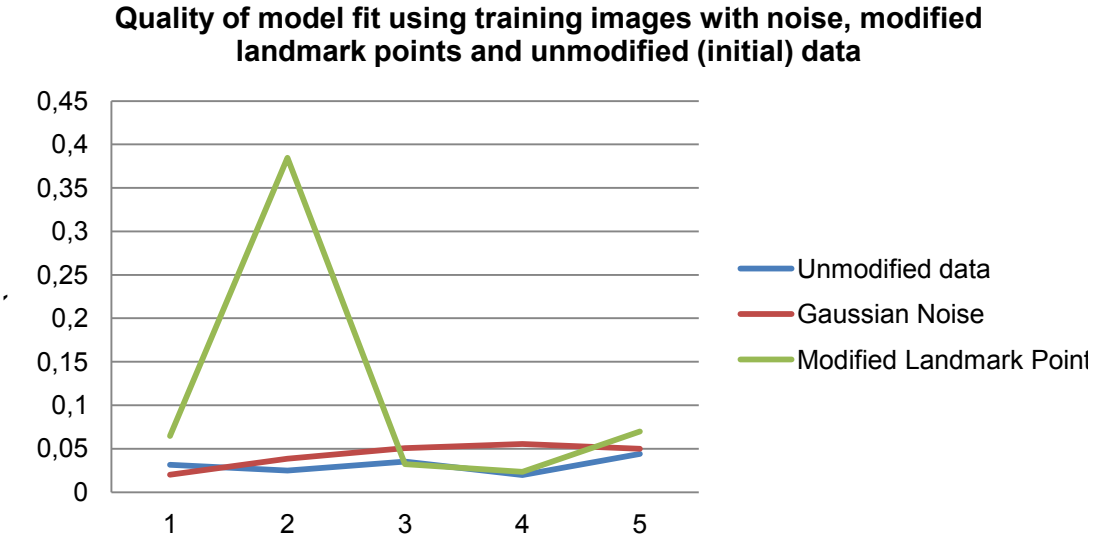


Figure 5.19

6

Discussions and Conclusions

6.1 Discussions

This master thesis involved detecting faces and fitting a trained face model on the detected face using a multiresolution Active Shape Model approach. As described previously, although the technique works quite well, improvements and proposals could be made, such as:

- *Automatic labelling of faces*

Creating a fully automated system to measure corresponding landmark points from a set of training images without failures, is still under development. Large improvements have been made towards this direction but still nothing significant. As the number of training images increase, measuring manually landmark points would be very time consuming and off course not accurate enough. It is important, points measured on the face to be placed precisely. Misplacing a landmark point may generate a face model which is distorted at that specific position. If there is noise in the images, the automatic system might fail to accurate place the landmark point. According to Cootes in [26], known points found could be set to one and the rest to zero. It is still possible to build reliable face models even if only a very small number of points is missing from the training set.

- *Choosing correct training examples*

For the comparison of landmark points, it is essential for all training faces to be accurately placed within their image. Typically, data alignment is important for bringing the trained shapes to a common coordinate system, so as to achieve a similar normalised size. In addition, the chosen training images should have similar intensities around the facial features. Moreover, the example images used for constructing the face model should contain the variability of the search face.

For example, frontal upright faces that look directly at the camera don't have much variation, thus the training faces should be similar. If there are training faces with side variation or variation of some facial parts, then they should be truncated because only a very small part of the total variation would be explained. On the other hand, using a large number of training images expressing the same variation is not reasonable enough, therefore it's better to train a model covering any expected variation.

- *Using 24 bit images*

The ASM method was applied on 8 bit training images. Nevertheless, the same process could be realized on 24bit images too. In this case, the gray scale appearance of every landmark point would be expressed by a $(3 \times n) \times 1$ vector, where n is equal to the number of interpolated points on the profile normal. For every channel, the gray level intensity is extracted and placed into the vector. Therefore, the profile matching is then to be done using all 3 channels simultaneously. Main disadvantage of this approach is that it is computationally more expensive, especially when a multiresolution ASM is used.

- *Having a small number of training examples*

It is often the case where there are fewer training examples N than landmark coordinates $2n$. In this situation no more than $N - 1$ degrees of freedom in the model are allowed and the eigenvectors of the $2n \times 2n$ covariance matrix can be calculated from a smaller $N \times N$ covariance matrix. Thus, using the method subject to Appendix C, the eigenvectors of the equivalent nonzero eigenvalues can be computed.

- *Using 2D profiles*

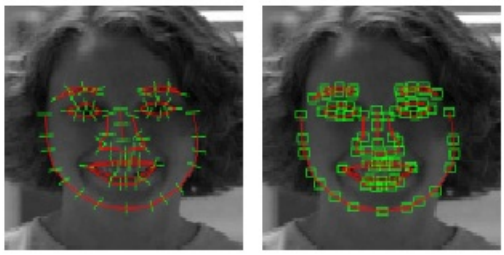


Figure 6.1: *(left image) 1D profiles created through the computation of the normals to the shape boundary and the 2D profiles (right image) defined using a square region around each landmark point (Seshadri & Savvides, 2009)*

According to Milborrow in [23], using 2D profiles rather than 1D could increase the quality of fit as well as the convergence time. Matching 2D patches could rise the probability of minimizing a given energy function rather than a simple Mahalanobis distance (see Fig. [6.1]).

- *Updating the Pose and Model Parameters*

Updating pose and model parameters is an important factor for the convergence method, although significant improvements have been made.

Choosing ideal weights for each of the landmark points may increase the confidence of their displacement towards a better position. Various scientists have introduced and developed their own weight method to check the displacement confidence of the points. Within this thesis, all weights except the ones calculated for the alignment of the training faces, where set to unity.
The experimental results could probably be improved if the confident of each point was known. Specifically, images where the face model failed to provide a good outline fit, could be improved using weights.
Moreover, the limit applied on the model parameters to maintain model's shape, proposed by Cootes in [25] is:

$$b \rightarrow b + W_b db - k_b W_b b \quad (0 < k_b < 1) \tag{6.1}$$

This would give more weight to shapes that are closer to the mean shape and less to the ones which are more deformed. On the other hand, applying these limits might not provide good solution between the face image and the model face. Thus, in this case it is preferable to use the fix limits introduced in Eq. [3.40]. Milborrow in [23] applies the fix limit approach too. Cootes in [32] suggests that after updating the model shape, weights should be given as to the quality of the update for each point. If a point is not moving in the correct direction, the weight should be decreased else otherwise. The formula is as follows:

$$w_i = \frac{1}{2 + |dX_i|^2} \tag{6.2}$$

where $|dX_i|$ is the adjustment correction in both x and y direction, of the point *i*.

- *Appling PCA on gray level intensities to find the desired movements*

This method involves modelling the gray level appearance around each landmark point using Principal Component Analysis and determines the desire movements through statistical comparison. According to Cootes in [27], this approach leads to more reliably and accurate search results. Furthermore, Kroon in [13] believes that this method works better on RGB images.

- *Real time Active Shape Models*

Active Shape Models have proved to work well in image segmentation, feature point location and extracting objects contours. Nevertheless, applying this method for face detection and recognition, using a video sequence is a big challenge. In this case, the face model would have to fit the face in real time. Thus, when the face is moving, the model would adapt to the face in real time. It would also be interesting to see how the ASM could be applied on a picture or a video sequence capturing multiple faces.

- *Comparison with the Active Contour Models*

Active Shape Model is a method very similar to the Active Contour Models (Snakes). The Active Contour Model approach is based on an energy minimized spline function guided by external and image forces (constraints) pulling it towards image features. In both cases, the results are comparable even for noisy and cluttered objects. The difference occurs in the training phase where the Active Contour Models are much easier to train rather than the ASM, but the model in the Active Contour case is not that specific and usually implausible shapes may be generated for complex objects [25].

Last but not least, the ASM method is more robust to noise, clutter and occlusions, because the variability is better controlled and the results remain within the predefined domain limits.

6.2 Conclusions

Active Shape Models is one of the simplest methods used to detect objects, in this case, facial features. Nevertheless, it is an essential requirement to have good initial values for the pose parameters (model parameters are set to zero). If the method fails to converge it is most likely that the placement of the face model in the image is not good enough. Therefore, the Viola/ Jones face detector is used to initially detect the face and subsequently place the face model as good as possible on the face image. Experiments made on a set of different facial upright frontal images, provided by the MUCT dataset, showed that the face detector and also the initial placement of the face model worked quite well in most cases. Failures, in the sense of bad model placement, occurred when a person was wearing glasses. On the other hand, well placed models could still fail to converge, on images where the gray intensities around the faces was constant, for example the existence of beard where the edges are not visible. The quality of the initial position of the model and also its final convergence was measured by the ***me17*** method.
In its initial form, the Active Shape Model method would run directly on the original image provided that profile normal lengths are given. In this thesis work this was extended, applying a multi resolution approach using two pyramid image levels with an objective to achieve a more robust model fit and convergence within the coarse image.
The reliability of the correction vectors, is strongly dependant from the position where the Mahalanobis distance is minimum within the displacement positions. The Mahalanobis distance is very sensitive when it comes to the computation of the inverse covariance matrix.
As already explained, strong linearities between the interpolated points could result a covariance matrix which is not invertible (not positive definite and not full rank).
That is why a modified covariance matrix was calculated (through spectral decomposition), which fulfils the inversion properties and helps to compute the correct point displacement.
Computing the profile normal is also very critical. Representing a face contour via line segments connected by landmark points and calculating the profile normal at each point using the next – previous point relationship, may not accurately represent the definition of a normal at some point. Therefore, knowing the number of landmark points per face part, the profile normals would be calculated independently for each part.
Moving on with the gray level appearance around each landmark point, the training images should have similar intensities values around each facial area. This is essential because the profile matching should be done between similar intensities.
What still remains ambiguous is the ideal size of the Allowable Shape Domain. Results showed that the size of the domain is very sensitive and if too small or too large the method will fail. For a predefined domain size, all Cootes images used for searching (except for one) converge. Still it is hard to believe, that although the same data set was used, this particular image would fail. Have in mind that even increasing the number of iterations the result wouldn't change.
Last but not least, adding noise to the training images and increasing variability of the landmark points affected the final result. It became clear that significant changes occurred after adding Gaussian noise to the images. For each one of the search images the domain size had to change otherwise the method would fail.

On the other hand, changing the displacement of the landmark points by a small random value didn’t result in any major changes.

Literature

[1] Alejandro F.Frangi, Joes J.Staal, Bart M., Max.A Viergever, *Active Shape Models Segmentation With Optimal Features*, IEEE Transactions on medical Imaging, Vol.21, No.8, August 2002.

[2] Amy Ross, *Procrustes Analysis*, Department of Computer Science and Engineering, University of South California, report.

[3] F.L.Bookstein, *Morphometric Tools for Landmark Data*, Cambridge University Press, London/New York, 1991.

[4] David Cristinacce, *Automatic Detection of Facial Features in Gray Scale Images*, University of Manchester, Faculty of Medicine, Dentistry, Nursing and Pharmacy, Ph.D thesis, 2004.

[5] E.H.Adelson, C.H.Anderson, J.R.Bergen, P.J.Burt, J.M.Ogden, *Pyramid methods in image processing*, RCA Engineer, 1984.

[6] Ghassan Hamarneh, Rafeef Abu – Gharbieh, Tomas Gustavsson, *Active Shape Models – Part I: Modeling Shape and Gray Variations*, Department of Signals and Systems, Imaging and Image Analysis Group, Chalmers University of Technology, Göteborg, Swedem. Proceedings on the Swedish Symposium on Image Analysis, SSAB 1998.

[7] Ghassan Hamarneh, Rafeef Abu – Gharbieh, Tomas Gustavsson, *Active Shape Models – Part II: Image Search and Classification*, Department of Signals and Systems, Imaging and Image Analysis Group, Chalmers University of Technology, Göteborg, Swedem. Proceedings on the Swedish Symposium on Image Analysis, SSAB 1998.

[8] Gary Bradski and Andrian Kaehler, *Learning OpenCV*, Computer Vision with the OpenCV Library, September 2008.

[9] Gordon Simons and Yi – Ching Yao, *Approximating the inverse of a symmetric positive definite matrix*, Department of Statistics, University of North Carolina Chapel Hill and Institute of Statistical Science, Academia Sinica, Taipei, Taiwan.

[10] Jung – Bae Kim, Seok – Cheol and Ji – Yeon Kim, *Fast Detection of Multi – View Face and Eye based on Cascaded Classifier*, Computing Lab, Samsung Advanced Institute of Technology, Korea.

[11] James E.Gentle, *Numerical Linear Algebra for Applications in Statistics*, Springer.

[12] Keshav Seshadri, Marios Savvides, *Robust Active Shape Model for Landmarking Frontal Faces*, Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, June 2009.

[13] Dr. Kroon, University of Twente, *Applying the Active Shape Model approach on hand images*, MatLab Code, February 2010. Link: http://www.mathworks.com/matlabcentral/fileexchange/26706-active-shape-model-asm-and-active-appearance-model-aam

[14] Juan Soulié, *C++ Language Tutorial*, released in June 2007. Available online at: http://www.cplusplus.com/doc/tutorial/

[15] Mikkel B. Stegmann and David Delgado Gomez, *A Brief Introduction to Statistical Shape Analysis,* Informatics and Mathematical Modelling, Technical University of Denmark, 2002.

[16] M.I.Jordan, *An Introduction to Linear Algebra in Parallel Distributed Processing*, Formal Analysis, Chapter 9.

[17] Min.Wang, Yi ling.Wen, Li.Fang, Wei ping.Sun, *An Improved Active Shape Model Application on Facial Feature Localization*, The School of Information and Control Engineering, Xi'an University of Architecture and Technology.

[18] Paul Viola and Michael Jones, *Rapid Object Detection using a Boosted Cascade of Simple Features*, CVPR 2001.

[19] Rainier Lienhart and Jochen Maydt, *An extended Set of Haar – like Features for Rapid Object Detection*, Intel Labs, Intel Corporation, Santa Clara, California, USA.

[20] Richard Hartley and Andrew Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, March 2004.

[21] Stephen Milborrow, John Morkel, Fred Nicolls, *The MUCT Landmarked Face Database*, University of Cape Town.

[22] Stephen Milborrow, *Locating Facial Features with Active Shape Models*, Master Thesis, University of Cape Town, November 2007.

[23] Stephen Milborrow and Fred Nicolls, *Locating Facial Features with an Extended Active Shape Model*, Department of Electrical Engineering, University of Cape Town, South Africa.

[24] Rasmus Elsborg Madsen, Lars Kai Hainsen and Ole Winther, *Singluar Value Decomposition and Principal Component Analysis*, Lecture notes.

[25] T.F.Cootes, C.J.Taylor, D.H.Cooper, J.Graham, *Active Shape Models - Their Training and Application*, Computer Vision and Image Understanding, Vol. 61, No.1 January, pp. 38 - 59, 1995.

[26] T.F.Cootes, C.J.Taylor, D.H.Cooper, J.Graham, *Training Models of Shape from Sets of Examples*, Department of Medical Biophysics, University of Manchester.

[27] T.F.Cootes, C.J.Taylor, *Active Shape Model Search using Local Grey – Level Models: A Quantitative Evaluation*, Department of Medical Biophysics, University of Manchester.

[28] T.F.Cootes, C.J. Taylor, *Statistical Models of Appearance for Computer Vision*, Imaging Science and Biomedical Engineering, University of Manchester, March 2004.

[29] T.F.Cootes, C.J.Taylor, *Active Shape Models – 'Smart Snakes'*, Department of Medical Biophysics, University of Manchester.

[30] T.F.Cootes, C.J.Taylor, A.Lanitis, *Active Shape Models: Evaluation of a Multi – Resolution Method for Improving Image Search*, Department of Medical Biophysics, University of Manchester.

[31] T.F.Cootes, A.Hill, C.J.Taylor, *Active Shape Models and the Shape Approximation Problem*, Department of Medical Biophysics, University of Manchester.

[32] T.F.Cootes, A.Hill, C.J.Taylor, J.Haslam, *The Use of Active Shape Models For Locating Structures in Medical Images*, Department of Medical Biophysics, University of Manchester.

Appendix A: Software structure

Users Input

For the program to executed, the user must set the following parameters:

Table A1

Description	Code variable name	Initialization
Directory of the training images	***string*** *images_names*	FIXED
Directory of the landmark points	***string*** *file_folder*	FIXED
Directory of the Pyramid Images	***string*** *Pyramid_Folder*	FIXED
Search Image	***IplImage**** *Search_image*	USER
Number of label points	*int number_label_pts*	USER
Threshold	*float threshold*	USER
Length of landmark intensity profile (model)	*int nsamples_model*	USER
Search length	*int nsamples_search*	USER
Number of Pyramid scales	*int nScales*	USER

The first three directories remain fixed and the user should not modify the name of the folder where the images and landmark coordinates are stored. The program reads two text files, one for the images and one for the landmark coordinates. The user can only modify what is inside these text files. This means that training images and corresponding landmark points which are not required in the training process can be discarded by erasing the index (name) of this set.
Every row represents a subset of the total training set and is linked either to the text file containing the corresponding coordinates or the image of the specific subset. So for example in figure [A1.a], when the system reads the first line which is "***data11.txt***" it will then call the text file containing the coordinates of the 11th training image. For the images the procedure is similar.

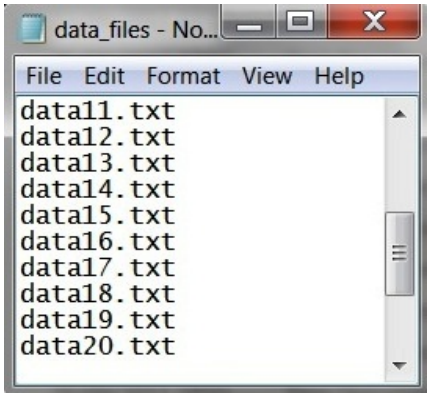


Figure A1.a: *text file of landmark points*

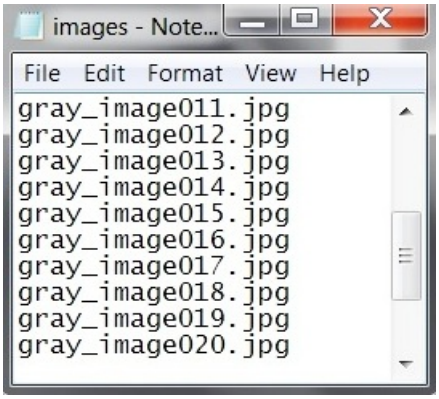


Figure A1.b: *text file of the training images*

Declaration of structures and classes

This chapter describes in detail all main structures/classes implemented for this thesis work. The first structure, called ***Whisker*** {} has the following form:

```
struct Whisker
{
        CvMat *Contour_Points;
        CvMat *Contour_Normals;
        CvMat *intervalx;
        CvMat *intervaly;
        CvScalar samples;
        CvMat *derivatives_norm;
        CvMat *covariance_matrix;
        CvMat *mean_samples;
        vector <CvMat*> derivatives_norm_vec, inputMatrix;
};
```

Listing A1: *Whisker structure*

The above structure is used to describe and capture all information needed for defining a profile normal of a landmark point. Its members are described in the table below:

Table A2

Members name	Definition
***CvMat** *Contour_Points*	It's a matrix containing the coordinates of the landmark points in the current training image
***CvMat** *Contour_Normals*	It's a matrix containing the normals to the landmark points in the image
***CvMat** *intervalx*	The linear intervals on the line in the x direction
***CvMat** *intervaly*	The linear intervals on the line in the y direction
***CvScalar** samples*	Number of samples taken on the profile normal. This number concerns only the one side of the normal.
***CvMat** *derivatives_norm*	It's a matrix containing normalised derivatives of the gray level appearance
***CvMat** *covariance_matrix*	Covariance matrix of a point
***CvMat** *mean_samples*	This is the mean normalised derivative of a point
vector <CvMat*> *derivatives_norm_vec*	It's a vector that groups each landmark point, its normalised gray scale appearance in all images
***vector <CvMat*>** *inputMatrix*	A vector containing the gray level appearance of the current point in all images

```
class GrayLevelAppearance
{
public:
        Whisker CurrentImage;
        IplImage *image;

        void GetContourNormalsFace (string path, CvMat *Contour_Points,
                                    CvMat *ContourNormals);

        void GetContourNormals (string type, CvMat *Contour_Points,
                                CvMat *ContourNormals);

        void linear_intervals (CvMat *Contour_Points, CvMat *ContourNormals,
                               CvMat *intervalx, CvMat *intervaly, CvScalar *samples);

        void GetDerivatives (IplImage *src_image, CvMat *intervalx, CvMat
*intervaly, CvMat *derivatives_norm, double samples);
};
```

Listing A2: *Class GrayLevelAppearance*

The above is considered a more generalised class, containing the aforementioned structure of listing A1, the image from which the gray level appearance of each landmark point will be extracted, subject to the following four functions described below.

```
class PyramidLevel
{
public:
        GrayLevelAppearance  Appearance;
        vector<CvMat*> S;                   // Mean Covariance matrix
        vector<CvMat*> gs;                  // Mean normalised intensity
};
```

Listing A3: *Class PyramidLevel*

The class ***PyramidLevel*** {} is an even more generalised class from the above two. It contains three public members, the first one being the class described previously and the other two are vectors containing the covariance matrix and the mean gray normalised appearance of each landmark point.
In a tree representation, ***PyramidLevel*** {} would be the parent, ***GrayLevelAppearance*** {} the inner node and the structure ***Whisker*** {} the leaf node. The relationship connecting the ***PyramidLevel*** {} class with the ***GrayLevelAppearance*** {} is a composition because even if the ***PyramidLevel*** {} is destroyed, the ***GrayLevelAppearance*** {} can be still be defined in the initial training image (only the pyramid of the image is destroyed). On the other hand, if the ***GrayLevelAppearance*** {} is destroyed, a whisker cannot exist. In this case, the relationship will be aggregation. The UML diagram below illustrates this relationship:

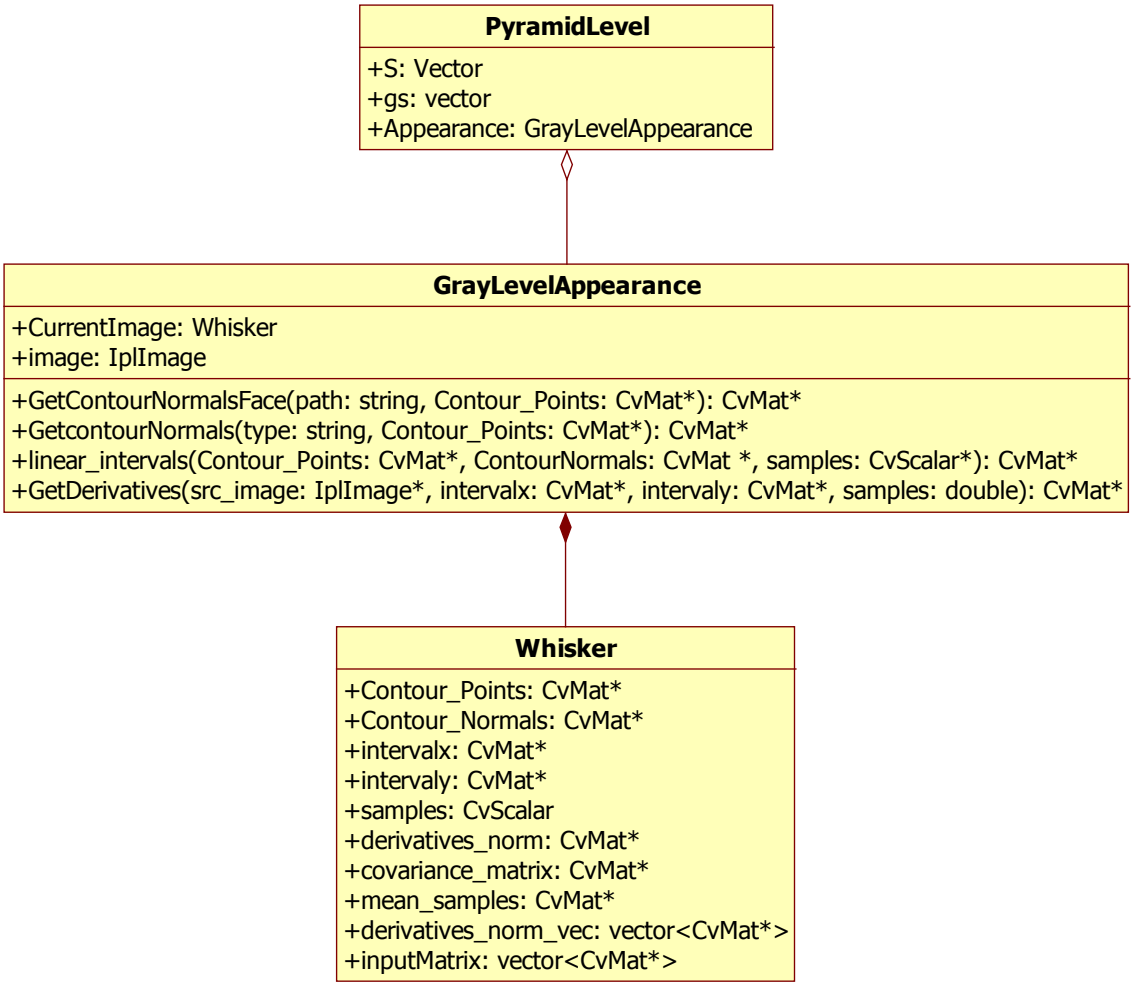


Figure A2: *UML representation of the connectivity between the classes and structures*

```
struct PCAParam
{
        CvMat *P;
        CvMat *b;
        CvMat *mean_ASD;
        CvMat *S;
        float Threshold;
        CvMat *eig_Vec;
        CvMat *eig_Val;
        vector<double> modes;
        vector<CvMat*> new_shapes;
};
```

Listing A4: *Structure PCAParam*

The above structure ***PCAParam*** {} is the main structure used to capture the statistics of the aligned shapes. The table below gives the definition for the members of this structure:

Table A3

Member Name	Description
***CvMat** *P*	A matrix containing the most significant eigenvectors
***CvMat** *b*	The model parameter vector whose size depends on the number of the most significant eigenvalues
***CvMat** *mean_ASD*	The mean aligned shape
***CvMat** *S*	The covariance matrix of the aligned shapes
***float** Threshold*	Part of variance to be explained by the shape model - Defines the number of modes
***CvMat** *eig_Vec*	Total number of eigenvectors deduced from the aligned shapes
***CvMat** *eig_Val*	Total number of eigenvalues deduced from the aligned shapes
***vector<double>** modes*	A vector containing the most significant eigenvalues
***vector<CvMat*>** new_shapes*	A vector containing the coordinates of new shapes produced from the most significant eigenvalues and eigenvectors

Last but not least, to compute the ***me17*** quality fit, a text file named ***me17_fixedPoints.txt*** is used. This file contains the fixed landmark coordinates of the face where the face model is applied on. Thus, when the search process is completed, this text file is inserted into the program for comparison. It's also used when the face model is placed on the image, in order to check the quality of the initial position.

Appendix B: Aligning a pair of shapes

Given two similar shapes, x_1 and x_2, a rotation θ, scale s and translation (t_x, t_y) could be found to map x_2 into $M(x_2) + t$ as to minimize the weighted sum:

$$E = (x_1 - M(s,\theta)[x_2] - t)^T W (x_1 - M(s,\theta)[x_2] - t) \quad \text{(B.1)}$$

where,

$$M(s,\theta)\begin{bmatrix} x_{jk} \\ y_{jk} \end{bmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta \\ s\sin\theta & s\cos\theta \end{bmatrix}\begin{bmatrix} x_{jk} \\ y_{jk} \end{bmatrix} \quad \text{(B.2)}$$

$$t = \begin{bmatrix} t_x, t_y, \ldots, t_x, t_y \end{bmatrix}^T \quad \text{(B.3)}$$

and *W* is a diagonal matrix of weights for each point.

If,

$$a_x = s\cos\theta \quad a_y = s\sin\theta \quad \text{(B.4)}$$

a least square approach (differentiate with respect to each of the variables a_x, a_y, t_x, t_y) could be achieved:

$$\begin{pmatrix} X_2 & -Y_2 & W & 0 \\ Y_2 & X_2 & 0 & W \\ Z & 0 & X_2 & Y_2 \\ 0 & Z & -Y_2 & X_2 \end{pmatrix}\begin{pmatrix} a_x \\ a_y \\ t_x \\ t_y \end{pmatrix} = \begin{pmatrix} X_1 \\ Y_1 \\ C_1 \\ C_2 \end{pmatrix} \quad \text{(B.5)}$$

where,

$$X_i = \sum_{k=0}^{n-1} w_k x_{ik} \quad \text{(B.6)} \qquad Y_i = \sum_{k=0}^{n-1} w_k y_{ik} \quad \text{(B.7)}$$

$$Z = \sum_{k=0}^{n-1} w_k \left(x_{2k}^2 + y_{2k}^2\right) \quad \text{(B.8)} \qquad W = \sum_{k=0}^{n-1} w_k \quad \text{(B.9)}$$

$$C_1 = \sum_{k=0}^{n-1} w_k (x_{1k}x_{2k} + y_{1k}y_{2k}) \quad \text{(B.10)} \qquad C_2 = \sum_{k=0}^{n-1} w_k (y_{1k}x_{2k} - x_{1k}y_{2k}) \quad \text{(B.11)}$$

Appendix C: Calculating the eigenvectors of the covariance matrix when there are fewer samples than coordinates

When there are fewer training examples *N*, than coordinates *2n*, the eigenvectors of the $2n \times 2n$ covariance matrix *S* can be calculated from the eigenvectors of a smaller $N \times N$ matrix derived from the same data. Due to the fact that the eigenvector calculation time is equal to the cube size of the matrix, this method could save up a lot of time.

Given N examples x_i $(i = 1, \dots, N)$, let D be a $2n \times N$ matrix:

$$D = (x_1, x_2, \dots, x_N) \tag{C.1}$$

As mentioned in chapter 3, the covariance matrix could be written in the form:

$$S = \frac{1}{N} DD^T \tag{C.2}$$

Let T be a $N \times N$ matrix:

$$T = \frac{1}{N} D^T D \tag{C.3}$$

and let e_i $(i = 1, \dots, N)$ be the unit, orthogonal eigenvectors of T corresponding to eigenvalues γ_i:

$$Te_i = \gamma_i e_i \quad (i = 1, \dots, N) \tag{C.4}$$

Then from [C.3], [C.4] is equal too:

$$\frac{1}{N} D^T De_i = \gamma_i e_i \tag{C.5}$$

Premultiplying by D,

$$\frac{1}{N} DD^T De_i = \gamma_i De_i \tag{C.6}$$

$$S(De_i) = \gamma_i (De_i) \tag{C.7}$$

Then if e_i is an eigenvector of T, then De_i is an eigenvector of S and has the same eigenvalue. The N orthogonal eigenvectors of *S* are then p_i $(i = 1, \dots, N)$, where:

$$p_i = \frac{1}{\sqrt{\gamma_i N}} De_i \tag{C.8}$$

with $\lambda_i = \gamma_i$.
The scaling factor in [C.8] is required to give the eigenvectors unit length.
Orthogonality can be easily shown:

$$p_i^T p_j = \frac{1}{\gamma_i N} e_i^T D^T De_j = \frac{1}{\gamma_i} e_i^T Te_j = e_i^T e_j = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases} \tag{C.9}$$