Department of Computer Science
UNIVERSITY COLLEGE LONDON

MSc Project Report

Using the Integrated Shape and Pose Model for Recognition

Author: Fernando Segundo (fersegundo@yahoo.es)
Supervisor: Bernard F. Buxton

September 2003

Abstract

The Linear Combination of Views (LCV) technique enables the representation of three-dimensional objects by a linear combination of several two-dimensional images or line drawings.

Our system models the appearance of images by simple grey-level profiles, and uses them to build an Active Shape Model (ASM) type model that automatically searches and locates the landmark points of an object in a given image, constraining the search only with the LCV model. This way, our Integrated Shape and Pose Model (ISPM) is used to search novel images simultaneously for face features and to determine their pose.

Our work, proceeds in two main stages: first we will prove that our approach is valid when using line-drawings as input data, and secondly, we will move on to searching in grey-level images. As it turns out, the work carried out proved to be quite successful from a learning and also practical point of view.

Acknowledgments

I would like to thanks my project supervisor, Bernard F. Buxton for the help he has given me throughout the project, and for all the things I have learnt form him. Also, and specially, to Ben Dias who has been of incommensurable help during the last few months.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Problem statement and motivation

Statistical shape modelling has been an active area of interest over the past decade. Cootes et. al. [8, 7] showed how Flexible Shape Models can be used to model the shape of deformable objects such as mechanical parts, hands and faces. This technique has been proved to give very positive results for feature recognition. For example, Matthews et. al. [25] compared the performance of an Active Shape Model (ASM) based lip reader and an Active Appearance Model (AAM) based one.

The main flaw of this technique is that it does not cope with pose changes of the object in front of the camera [14]. By pose, we mean a rotation through a large angular range about axes parallel to the image plane (in the case of a head, looking up, down, right or left).

This flaw could be dealt with by modelling the shape in three-dimensions or by building a coupled-view Flexible Shape Model (cvFSM) [9,7]. Although the coupled-view FSM successfully extends the range of views over which the FSM may be used, it does not address the theoretical problems associated with the FSM. In particular, the FSM may confound *extrinsic* view on pose variations with *intrinsic* variations of an object's shape and appearance. For example, for face images, the FSM often produces modes of variation that mix view and expression changes and its performance may be biased as a function of viewpoint [13,14]. Similarly, the cvFSM is not properly view independent and, indeed, seeks to relate viewing angle to the model parameters [9,7].

More interesting is the Linear Combination of Views (LCV) approach, in which three-dimensional objects are represented by a linear combination of two-dimensional images (Hansard and Buxton [18]) or line drawings (Ullman and Basri [28]). Although this technique produces very good, realistic-looking results for rigid objects and can correctly account for changes in viewpoint, it breaks down when confronted with objects that can change shape.

Dias and Buxton [11,12,15] successfully demonstrated that these two techniques (FSM and LCV) can be combined into an Integrated Shape and Pose Model (ISPM), thus modelling the shape and pose of a flexible object independently.

The motivation for this project is thus to extend the work of Dias and Buxton, to model the appearance (grey-level values) of images and then, using the grey-level profiles, to build an Active Shape Model (ASM) type model that will automatically search and locate the landmark points of an object in a given image, constraining the search only with the LCV model. This way, this Integrated Shape and Pose Model (ISPM) could be used to search novel images simultaneously for face features and to determine their pose.

Developing a system that could extend the ISPM approach of Dias and Buxton [11,12,15] is, unfortunately, too ambitious for a short MSc project. Our aim therefore has been restricted to seeing if the LCV method provides sufficient constraint to enable a simultaneous search for features and pose to be made for *rigid objects*. In particular, the nose of face images will be used as the target object.

This "rigid object" is chosen as our exemplar because, if the features from such stable points of the face (points that do not move much while changing expression) could be successfully detected and the face pose determined, we could use the ISPM to build an Active Shape and Pose Model (ASPM) of other face features, such as the mouth, that is independent of the face pose. The information provided by the ASPM, could be further analysed to build a robust speech recognition system, or to animate faces of avatars. [10].

1.2 Report Structure

Prior to being able to search the grey-level images for features, we must prove that the search algorithm is geometrically correct, i.e. that it works given geometric data. Our work therefore proceeds in three stages. First, points will be used as input data. Secondly, we will use line-drawings as input data (this could be regarded also as the minimal grey-level profile possible). Finally, we'll move on to grey-level images. This report will be almost completely divided into two parts. In the first part the first two geometric steps will be covered, while the second part will cover the last step in which

pixel values or grey-levels are used. The first part will covered in the chapters 2, 3, 4, 5 and 6, while chapters 7, 8, 9, 10 and 11 will cover the second part.

The chapter's contents are as follows:

- The first half of chapters 2 and 7, contain an overview of the previous work in the field, some of which this project is based, identifying problems in the systems and assessing their strengths and weaknesses. In these chapters, the techniques used to build the system are also reviewed.

- In chapters 3 and 8, algorithms used are described, together with the description of some of the data structures used and how the system is structured into functions.

- In chapters 4 and 9 we give a brief overview of the implementation.

- In chapters 5 and 10 we introduce our framework and methodology for the testing and validation of our system, explaining why we chose to approach the testing phase in such a way.

- In chapter 6 and 11 a summarised version of the results is provided. These are an overview of the results that lead to the conclusions. We discuss the parameters involved in the system and those chosen to be varied and tested.

- Finally, in chapter 12 we assess the overall quality of the work. We present what we believe are the main achievements of this project and discuss areas for improvement and future work.

The final chapter is followed by a bibliography, and, last of all, we provide a more detailed set of results in appendix A and a source code listing in appendix B.

2 Geometrical approach

In this first part of the project, the aim is to automatically locate landmark points given a binary line-drawing image. To do this, an iterative algorithm will be necessary that will search for valid instances of the object (in our case, a stylised, idealised geometrical drawing of the nose). This initial approach can be viewed as a testing stage for the pose-alignment algorithm, in which the problem of finding where the object's edges are located was obviated.

In fact, this initial approach is completely analogous to the way Cootes and Taylor [8,7] proceeded in their early work on development of the PDM/FSM. Their early work is therefore briefly reviewed first in the remainder of this chapter. It is followed by a review of the linear combination of views (LCV) approach. The two are then brought together to define the initial geometrical pose/feature search.

2.1 Background

2.1.1 Active Shape Models

Cootes and Taylor [8,6,7] developed this technique, which they defined as algorithms that iteratively refine the estimate of the pose, scale and shape of models of image objects. The method consists of a combination of two components: a compact model representing the shape of a set of variable objects, known as a Point Distribution Model (PDM), and an iterative algorithm of image search known as the Active Shape Model (ASM).

Before going into the iterative algorithm of the ASM, the PDM will be introduced:

2.1.1.1 Point Distribution Models

A PDM is generated via a principal components analysis of a training set of N object descriptions $\{y_i, (1 \leq i \leq N)\}$. An object description, y_i, is simply a labelled set of points $\{y_{i,j},(1 \leq j \leq n)\}$ which we will call landmarks. The analysis involves: aligning the set of examples into a common frame of reference, $\{x_i = \text{aligned}(y_i),(1 \leq i \leq N)\}$; calculating the mean of the aligned examples, $\bar{x}$, and the deviation from the mean of each aligned example $\Delta x_i = x_i - \bar{x} \partial x_i$; calculating the eigensystem of the co-variance matrix of the deviations, $C = \frac{1}{N}\sum_{i=1}^{N} \Delta x_i \, \Delta x_i^T$. The t principal eigenvectors of the eigensystem are then used to generate examples of the modelled objects via the expression:

$$x = \bar{x} + Pb \, \mathrm{X} \qquad (2\text{-}1)$$

where b is a t-element vector of shape parameters. P is a (2n x t) in 2D or (3n x t) in 3D matrix of the first t eigenvectors of the covariance matrix C, i.e. the eigenvectors p(k) corresponding to the t largest eigenvalues $\lambda(k)$, k=1…t, of C. Since C is positive semi-definite, such vectors may always be chosen. Given a shape described by a set of landmarks x, the shape parameters b may be formed from

$$b = P^T (x - \bar{x}) \qquad (2\text{-}2)$$

By selecting b from a pre-defined shape vector space, established from the set of training examples, new instances of the modelled object(s) can be generated. This enables the PDM to represent previously unseen examples and forms the basis for locating examples of the modelled object(s) in unseen images via the ASM.

2.1.1.2 Image Search

For a flexible PDM, the search for an example object in a new image is carried out via the ASM. Each iteration of the search has two steps: image data interrogation followed by shape approximation.

In the first step, we commence with an initial estimate of the position, orientation, scale and shape of the model points in an image. The image data in the vicinity of each model point is interrogated, usually along profiles normal to the boundary/surface of the model object. This leads to a new proposed set of landmarks within the image.

In the second step, limits are placed on the shape parameters obtained from by the set of model points found in step one, whilst enforcing global shape constraints imposed by the training set, and allowing only certain deformations to occur. Thus, a correct set of points is obtained that is treated as the initial estimate in the next iteration. This way, the model instance slowly changes its parameters so as to completely fit the object in the image.

All this is assuming we are in the model frame of reference (aligned). In practice, of course, a new image is very unlikely to contain the object of interest in such a convenient position. To get back to an image frame of reference, the object model must be also allowed to rotate, scale and shift (similarity transformation):

$$x'_j = Q(\bar{x}_j + p_j b) + \varepsilon'_j, \quad \text{for every landmark } \boldsymbol{j} \tag{2-3}$$

$$\text{where the transformation} \quad Q = \begin{bmatrix} s\cdot\cos(\theta) & -s\cdot\sin(\theta) & t_x \\ s\cdot\sin(\theta) & s\cdot\cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2-4}$$

if homogeneous co-ordinates of landmark *j* are written as a column vector $x_j = \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix}$.

This technique has an important limitation. It assumes a near fronto-parallel view and breaks down when the object rotates through a large angular range about axes parallel to the image plane. In addition, the method tends to mix *extrinsic* variations in the shape of an object in an image caused by changes of viewpoint, with *intrinsic* variations caused, for example, for a face, by changes of expression. Cootes and Taylor [7,9] tried to overcome this by building two different Flexible Shape Models on images captured simultaneously from two views and coupling them together in a single model. Dias and

Buxton [11,12,15] criticised this method, saying that pose and shape parameters should be modelled independently because they ought to be independent of each other.

To overcome this deficiency, Dias and Buxton, [11,12,15] have recently introduced an integrated shape and pose modelling procedure that uses the affine trifocal tensor to provide the mappings between images of an object taken from different viewpoints and subsequently builds a statistical model by use of PCA techniques [11,12,15]. The work reported in these papers show that the modelling technique is very successful and, in particular, enables extrinsic pose variations to be well separated from intrinsic variations. However, a search technique akin to the ASM or equivalent to those used by Ullman and Basri [28] for the linear combination of views (described in the following section) has yet to be developed. The work reported here may be regarded as a first step towards this objective by developing a search technique that is effective in the linear combination of views approach which itself may be used when the object of interest is rigid and, of course, imaged under affine conditions of weak or paraperspective.

Our approach is to see if such a method can be developed by using the kind of techniques employed by Cootes and Taylor [6] for the ASM, for line drawings; processed on sample images, for example with edge profiles; or for an active appearance model (AAM) when (in principle) all pixels in the image of an object are used.

We note in passing that, in particular, when line drawings or edge profiles are used, these techniques are very similar to those used by Harris in a system developed for the real-time tracking of rigid objects. Harris [1] used a similar search technique as part of the RAPiD (Real-time Attitude and Position Determination) system. Before carrying out this search, the authors essentially placed several points midway between the model landmark points to add variability and to avoid the search being ill-defined ot getting stuck in a local, but sub-optimal solution.

2.1.2 Linear Combination of Views

First introduced by Ullman and Basri in 1991 [28], the LCV system enables a three-dimensional object to be represented as a linear combination of several two-dimensional images of the object, based on the linear relations which exist between images taken with an affine camera. The two-dimensional images representing the object, are called basis views. In mathematical terms, let the set of basis views be $\mathrm{M} = \{M_i,...,M_k\}$. A new image P will be a valid instance of the three-dimensional object if:

$$\mathrm{P} = \sum_{i=1}^{k} \alpha_i \mathrm{M}_i \tag{2-5}$$

where α_i are constants that indicate how close the instance is to each of the basis views. Ullman and Basri worked with line-drawings of rigid objects, so that (2-5) is applied to points on an object such as corners on the line drawing or, more generally, points of high curvature on the object boundary or edge features.

Hansard and Buxton [18] extended this idea to real images in the context of novel view synthesis by fitting polynomial models to the coefficients of the combination. This enabled novel views of an object to be synthesized, for example, by interpolation of existing images. Their work gave very realistic-looking results but was still restricted to rigid objects.

We shall now summarize briefly the LCV approach as applied to the geometry of a set of landmark points in the images of the object or class of objects of interest. If we assume that two basis views are used (as we will be using in our system), in LCV, the j^{th} landmark, (x_j, y_j) in an image is related to the j^{th} landmark in the two basis views $(x_j^{'}, y_j^{'})$ and $(x_j^{''}, y_j^{''})$ by two linear relationships which, if the images are centred (i.e. with the origin of coordinates chosen at their centroids), may be written as:

$$\begin{aligned} \mathrm{x_j} &= \mathrm{a_1 x_j^{'} + a_2 y_j^{'} + a_3 x_j^{''} + a_4 y_j^{''}} \\ \mathrm{y_j} &= \mathrm{b_1 x_j^{'} + b_2 y_j^{'} + b_3 x_j^{''} + b_4 y_j^{''}} \end{aligned} \tag{2-6}$$

Here, by centred images we mean that the origin of the image coordinates (x, y), (x', y') and (x'', y'') is always chosen at the centroid of the landmarks visible in each image. If the same number of landmarks, n say, is visible in each and they are in 1:1 correspondence (i.e. in principle (x_j, y_j), (x'_j, y'_j) and (x''_j, y''_j) are images of the same point on the object in 3D), this means

$$\begin{aligned} \frac{1}{n}\sum_{j=1}^{n} x_i &= \frac{1}{n}\sum_{j=1}^{n} y_i = 0 \\ \frac{1}{n}\sum_{j=1}^{n} x'_i &= \frac{1}{n}\sum_{j=1}^{n} y'_i = 0 \\ \frac{1}{n}\sum_{j=1}^{n} x''_i &= \frac{1}{n}\sum_{j=1}^{n} y''_i = 0 \end{aligned} \qquad (2\text{-}7)$$

As a result of (2-7), there are no constant terms (a_0 or b_0) in equations (2-6).

With the equation (2-6), and given at least four corresponding landmark points in the image being represented, and in both basis views, we can use standard least squares techniques [16,21] to estimate the eight LCV coefficients. This implementation of the LCV, although it effectively allows for symmetric treatment of the basis views [21], does not impose the correct multi-view constraints on the LCV coefficients as it is over-complete, in the sense that, in principle, only three of each of the four variables $(x'_j, y'_j, x''_j, y'''_j)$ are required in (2-6). In fact, under perfect affine projection and in the absence of noise, one in each of the coefficients a_1-a_4 and b_1-b_4 is independent on the others

Dias and Buxton [11,13], thus reformulated the LCV technique via the Centred Affine Trifocal Tensor (CATT) [2] to impose the correct multi-view constraints while preserving the symmetric treatment of the basis views. Under the CATT, the linear relationships between three images with corresponding landmark points (x_j, y_j), (x'_j, y'_j) and (x''_j, y''_j) are represented by twelve parameters $\{t_1,...,t_{12}\}$, that can be written as follows:

$$
\begin{aligned}
t_1 x_j + t_5 y_j + t_9 x_j^{''} + t_{11} x_j^{'} &= 0 \\
t_2 x_j + t_6 y_j + t_9 y_j^{''} + t_{12} x_j^{'} &= 0 \\
t_3 x_j + t_7 y_j + t_{10} x_j^{''} + t_{11} y_j^{'} &= 0 \\
t_4 x_j + t_8 y_j + t_{10} y_j^{''} + t_{12} y_j^{'} &= 0
\end{aligned}
\tag{2-8}
$$

where, the parameters satisfy the following constraints [27]:

$$
\begin{vmatrix} t_1 & t_2 & -t_9 \\ t_3 & t_4 & -t_{10} \\ -t_{11} & -t_{12} & 0 \end{vmatrix} = 0 \;,\; \begin{vmatrix} t_5 & t_6 & -t_9 \\ t_7 & t_8 & -t_{10} \\ -t_{11} & -t_{12} & 0 \end{vmatrix} = 0 \tag{2-9}
$$

The CATT may be normalized by choosing

$$
\sum_{i=1}^{12} t_i^2 = 1 \tag{2-10}
$$

Constraints (2-9) deal with the over-completeness. The normalization is required to ensure that the equations do not admit a trivial solution.

There is a relationship between the parameters of the LCV equations (2-6) and the elements of the CATT, viz.:

$$
\begin{aligned}
a_1 &= \frac{qt_{12} - st_{11}}{ps - qr} \quad & b_1 &= \frac{pt_{12} - rt_{11}}{qr - ps} \\
a_2 &= \frac{qt_{11} - st_{12}}{ps - qr} \quad & b_2 &= \frac{pt_{11} - rt_{12}}{qr - ps} \\
a_3 &= \frac{qt_{10} - st_9}{ps - qr} \quad & b_3 &= \frac{pt_{10} - rt_9}{qr - ps} \\
a_4 &= \frac{qt_9 - st_{10}}{ps - qr} \quad & b_4 &= \frac{pt_9 - rt_{10}}{qr - ps}
\end{aligned}
\tag{2-11}
$$

where,

$$
\begin{aligned}
p &= t_1 + t_4 \\
q &= t_5 + t_8 \\
r &= t_2 + t_3 \\
s &= t_7 + t_6
\end{aligned}
$$

These relationships are used in the implementation, for example, so that the grey levels of images whose geometry (i.e. landmark points) is to be mapped via the CATT may be inferred by the methods developed by Koufakis and Buxton [22]. In addition it is clear from these relationships and the fact that the constraints on the CATT (2-9) are cubic, taht deriving a set of suitable constraints that could be applied directly to the LCV coefficients a_1-a_4 and b_1-b_4 is very complicated. Note that, in the above each coefficient a_1-a_4 and b_1-b_4 is rational in the elements of the CATT. This is because the CATT equations are homogeneous whilst the LCV equations are explicit for x_j and y_j. This, of course, further complicates any attempt to derive a set of constraints on the a_1-a_4 and b_1-b_4.

The CATT can be conveniently written in matrix form as [12,15,13]:

$$T = \begin{bmatrix} t_1 & t_5 & t_9 & 0 & t_{11} & 0 \\ t_2 & t_6 & 0 & t_9 & t_{12} & 0 \\ t_3 & t_7 & t_{10} & 0 & 0 & t_{11} \\ t_4 & t_8 & 0 & t_{10} & 0 & t_{12} \end{bmatrix} \tag{2-12}$$

By writing the landmark co-ordinates (x_j, y_j) in matrix form as a rectangular 2x2 matrix:

$$X = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{pmatrix} \tag{2-13}$$

and similarly constructing X' and X" from the basis view co-ordinates (x_j', y_j') and (x_j'', y_j'') respectively, and then stacking the X, X' and X" in the joint image [26].

$$Y = \begin{bmatrix} X \\ X'' \\ X' \end{bmatrix} \tag{2-14}$$

Since noise processes are inevitable, given X, X' and X" over three or more points, the elements of the CATT are obtained by minimizing the squared error

$$\varepsilon^2 = \left\| T_i Y_i \right\|^2 \tag{2-15}$$

subject, of course, to the constraints (2-9) and normalization (2-10).

Finding the CATT is equivalent to finding, as far as is possible under affine imaging, the pose of the object as represented by X, relative to its pose in the basis views X' and X". Similarly, determining the LCV coefficients $a_1 \ldots a_4$ and $b_1 \ldots b_4$, amounts to determining the pose of the object, although in this case, as the trifocal constraints are not usually applied, the solution is, of course, not completely correct. In particular, it may admit solutions with small error ε for inputs X that do not correspond to a view of the rigid object represented by X' and X". In spite of these difficulties, the technique was proposed by Ullman and Basri [28] as one way of effectively finding the pose within the LCV approach. They also suggest a number of alternatives, including:

(i.) using a small number of corresponding features,
(ii.) searching for the coefficients,
(iii.) using a linear operator

Of these, the first (i.) is just a special case of the general solution to (2-6) in the least squares sense, designed (a) to reduce the computational burden when there is a library of possible objects (i.e. the technique is being used for object recognition) and, more importantly, (b) to attempt to circumvent the problem of requiring lots of reliable features correspondences. Unfortunately, at least four (that is, for the uncentred, affine mapping as in Ullman and Basri [28], or also four for the centred, overcomplete mapping as in Koufakis and Buxton [22]) correspondences are always required, moreover the reliability and accuracy of the method deteriorates sharply unless there are a sufficient number of redundant features to ensure that, for example, outliers due to incorrect correspondence can be identified, or occlusion effects allowed for [22].

The second possibility (ii.), to search, is of greater interest, though, since the parameters span an 8-dimensional continuous space, some sort of gradient descent must be used, to guide the search. There is the danger therefore of the process being trapped in local minima and terminating at a sub-optimal, incorrect solution.

It is important therefore if such a method is used that a good starting point can be found or that the behaviour of the system is not sensitive to the choice of starting point, i.e.

that there is a good "basin of attraction" within which the algorithm will converge to the correct solution. Since, it is almost impossible systematically to sample such a basin of attraction in an 8-dimensional space, it is important, and also to improve the convergence of the algorithm, to reduce the dimensionality whenever possible. The traditional approach (see for example, Ullman and Basri [28]) is to complete simple global features such as the centroid of the object of interest, and for example its area and several other central moments, its orientation (from the second order moments) etc. The rational is that such features will not be very sensitive to noise, image processing artefacts and segmentation errors. In our case, the object of interest is always centred in every image, so the centroid is always zero, but we made no attempt to use other global features to reduce the dimensionality of the search.

The final technique mentioned by Ullman and Basri, (iii) essentially introduces a type of projection operator that may be used for recognition of an object by looking for projection into a null space. It seems designed to avoid the problem of calculating the coefficients that represent the object's pose and is therefore of little help to us.

3 Algorithms Design for the geometrical approach

In section 2.1.1.2, we saw how Cootes et. al. [6] iteratively searched the image for new instances of the modelled object by enforcing the shape to be within the limits of the PDM built using the training set. Based on this idea, we perform a similar iterative search within the LCV, but enforcing a correct pose within the basis views and having the shape fixed. The search is divided into two separate steps: *image data interrogation* followed by *pose approximation.*

This chapter will also be separated in two sections, one for the data interrogation design and another for the pose approximation. But prior to those, the details of the input data of the system will be accounted for. Another section will be dedicated to clarify the concepts of model space and image space.

3.1 Input Data

The CATT used in the extended LCV method needs two basis views to be defined. These images must be sufficiently separated in view-space in order to provide a good representation of the object of interest. If they are not, determination of the pose, i.e. of the CATT, will fail. The system is provided with such a pair of line-drawing images, together with their corresponding landmark points coordinates, which represent an object taken under affine projection at -25 degrees and 25 degrees rotated about an axis vertically parallel to the image plane (Figure 3-1). Along with the basis views, there are a set of intermediate views separated 5 degrees from each other, forming a total of eleven line-drawing images. There will be no shape changes in the object.

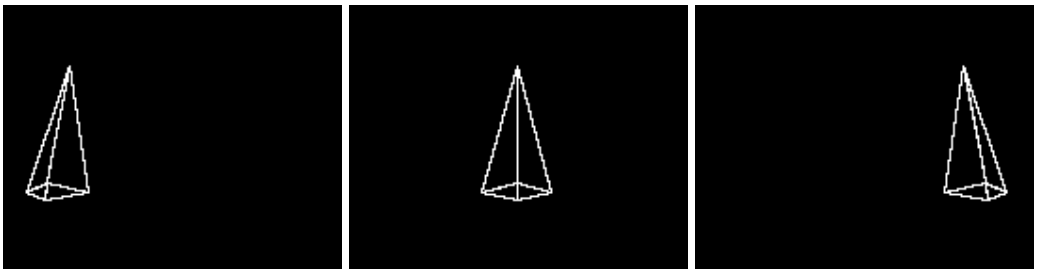


Figure 3-1. Line-drawing images of basis views (left and right) and frontal view (centre).

Our target object is a nose-like-object represented with six landmark points. The system will be initially designed to work with a simple kite instead of a nose, so that the program reaches a testable stage as soon as possible, as we will see in section 5.1. Figure 3-2 shows the structure, connectivity and order of the points for both the nose and kite objects.

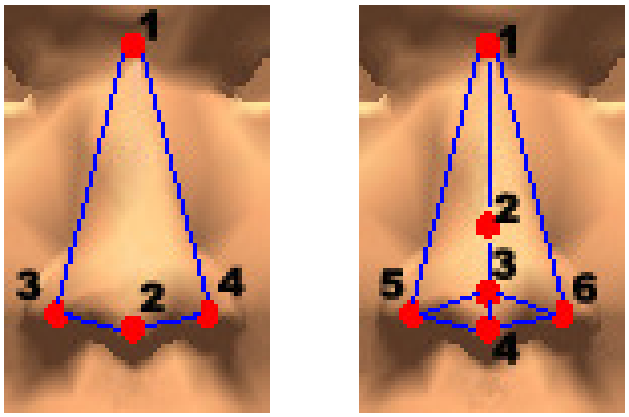


Figure 3-2. Kite (left) and Nose (right) structures.

For the system to start the searching algorithm, we also need an initial position, orientation and scale. We could use the frontal view (0-degree view) as the initial guess for the iterative algorithm or use some basic binary image processing to calculate a better guess from the given line-drawing. For example, we could follow the suggestions of Ullman and Basri [28] and use some prominent feature points of the object for early localization, or some low-order moment features to determine the orientation and scale of the object. Such information would be provided by second order moments. First order moments from which the centroid of the object may be determined well, of course, effectively have already been used in methods based on the CATT as the image objects always have to be centred at the origin before the CATT can be used.

3.2 Image Space vs. Model Space

An easy but important concept, which will appear throughout the algorithm design, is how and when the data is in image space or model space. Model space is when the set of coordinates describing the object are centred and scaled. Image space is when the data is expressed in the image coordinates.

Each input view has its own centre, calculated trivially when the system starts, and its own scale, calculated with respect to one of the images (for example, the first).
When searching the target image, the starting point of the search must obviously be expressed in image space. Initially, it is centred and scaled using the initial guess, but as the object's position changes at each iteration (hopefully towards the target image), the centre and scale changes accordingly.
It is necessary to express the points searched in model space in order to use the Centred Affine Trifocal Tensor in the pose-approximation step of the algorithm to enforce a valid pose. Thus, every iteration, after finding the new landmark points in the image and updating the centre and scale, the data is transformed from image space to model space to apply the CATT constraints, and back again to image space for the next iteration.

3.3 Image Data interrogation

The aim of the image data interrogation phase is to move the landmark points around the vicinity of the current estimates, so as to get a new object representation that is closer to the target object.

Firstly, we need to add several points in-between the original landmark points to avoid the search being ill-defined or getting stuck in a local minimum [1]. Two such points between each pair of landmark points seemed like a reasonable option. If we added only one, this middle point would probably not have enough force by itself to push the solution out of a stagnant situation. On the other hand, if too many such points were to be added, we would be removing importance from the original landmark points at the vertices of the object.

Once these subsidiary points have been added to the model, and starting from the un-centred and un-scaled guessed initial location, we iterate around the main program loop, as outlined in the following pseudo-code:

```
while (Not Converged)
{
      UnScale(start, scale)
      UnCentre(start, centre)
      points_found = ImageInterrogation(start, I)
      centre = Centre(points_found)
      scale = Scale(points_found)
      corrected_points = CorrectPose(points_found)
      start = corrected_points
}
```

For a convergence criterion, the root mean square (RMS) difference between the resulting landmark points found in any given iteration and the landmark points found in the previous one was used and the program stopped when this fell below some threshold. Note that this criterion means the search terminates when there is little change in the proposed pose of the object, and that there is no guarantee that the error between the current, synthesized view and the *target* will be zero or even close to zero. The algorithm may, therefore fail by getting stuck in an incorrect pose estimate that appears as a local minimum of the RMS error between the synthesized view and the target. Also note that the subsidiary points placed between the landmarks were not used in the calculation of the stopping criterion as their positions are determined by those of the landmarks.

We have also to decide the image space we will consider for the image data interrogation, as we cannot interrogate the whole image at every pixel, for every iteration. The search space must be limited somehow, two main approaches were considered:

- To search around the immediate neighbourhood of each point (in a limited area).
- To search along lines normal to the boundary of the model object (within a limited range). In particular, with this approach we would search perpendicular to the object boundary at the subsidiary points, and along the bisectors of the vertices in the case of the corner points, as shown in Figure 3-3. This is the most common approach used in the literature [7,1], probably because a one-

dimensional search is much faster. There are several issues that must be addressed, including:

- If the search fails for a given point, we could alternatively try with the neighbourhood approach. Or, as we will see in the next section, ignore this point if we are using a least squares solution in the pose approximation step.
- When a point has more than two boundary lines or edges connecting it to other points, we can either consider only those bisectors formed by strictly adjacent edges or take into account all the possible combinations (see the dashed lines in Figure 3-3). The second option was found to give better results.
- We could extend the search also along the edges connecting to other points (see dotted lines in Figure 3-3). This turned out not to be very helpful.

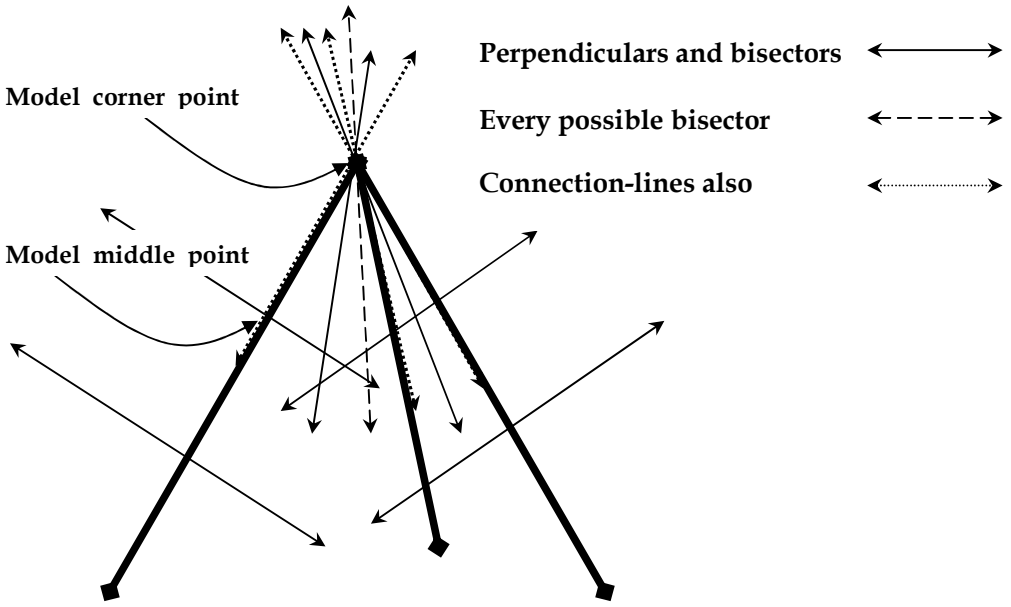


Figure 3-3. Perpendiculars, Bisectors and more bisectors alternatives at the vertex of the object.

Once we have defined the search space, we just have to find the closest intersection between the lines of search and the line-drawing image. In order to make the system easily testable, a pre-step was made before looking for the intersections in the line-drawing image. The target points (the ones we are trying to find) were used directly to

calculate the intersections geometrically without using the line-drawing at all. This serves as an errorless simulation of the image data interrogation to test the pose approximation alone.

3.4 Pose Approximation

On its own, the data interrogation step introduced in the previous section, would not make more than one iteration, because the points found would always lie on the target, so that in the second iteration, exactly the same points would be found, making the algorithm stop. Furthermore, it would be theoretically incorrect. The points found will not obey the constraints imposed by the CATT, so that it would not be a correct pose-instance that could in principle be generated from the basis views.

For these reasons we must enforce "correctness" of the set of landmarks found by the data interrogation. Once again, the algorithm was designed in several incremental steps depending on how we assume a "correct" pose-instance is generated:

3.4.1 An analogy using one view

It is easiest to explain how the model is updates by assuming that, for the moment, the image is of a flat object, so that it could be generated from a single basis view. In general, the required transformation would be a homography (a linear transformation in projective image space) [19], but under weak or para-perspective, a simple affine transformation of the image co-ordinates suffices. It is also useful, before going into the three-dimensional multi-view models, that the system implemented is verified for single-view transformations. Thus, we suppose that

$$X = A\,X', \tag{3-1}$$

where X' represents the landmark points in the basis view, and X represents them in the image to be generated.

For all of these, the transformation may be calculated via standard least squares by minimising the error:

$$e^2 = \| X - AX' \|^2 \quad , \tag{3-2}$$

and the transformation matrix calculated, provided we have enough landmark points in a non-degenerate configuration, from:

$$A = X\,(X')^{\dagger} \, , \tag{3-3}$$

where $(X')^{\dagger}$ is the pseudo-inverse [16]of matrix X'

As introduced above, transformation **A** can be:

- **Similarity transformation:**

  When the model is only allowed to scale, shift and rotate (around the camera axis) we have a similarity transformation. This is what Cootes and Taylor did in their shape modelling, as we saw in section 2.1.1.2. This transformation has three degrees of freedom and the matrix A may be presented as

$$A = \begin{bmatrix} s\cdot\cos(\theta) & s\cdot\sin(\theta) & t_x \\ s\cdot\sin(\theta) & s\cdot\cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3-4}$$

  Since, in this case, the elements of *A* are not independent, the solution that minimizes the error (3-2) is no longer simply given by the pseudo-inverse. However, the problem may be solved straightforwardly by elementary algebraic means even though the solution may sometimes appear a bit lengthy. Since we have not used this transformation in our work we do not reproduce it here.

- **Affine transformation:**

  If the object is a large distance from the camera compared to its size then, as noted above, the mapping will be affine in the camera co-ordinates. This allows more complex transformations, such as shears, than the similarity transformation

described above. Such an affine transformation has six degrees of freedom and may be represented compactly by the matrix

$$A = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3-5}$$

- **Plane to plane homography transformation:**
  A more general transformation, with eight degrees of freedom, that allows for perspective effects i.e. may be used when the object is very close to the camera or there is a very large range of depths across the object in comparison to its closest distance to the camera. In this case, all nine elements of the transformation matrix are.

$$A = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ e & f & g \end{bmatrix} \tag{3-6}$$

if X is represented in homogeneous co-ordinates, i.e. each landmark co-ordinate (x_i, y_i) us written as the column vector $X = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

3.5 Stereo Transformations

As noted previously, the transformations based on a single image are only valid when the object is flat. For 3D objects, the new, target image must be composed from two or more others. In principle, two images would enable us to reconstruct the object's shape in 3D which could then be projected into the view of the new, target image. Under affine (weak or para-perspective) imaging conditions these transformations are all affine, so in general, there is an affine mapping form the two initial images, known as basis views, to the target.

There are two approaches considered:

- **Direct Affine transformation**

  The simplest such mapping may be written as a pair of equations linear in the image co-ordinates (x_i, y_i), $(x_i^{'}, y_i^{'})$ and $(x_i^{''}, y_i^{''})$ of corresponding landmark points in the target image (x, y) and the basis views, $(x^{'}, y^{'})$ and $(x^{'}, y^{'})$, respectively.

$$\begin{aligned} x_i &= a_1 x_j^{'} + a_2 y_j^{'} + a_3 x_j^{''} + a_4 y_j^{''} \\ y_i &= b_1 x_j^{'} + b_2 y_j^{'} + b_3 x_j^{''} + b_4 y_j^{''} \end{aligned}, \qquad (3\text{-}7)$$

  It was already discussed in section 2.1.2 how this transformation is useful but is not correct, as it suffered from over-completeness. It was decided to include it, nonetheless, owing to its simplicity and common use among fellow researchers, so that it can be used as an extra step to test the system.

- **CATT**

  This is the final step for this first half of the project. It basically consists of enforcing a correct pose of the object described by the set of landmarks found in the search phase. This is achieved by applying the CATT constraints (reviewed in section 2.1.2.) to the points found, and obtaining the CATT parameters that defines the pose of the object. The details of the CATT design will not be discussed here, as it was taken from the work already done by Dias and Buxton [11,12,15].

  In section 3.3, we mentioned the possibility of not finding a landmark in the data interrogation step, and how this problem could be addressed by ignoring the pose approximation step. Indeed, the parameters of the CATT can be obtained with as few as three corresponding points (as explained in section 2.1.2), so that we can ignore any landmark not found in the image as long as there remains three or other points that were found. Of course, the lack of redundancy will diminish the accuracy of the method, but this is of little importance if we assume that not many points will be omitted.

4 Implementation for the geometrical approach

The system was developed using *MatLab 6*. The main reasons why this platform was chosen were that it allows for a straightforward implementation of many of the methods needed, which may be based on matrix operations. In addition, the code is easily amended, which is a very desirable property in this type of research projects. The other main reason was that the project builds on the work of Dias and Buxton, and some of the code used, which was provided by Dias, is in *MatLab*. *MatLab* has the inconvenience of producing slow programs, but speed is not a requirement in our system.

We have so far presented some background information and a series of algorithms, up to the point where the reader should now be able to implement a system similar to ours. This chapter is concerned with the implementation details of our system. In particular, we will illustrate how the data is processed by means of a data flow diagram, then the main functions will be listed hierarchically, and finally some functions details will be given.

4.1 Data Flow

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. In the following data flow, which is intended to show the main, iterative part of the procedure. Some details such as centring and scaling are obviated.

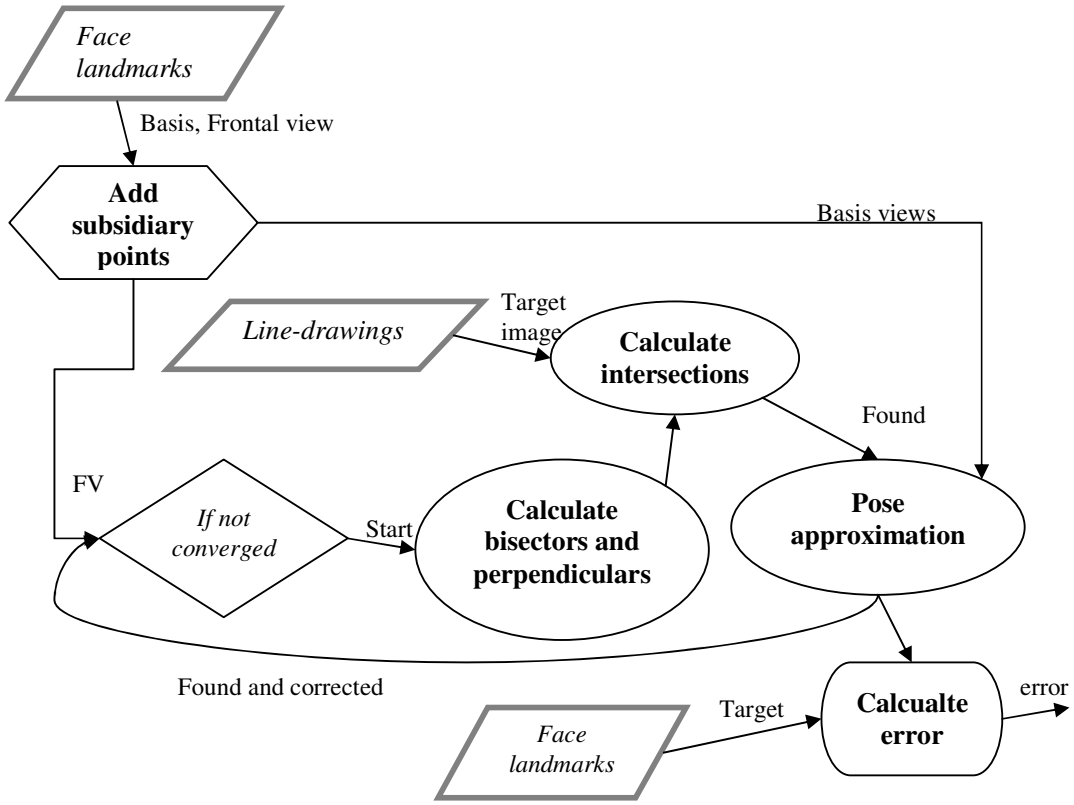


Figure 4-1. Data flow diagram.

4.2 Main functions description

Given the above data flow, we identified the following functions, listed hierarchically in accord with the data flow, that are required in order to implement the geometrical search process described in the design chapter.

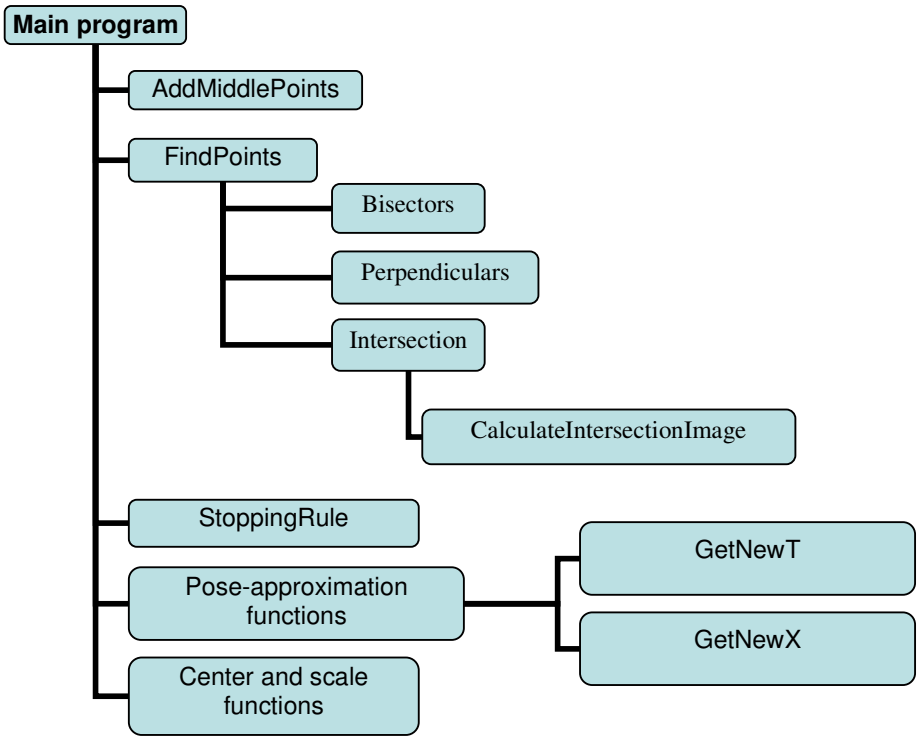


Figure 4-2. Functions Hierarchy.

4.2.1 AddMiddlePoints

This function receives as inputs the set of landmarks and the way they are connected amongst themselves (connection table). It outputs the set of landmarks including the intermediate subsidiary point generated and a look-up table that indicates, for each pair of subsidiary points (assuming there are only two between each pair of connected landmarks), from which pair of landmarks they were generated. This look-up table will be needed in the function Perpendiculars. The actual generation of the intermediate subsidiary points is trivial and will not be discussed here. The generation of the look-up table is more subtle.

If we take the kite structure (see figure 3-2 in section 3.1) there will be four connections between the landmarks, and, if we assume there are two subsidiary points between each pair of landmarks, for example, the output to the function after adding the subsidiary points will be a vector: $\langle 1 \quad 2 \quad 3 \quad 4 \,|\, 5 \quad 6, \quad 7 \quad 8, \quad 9 \quad 10, \quad 11 \quad 12 \rangle$, where the first four are the original landmarks and the rest are subsidiary points, grouped logically in

pairs that belongs to the same connection-line. Thus, the look up table was designed to be:

$\begin{bmatrix} 1 & 3 \\ 1 & 4 \\ 2 & 3 \\ 2 & 4 \end{bmatrix}$, so that, for any subsidiary point i in the output vector, we may calculate the connection-line they belong to by: *div((i-n),2),* where n is the number of original landmarks (which can always be calculated from the size of the connection table). And then, we can use this connection number as an index to the look-up table.

4.2.2 CATT

There are two functions that implement the detailed form of the CATT:

- T = GetNewT(basis1, basis2, X), a function that returns the closest twelve parameters of the CATT that permits a valid linear combination of the two given basis views into the given instance.
- X=GetNewX(basis1,basis2,T), this function returns the instance defined by the two given basis views and the twelve CATT parameters in **T**.

Both functions were provided by Ben Dias, being the result of his work.

5 Testing for the geometrical approach

The testing phase of this project is probably the most important one because we are trying to see if the methods implemented are correct and applicable. As the system was incrementally developed, there is a testing phase after each of the steps made. This way, we do not move on to the next stage of the project unless the results obtained are as expected, or at least acceptably coherent with the underlying theory.

To carry out the testing, we supervise the system using the correct output of the search, which is the set of landmarks of the object, or the pose (defined by the parameters of the transformation). Thus, for each iteration of the system, we can determine the distance between the current landmark points found and the correct ones. This distance is not to be confused with the difference between the estimates of the positions of the landmark points produced at each iteration tat was used as the convergence criterion (3.3). These are different as it is possible for the system to converge to a result far away from the correct one.

The testing is divided in two main stages. First we work with strictly geometrical point data, so that we use the target object's landmark points instead of the line-drawing to perform the search. In the second stage, the line-drawings are used and we will move on to working with implicit three-dimensional representations of the objects.

5.1 Geometrical

This testing stage was also divided in several smaller steps. We started by working with a reduced structure of the nose; taking into account only the exterior landmark points (see section 3.1), this makes some implementation details easier, and serves as a first checkpoint for the system testing. This framework was extensively tested, as it is the very first stage of our system.

5.1.1 Functionality tests

After the implementation of each system's functionality (that might require more than one function) some tests were carried out to test their validity using some simple inputs for which we know the answer. These were some of the tests carried out:

- *Including subsidiary points*. Given any two points, check that the intermediate subsidiary points generated are collinear with each other and with the original landmarks, and also that the minimum distance from each subsidiary point to the landmarks is equal to the minimum distance to the other subsidiary point (for two subsidiary points).
- *Bisectors and perpendiculars* calculation. This function takes a set of points and calculates either its bisector, if it is a landmark point, or else its perpendicular (in section 4.3 it is explained how this is done). The resulting lines are given as a triple (a,b,c) defining the line ($ax + by + c = 0$). To test this, a simple square structure was given as input and then drawn together with the output lines.

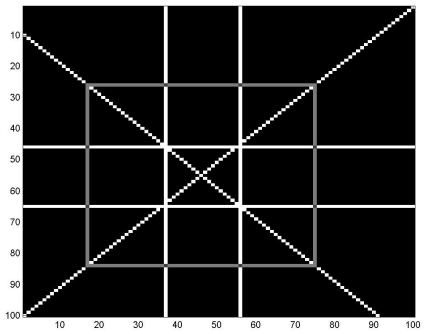


Figure 5-1. Test for bisectors and perpendiculars.

- To test the function that transforms one set of points into another, a small script that transforms a simple known structure and then draws the results, was written and used to carry out some basic transformations for which the results are intuitively obvious or easy to work out by hand.

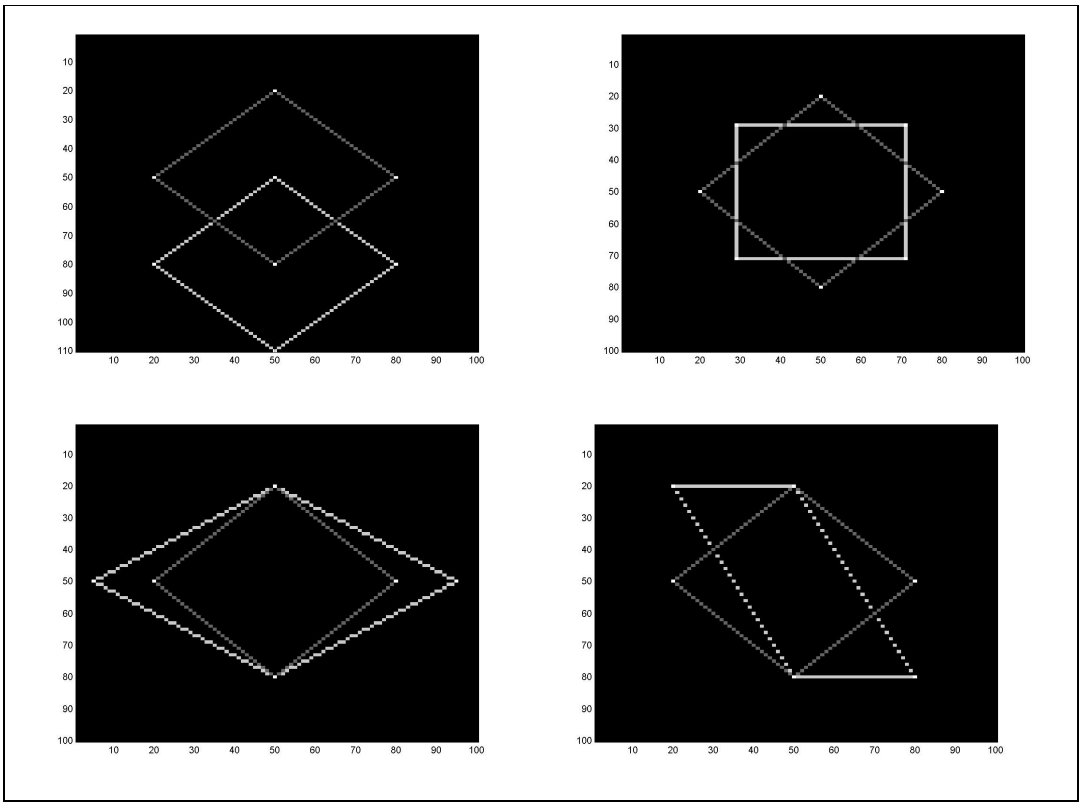


Figure 5-2. Tests for simple transformations: translation (top left), rotation (top right), scale in the x direction (bottom left) and shear in the x direction (bottom right).

- After tests of the kind described above were passed successfully, the searching function was tested by starting from a simple structure, and targeting also a simple structure specifically built so that the expected outputs are easily checkable by eye. In the last of the following tests, the target was scaled by a half and translated so as to make the search fail for some of the subsidiary points.

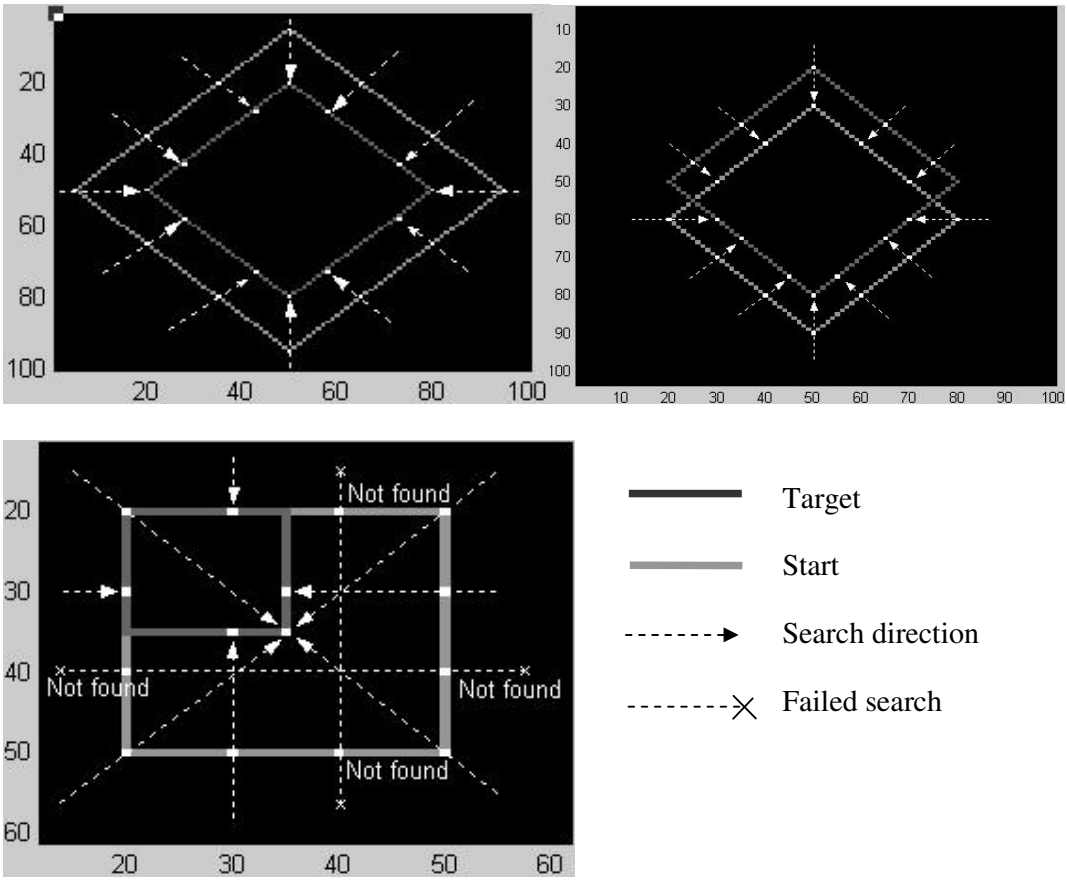


Figure 5-3. Testing the geometrical search: (a) Target slightly scaled (top left), (b) Target slightly translated (top right), and (c) Target scaled by a half and translated.

5.1.2 Integral tests

Once each individual function was known to work properly, the first set of tests involving the whole system was carried out to compare some search alternatives (see 3.3 and figure 3-3 for details). These were assessed by performing a large set of tests using the frontal view as the starting points for the search, and trying to find randomly affine-transformed instances of the frontal view. In this manner, a percentage of success for each of the alternatives was obtained. The advantage of this testing method is that we can sample what is happening within a high dimensional parameter space.

The transformations used for all of the tests were selected uniformly at random within the range represented by the matrix $\begin{bmatrix} 0.5..1.5 & -0.5..0.5 & 0 \\ -0.5..0.5 & 0.5..1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. One thousand such runs were made, for a maximum of 450 for each single search, and a convergence tolerance of 10^{-5}. At first, two alternatives were tested which did not utilise any subsidiary landmark points. The results are summarised in table 1.

Alternative tested	**percentage of success**
Using bisectors only	41%
Using connections between adjacent corner points as an extra search direction	42%

Table 1. Success rate for two alternative procedures, neither of which used subsidiary points.

These results are not very encouraging. Using the extra search directions represented by the connections between the landmark points does not seem to help much in the search.

To clarify this behaviour of the system, some specific cases were investigated in order to see what was actually happening for both alternatives. The image-results, zoomed to the region of interest, show the set of points found for all iterations. The points joined by lines form the starting structure and the big white points form the target structure. The plots given to the side of each image diagram are error curves (the RMS distance from the estimated landmark positions to the correct ones) as a function of the iteration number. Some positive (good) and negative (bad) results are shown and briefly discussed here. But the complete set is presented in appendix A for reference.

a. Using bisectors only

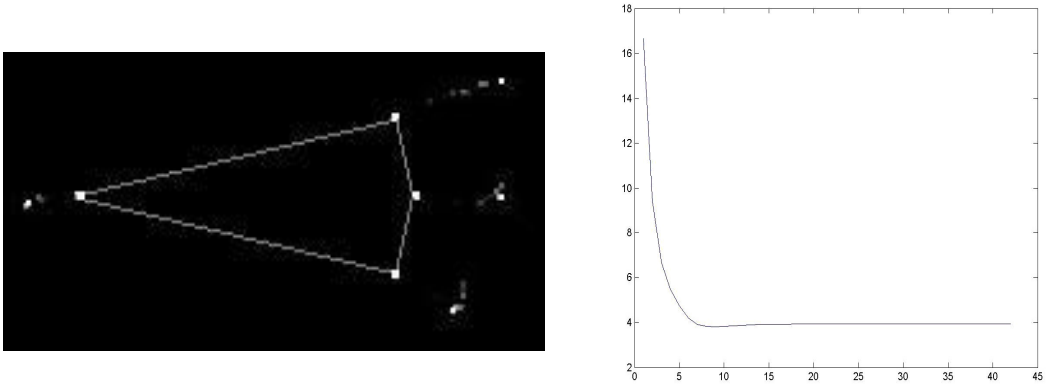


Figure 5-4. Positive (good) result, obtained by using only the bisectors.

The RMS error converges to 4, which represents convergence to within 2 units on average. For comparison, the length of the kite shape is approximately 70 units.

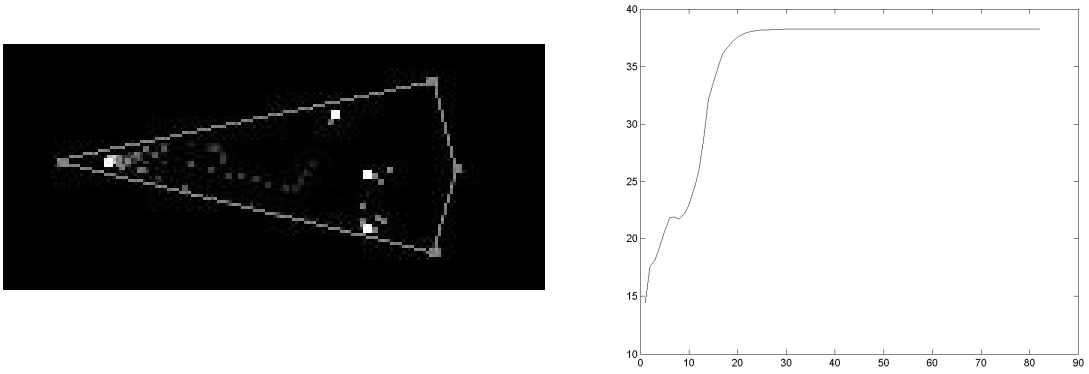


Figure 5-5. Negative (poor) result, obtained using only bisectors.

b. Using connections between adjacent corner points as an extra search direction

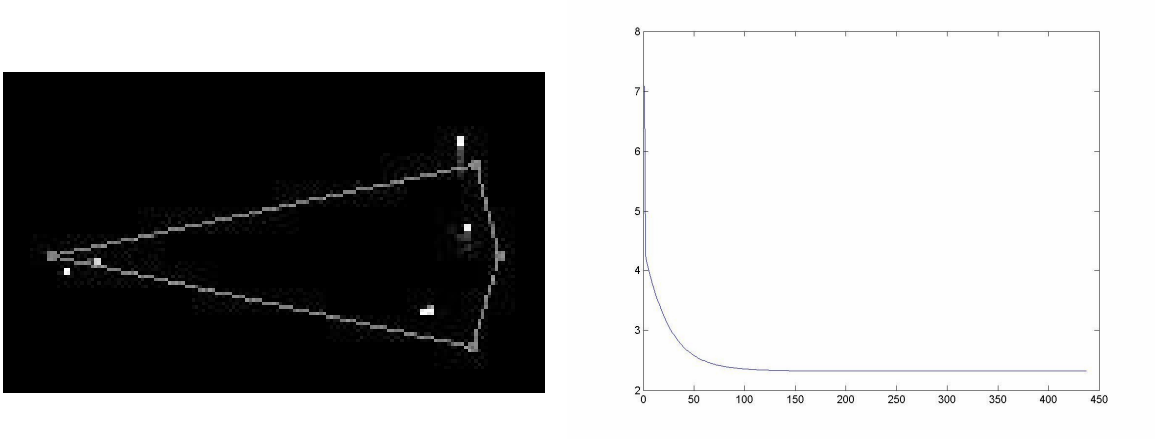


Figure 5-6. Positive (good) result, obtained using the connections between landmarks as extra search directions.

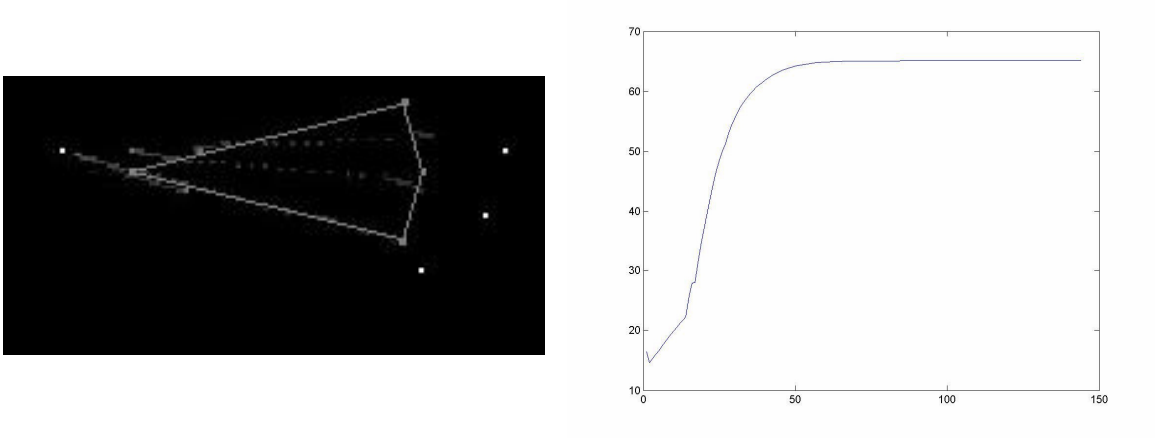


Figure 5-7. Negative (poor) result, obtained using the connections between landmarks as extra search directions.

In summary, it seems that the search cannot cope with many transformations, either because some point misses the target, or because a point is found too far away from its corresponding target point, so that the system unable to advance towards the correct solution and gets stuck.

At this point, the intermediate subsidiary points were introduced into the system (see section 3.3) to try and solve this problem. As before (i.e. using the same transformation

and parameters), some random tests were performed to obtain the success percentages shown in table 2.

Alternative tested	**percentage of success**
Subsidiary points added (no extra search direction)	84%
Subsidiary points added and extra search direction	79%

Table 2. Success rate for the kite shape when the subsidiary landmark points were added:

(i) without including the connections between the original landmarks as extra search directions, and (ii) when these extra search directions were included.

The subsidiary points effectively met their purpose, pushing the search towards the correct solution. As previously, some specific cases are presented below to confirm the good results obtained. Before doing so, it is worth noting that, in this case, adding the connections between the original landmark points as extra search directions reduces the success rate of the system.

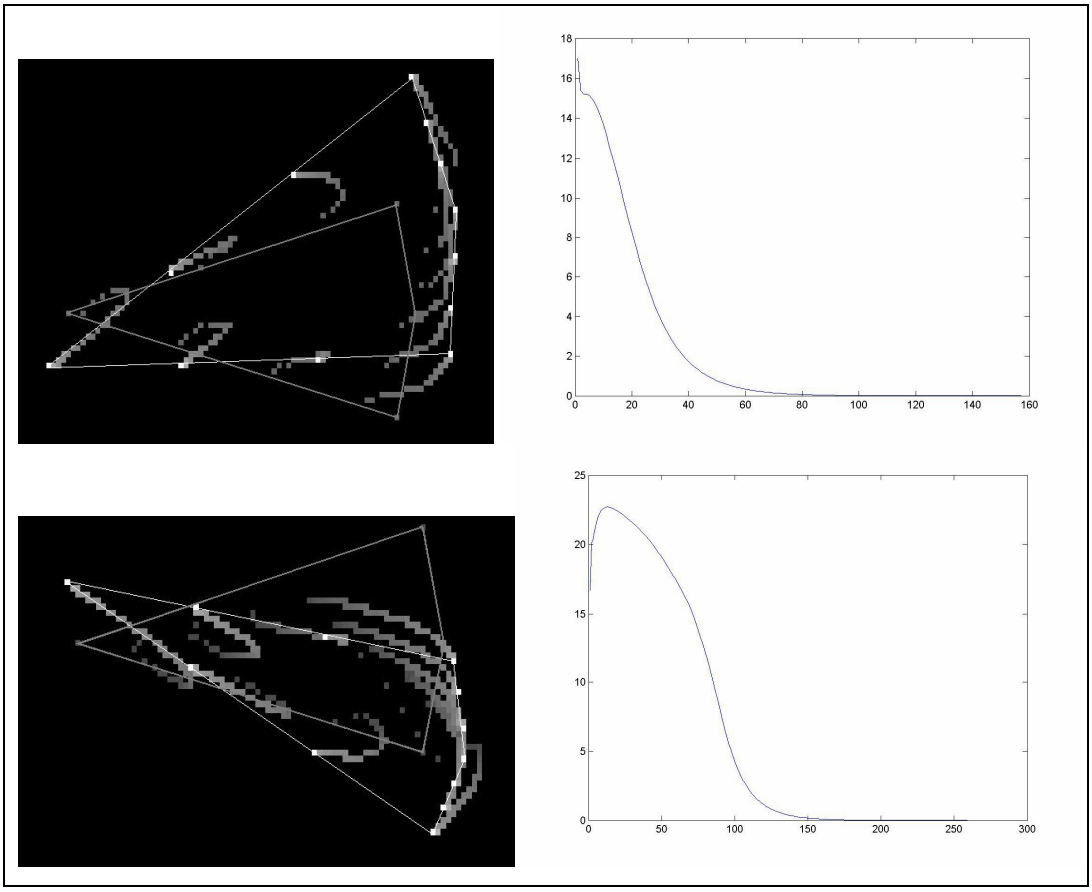


Figure 5-8. Positive (good) result, with subsidiary landmark points included.

However, in some cases, the search still failed, as illustrated by the following negative example:

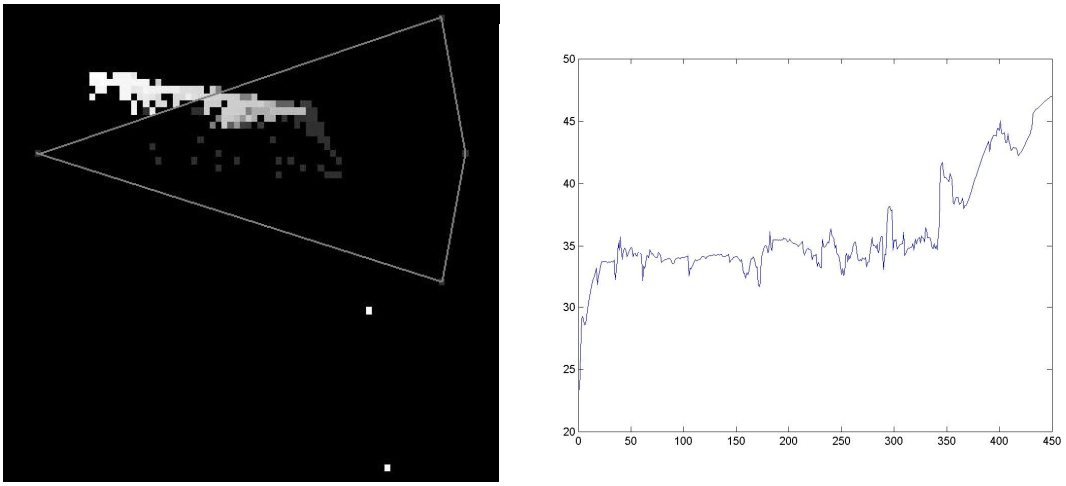


Figure 5-9. A negative result, which failed to converge even though the subsidiary points were included.

After these tests, we commenced working with the complete nose shape. The system was harshly tested to deal correctly with this structure. This step does not impose any further challenge to the system other than that it has to work in the same way as previously but for more landmark points.

When we consider the results of these tests taken together, we can confirm that the system works and that the method designed is implemented correctly, at least in the geometrical sense.

5.2 Minimal profile (line-drawings)

Given that the system worked satisfactorily for the landmark point data, we moved on to the next step of testing whether it would work for geometrical data derived from a line drawing as described in chapter 3.

5.2.1 Functionality tests

There were two developments of the system at this stage: using line-drawings as the target objects to be found, and considering a "correct" object instance to be the combination of two basis objects.

The first development is merely tested functionally, exactly as we did for the point geometrical case in section 5.1.1., giving as input to the system a drawing of some simple object structure, and starting the search from an object structure that is a known transformation of the object in the drawing.

For the second development, some functions involving centring and scaling are necessary (see section 3.2. for details). These functions were tested by giving them as input some points obtained by sampling a circle of radius 1 and checking that the resulting centre is the centre of the circle, and that the scale is half of that obtained by repeating the test for a circle of radius 2. The actual functions that deal with the calculation of the CATT parameters were not tested, because they were provided as noted in section 4.2.2.

5.2.2 Integral tests

By using some of the input line-drawings taken at different view angles, the system is observed to see if it is capable of finding such structures. This testing approach lacks generality, as such a set of line-drawings does not include all the possible difficulties. Nevertheless, it gives us a good indication of how the system is performing under "normal" conditions.

The two stereo transformations approaches, described in section 3.4.2., will be tested separately. In turn, each of these was divided in two set of tests: one with the initial centre and scale of the object, represented by the line drawing, known; and another with the initial centre and scale estimated approximately from the line-drawing itself.

For all of the following tests, two subsidiary landmark points were added between each pair of the original landmark points whenever the original landmarks were structurally connected. The system was allowed to run up to a maximum of 100 iterations and the convergence tolerance was 0.3 (see section 3.3.).

Only the test using one of the basis views as the target is shown here for each alternative, this is enough to see if the system is behaving in the manner expected, even if the results were not good. The whole experiments of tests will be discussed in the results chapter and the details given in appendix A.

Affine approach

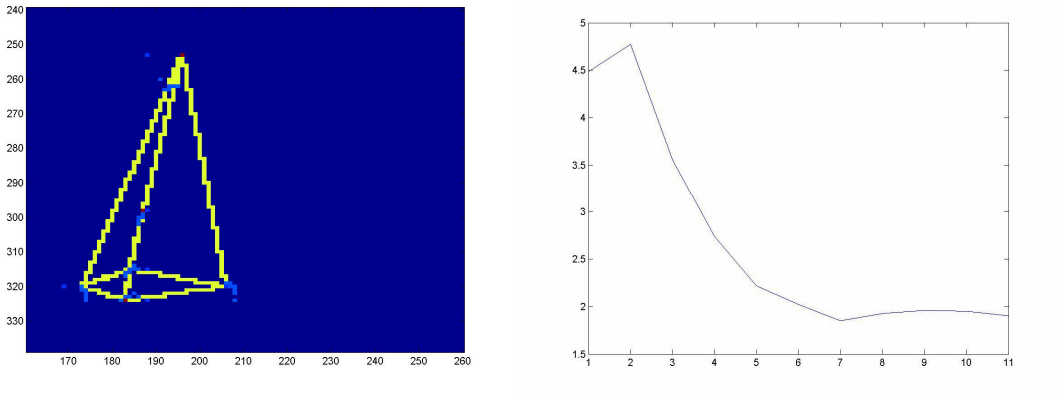


Figure 5-10. Test of the affine LCV transformation approach, for known centre and scale of the target object.

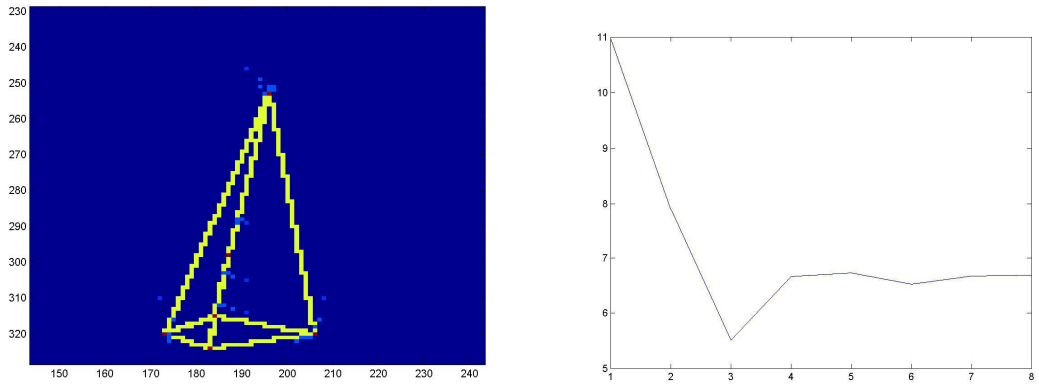


Figure 5-11. Test of the affine LCV transformation approach, when using an estimate of the centre and scale of the target object.

CATT approach

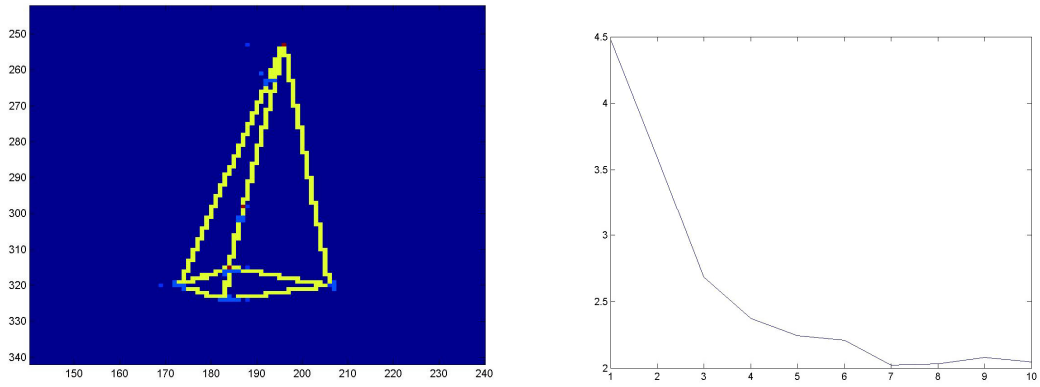


Figure 5-12. Test of the approach based on the CATT transformation, for known centre and scale of the target object.

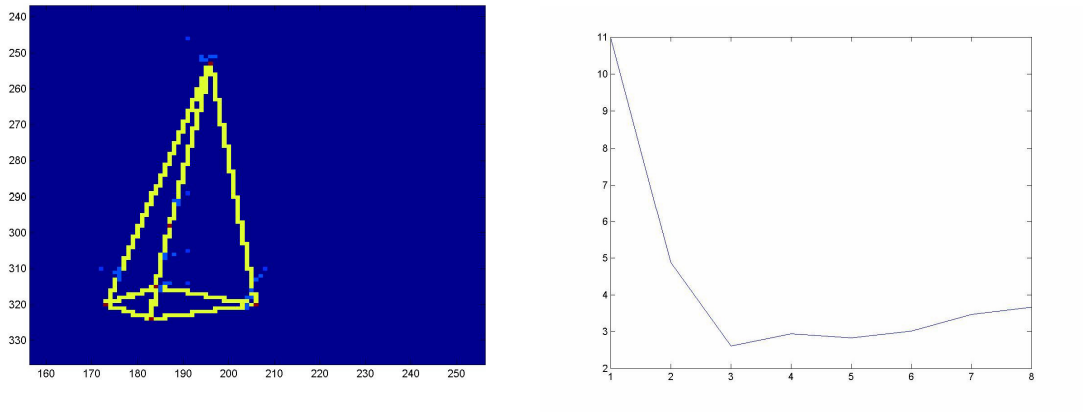


Figure 5-13. Test of the approach based on the CATT transformation, when using an estimate of the centre and scale of the target object.

6 Results for the geometrical approach

From the tests summarized in the previous chapter we can conclude that the system developed correctly implements the method designed. The results presented in this chapter are intended to enable us to decide whether the method proposed is applicable or not.

This chapter is divided into two sections. First, results are given for the kite-object, warped under single-view transformations, and using purely geometrical point data for the image search are presented. Following this, stereo transformations of the nose-like object are used with line-drawings of the target object as input.

6.1 Single-view transformations

The percentages of success that we saw in the test chapter, are not a very useful measure of the system's performance because they depend on the range of values over which we allow the parameters to vary. In fact, what we would like to determine is precisely the range of values the system can cope with. The space of all possible parameters is too big to be able test all the possible transformations (in the affine case, for example, it is a six-dimensional space), so we need to choose significant one-dimensional subspaces of it on which to test the system. This way, we can span the space of possible transformations in a controlled manner. If the errors for each of these subspaces are plotted, we can effectively assess the accuracy and robustness of the system by analysing the depth and width of the observed *basins of attraction* i.e., the interval of parameters values for which the system error is below a certain threshold.

To build these basins of attraction, some representative parameter is varied, thus defining a one-degree-of-freedom set of transformations. Then, using the frontal view as the target object, we start the search from the landmarks corresponding to the frontal view but warped by means of such a transformation. The basins of attraction were determined in this manner for the following types of transformations:

Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$, varying θ from $-\frac{\pi}{2} \ldots \frac{\pi}{2}$, in 20 equally spaced steps.

Translation in x direction

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$, varying t_x from -10…10, in 20 equally spaced steps.

Translation in x direction

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$, varying t_y from -10…10, in 20 steps as above.

An extra translation in a random direction was also performed.

Shear in x direction

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$, varying *b* from -3...3, in 20 equally spaced steps.

Shear in y direction

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$, varying *c* from -3...3, in 20 steps as above.

Scale in x

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$, varying *a* from -3...3, in 20 equally spaced steps.

Scale in x

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$, varying d from -3...3, in 20 steps as above.

In addition to each of the transformations, random noise could be added by moving the initial landmark points a little. A new set of basins of attraction were built for each of the transformations listed above, adding a random offset to the positions of all the landmark points. The amount of offset added went randomly from -5 units to 5 units.

The results obtained (which are again presented in detail in appendix A), are quite encouraging, especially those for the rotation transformation as illustrated below:

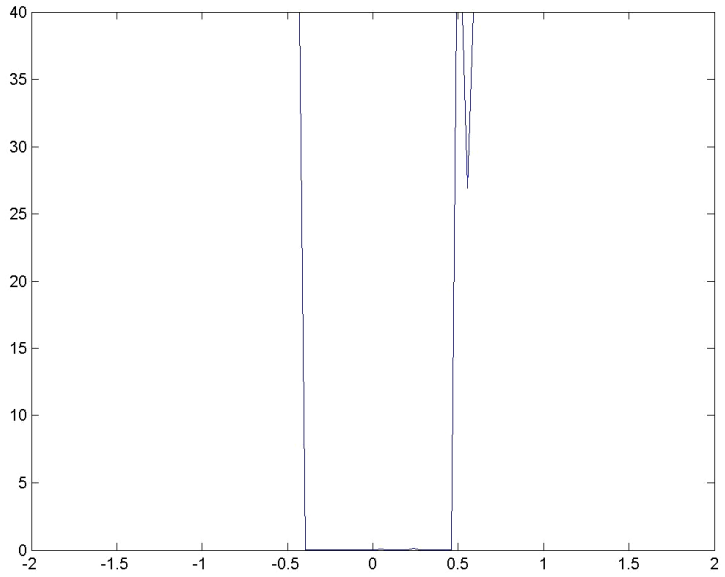


Figure 6-1. Basin of attraction for a rotation without added noise.

This specific basin of attraction shows how the error is almost zero for starting objects that are rotated with the range -24 to +26 degrees with respect to the target solution.

The basins of attraction obtained confirm and extend the hopeful results we obtained when testing the system (section 5.1). This means that, at this point, we can say that not only is the system correctly implemented, but that the method and approach chosen

seems able to cope with an acceptable range of different transformations as long as the image object is assumed to be a shape that could be (and was) generated by a linear transformation of just one other view of the object. We can therefore safely advance to the next stage of the project, where the pose will be varied by means of a stereo transformation.

6.2 Stereo transformations

In the testing phase of the project, some tests using the set of line-drawing as inputs were performed. To complement these tests, basins of attraction are again used to determine how much a target object (for example, the frontal view) can be translated from itself and the search still performs successfully.

The results for a random translation are shown in figure 6-2.

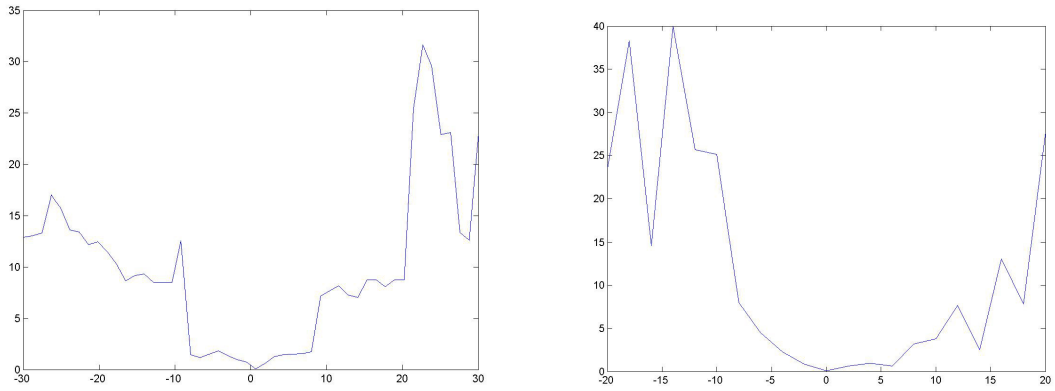


Figure 6-2. Basins of attraction for the stereo affine approach (left) and the CATT approach (right), for a randomly chosen direction of translation.

Although not as good as the results obtained when the target object was produced by transformation of a single view of the object, these results seem promising enough to enable the project to continue through its planned path: to try to deal directly with grey-level images instead of using geometrical points and line drawings as input.

7 Grey-level approach

Once the pose-alignment algorithm is proved to work correctly using geometrical point data and line drawings as input, we can extend the minimal profiles represented by the idealized line drawings to grey-level profiles

7.1 Background

In this section, we will describe the techniques used regarding the grey-level search, together with comments on the relevant work done in the field. This chapter will also include a brief review of the work done in lip reading, as this was the main motivation behind the project.

7.1.1 LCV

Ullman and Basri [28] expressed 3D objects as linear combinations of a small number of basis images, but they only worked with geometric information. In their work, they found that in general for objects with soft (curved) boundaries undergoing rigid transformation in 3D and scaling, three basis views of the object are required to be able to reconstruct novel intermediate views of the object. However, Koufakis and Buxton [22] obtained results that, for objects with weak boundaries so that the points on the object in 3D corresponding to the landmark points remain fixed in the object, suggested that with only two basis views a good approximation can be obtained. Hansard and Buxton [18] also worked with two-basis views for practical reasons, although they consider the use of more, for example, they point out that five or more are needed to parameterise a video sequence.

Koufakis and Buxton [22] modelled human faces for video compression purposes by using a hybrid approach that modelled the eye and mouth regions of face images separately by means of the Principal Components Analysis technique and a linear combination of three basis views to represent changes of viewpoint of the face as a whole. The eye and mouth regions were thus treated as flexible objects whilst the rest of

the face was treated as an otherwise rigid object. The drawback of their method is that it is only strictly applicable to objects whose shape changes are localised [11,15].
In the method of Koufakis and Buxton [22], the texture of the new face view is reconstructed using the texture of the basis views and a weighted interpolation technique with weights related to the linear combination coefficients. This method is later used by Hansard and Buxton [18].

To reconstruct the facial feactures, Koufakis and Buxton [22] manually placed control points on prominent facial features and divided the face area in the basis views and each novel view into triangular regions. Once the triangles are built, the novel face image can be reconstructed, for every pixel, by a technique called *piecewise linear mapping* originally developed by Goshtasby [17].

7.1.2 ASM

In section 2.1.1.2, we saw how the image was searched in the vicinity of each model point to find new instances. In this data interrogation procedure, Cootes and Taylor [6] relied on the object having strong edges, but the use of grey-level models seems a better approach [5].

In [4] Cootes et. al. compare the use of simple correlation procedures to the use of statistical grey-levels models combined with a multi-resolution technique. They claim that the second approach can more accurately locate objects and identify them, for example in the context of Printed Circuit Board inspection.

7.1.3 Lip Reading

There has been some research on visual lip reading using the ASM as the main tool for recognition. Two pieces of work will be briefly discussed.

Active Appearance Models (AAM) [3] model both shape and greylevel appearance in a single statistical model, so that the search for features (using standard ASM) and the

image greylevels are modelled together. Matthews et. al. conclude in [25] that this approach (AAM) is superior to other low-level, pixel-based, approaches in a lip-reading comparison context.

Luettin et. al. [24] model the deformation of the lips by means of an ASM based on *a priori* information taken from a database. The database contains examples taken from a lot of talkers and taken under different lighting conditions. They consider the use of grey level profiles an appropriate way to model dominant image features. They use one-dimensional profiles, but concatenate the individual profiles of all the points of the image and then perform PCA on the global profiles obtained. Then, they calculate the principal modes of profile variation, and use them together with the mean profile to approximate any profile of the training set. This approach assumes that the deviations of profiles at different model points are correlated with each other.

As the lip-reading scenario is merely the motivation for this project and we were unable to reach the point where we could, for example, experiment with lip data, we will not go further in this very brief review. For the interested reader, however, we note that for further reading, Simon Lucey [23] reviews extensively the two major problems for modern speech processing: the extraction of pertinent visual features, and the integration of acoustic and visual speech modalities.

8 Algorithm Design for the grey-level approach

In this chapter we will describe the main features of the design of the procedure used for the grey-level search. Since the technique builds on the methods established in chapter 5 for the geometrical cases, we shall concentrate here on the new aspects concerned with the grey-levels.

8.1 Input Data

The system is provided with grey-level images of a synthetic head model taken from different viewpoints under affine conditions, together with their corresponding set of landmark points.

As in section 3.1, the system needs a pair of basis views and a frontal view (FV) to initialise the system. Unlike in the geometrical case, we will use the rest of the set of images to test the performance.

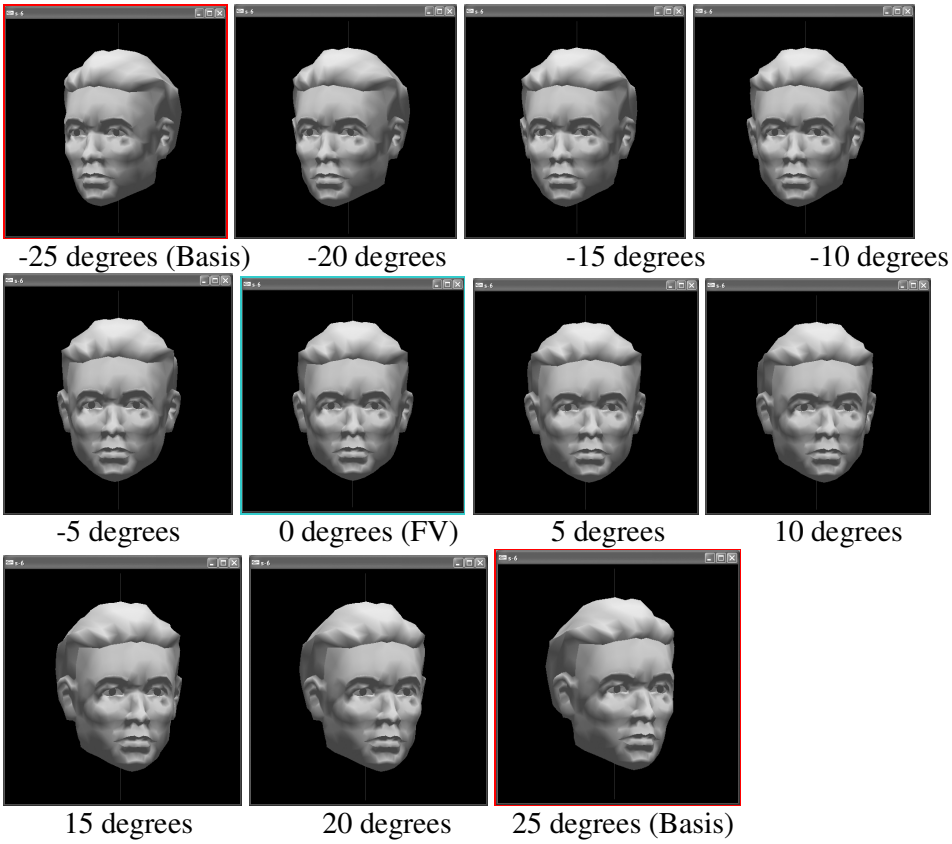


Figure 8-1. Training set.

As in the line-drawing approach, we need an initial orientation, position and scale, but this time we cannot make a guess from the target images, and rely instead on the frontal view.

8.2 Image Search

The approach, as a whole, is, as in the previous part of the project, an iterative method composed of two phases: an image search followed by the pose-approximation. In this part of the project only the first phase is changed. The pose-approximation is identical to that described previously for the geometrical part and line drawing data. The innovation of the second phase is that we use the grey-levels of the target image to search for "better" landmark positions instead of relying , for example, on the edges of the object in the image as was done in the first part of the project by using ideal edges, i.e. line-drawings.

During the course of our work, the grey-level search strategy had to be completely redesigned, as we will see in the testing chapter, so that two different methods were effectively designed. In the following subsections both approaches are described.

8.2.1 First Approach

The first thing that must be done is to build the *model profile.* This could be done by building the whole *joint image* (using the current pose) and then extracting the profiles from it. Although this is conceptually straightforward, it is computationally inefficient and wasteful, so a less complex alternative was chosen. The alternative consists of computing first the *basis profiles* (extracting them directly form the basis images), and then joining the profiles into the *joint profile*.

The profiles are built by interrogating the image along the bisectors/perpendiculars and up to a certain distance. The size of the basis profiles are calculated simply by transforming an auxiliary segment taken from the frontal view, that is aligned along the bisectors/perpendiculars and has a fixed size, to each of the basis views (see figure 8-2), and computing the distances between the end-points of the resulting segments and the

corresponding landmark point of the basis view. Thus, we obtain a pair of distances (one for each end-point) for each of the basis views ***a', b'*** and ***a", b"***. Then, the basis profiles are built by taking equally spaced samples up to ***a', b'***, ***a" and b"*** along the bisectors and perpendiculars. Each of the two possible search directions (that will be limited by ***a*** and ***b***) defines a *wing*, so that maybe one is larger than the other, but always on the same direction (the direction of the bisector/perpendicular). The profiles in the basis views (I' and I'') are pre-calculated before the main program loop, as they won't change throughout the iterations.

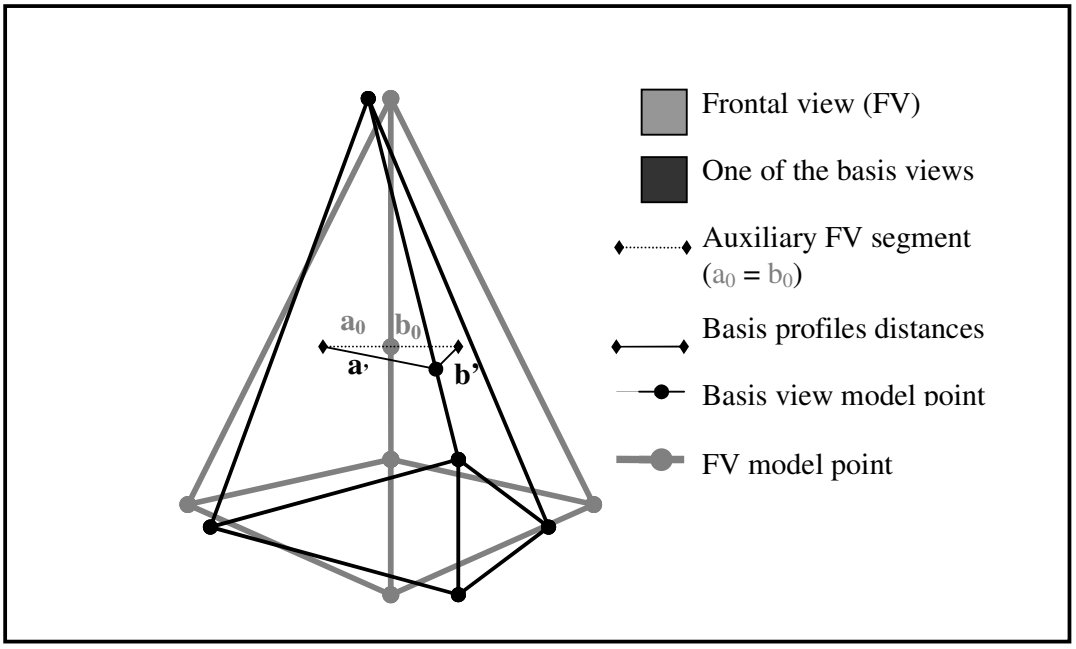


Figure 8-2. Calculating the size a profile of one of the basis views.

The joint profiles are calculated by interpolating the basis profiles intensities using the scheme proposed by Koufakis and Buxton [22]. This is done by weighting each of the values of the basis profiles as follows to produce I_i, the profile generated at the ith iteration,

$$I_i = w_i' I' + w_i'' I'', \tag{8-1}$$

where I' and I'' are the profiles in the basis views and w_i' and w_i'' are weights.

These weights are calculated from the distances of the current instance of the object to each of the basis views, as follows:

$$w_i^{'} = \frac{d''}{d'+d''} \text{ and } w_i^{''} = \frac{d'}{d'+d''}, \tag{8-2}$$

In turn, the distances d' and d'' are calculated, for each iteration, using the parameters of the stereo transformation (see section 2.1.2., equation 6), as follows [22]:

$$\begin{aligned} d' &= a_3^2 + a_4^2 + b_3^2 + b_4^2 \\ d'' &= a_1^2 + a_2^2 + b_1^2 + b_2^2 \end{aligned} \tag{8-3}$$

Once the joint profiles are built, we build the *image profiles* from the target image by taking equally spaced samples along the bisectors/perpendiculars, as observed above, up to certain distance $\boldsymbol{a_i}$ and $\mathbf{b_i}$ (for each of the profile wings). These distances will be calculated, again, by a weighted interpolation of the sizes of the basis profiles ***a', b'*** and ***a'', b''***, as follows:

$$\begin{aligned} a_i &= w_i^{'} a_i^{'} + w_i^{''} a_i^{''} \\ b_i &= w_i^{'} b_i^{'} + w_i^{''} b_i^{''} \end{aligned}, \text{ where } i \text{ is the iteration number.} \tag{8-4}$$

Finally, for every point, a search is performed along the search direction by matching the corresponding model profile to the image profile, and taking the best match, i.e. the one with the least squared difference, summed over the profile. The search is limited to three times the size of the profile segment ($3 \times a_i$ and $3 \times b_i$). The starting position of the image profile that gave the best match is the resulting point of the search. If there are several search directions for one point, we can adopt one of the following alternatives:

a. To take the point closest to the starting point.
b. To take the point found with best fit.
c. To take a weighted average of all the points found, the weights being determined by the corresponding error terms of the fit.

8.2.2 Second Approach

It can be seen that the above is fairly complicated. In addition, this first approach is incorrect because it does not take the pose into account correctly. The correct way of finding the basis profiles is by calculating first the necessary mappings from the frontal view to each of the basis views [22,17].

To do this, the image is manually divided into triangles ,as in [22] (see figure 8-2), and for each triangle a local linear mapping function is calculated, which maps every point of the triangle in the frontal view to a point in the corresponding triangle in each of the basis views. The corner points of the triangle in frontal view $(x_1, y_1), (x_2, y_2), (x_3, y_3)$, and the corner points of the triangle in one of the basis views $(x_1', y_1'), (x_2', y_2'), (x_3', y_3')$ and $(x_1'', y_1''), (x_2'', y_2''), (x_3'', y_3'')$, are corresponding 3D points, this is why we are entitled to calculate these local linear mapping for each of the basis views as follows:

$$\begin{pmatrix} x_1' & x_2' & x_3' \\ y_1' & y_2' & y_3' \end{pmatrix} = \begin{pmatrix} a' & b' & c' \\ d' & e' & f' \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} \tag{8-5}$$

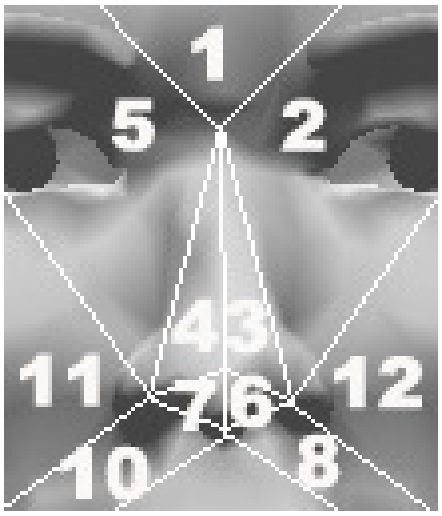


Figure 8-3. Triangulation and the order of the triangles.

Once the mapping for every triangle is known, given any point in the image, we can compute where it maps to in the basis views by: first calculating to which triangle it belongs, and secondly by using the corresponding transformation. We will use this for the end-points of the profile segments of the frontal view, thus, obtaining the end points of the profile segments of the basis views. Note that the profile direction is not necessarily along the bisectors/perpendiculars (as in the first approach) and that furthermore, each profile wing can be in different directions, and in principle, even in different triangles.

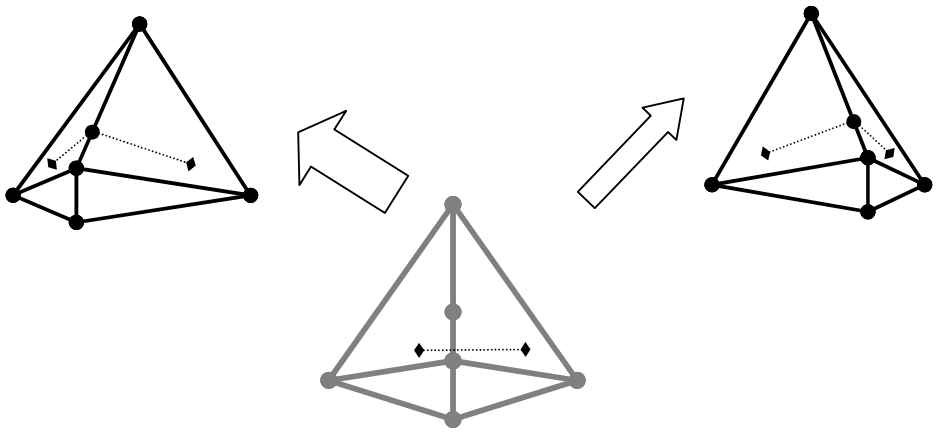


Figure 8-4. Calculation of the direction and size of the basis profile.

The end points of the target image profile are calculated using the current pose of the object (CATT parameters) and the position of the corresponding end points of both basis profiles. Each wing end point can, of course, be in different directions.

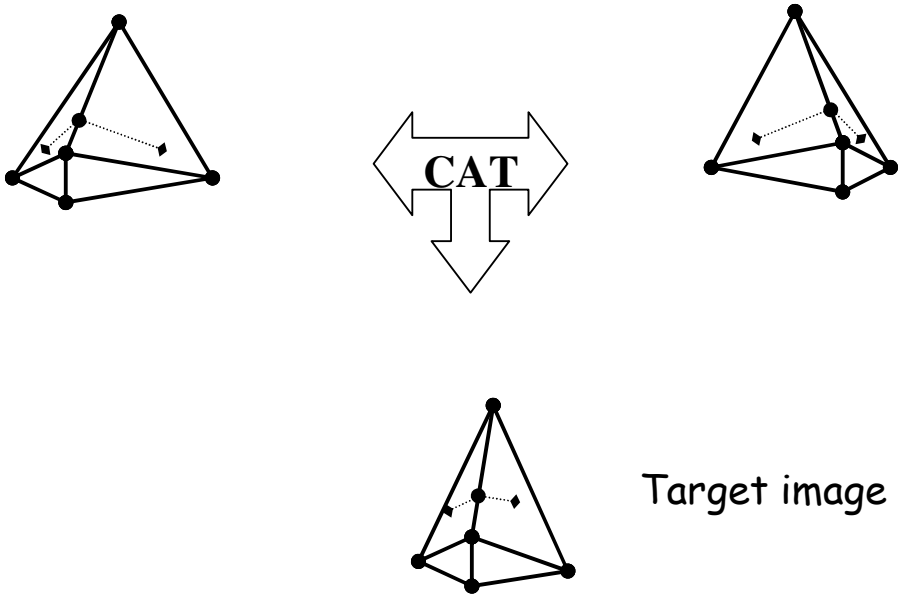


Figure 8-5. Calculation of the direction and size of the target image profile.

The model profile is obtained by weighting the intensity levels of both basis profiles, as we did in the first approach. Once this is done, the search along the profile direction can be made, comparing the model profile with the image profile at every point and, using any of the heuristics seen for the first approach, obtaining the best match.

9 Implementation Details for the grey-level approach

9.1 Data Flow

The following data flow diagram illustrates how the data is processed by the system in terms of inputs and outputs. Only the second approach is considered in the following data flow. Some details such as centring and scaling are obviated.

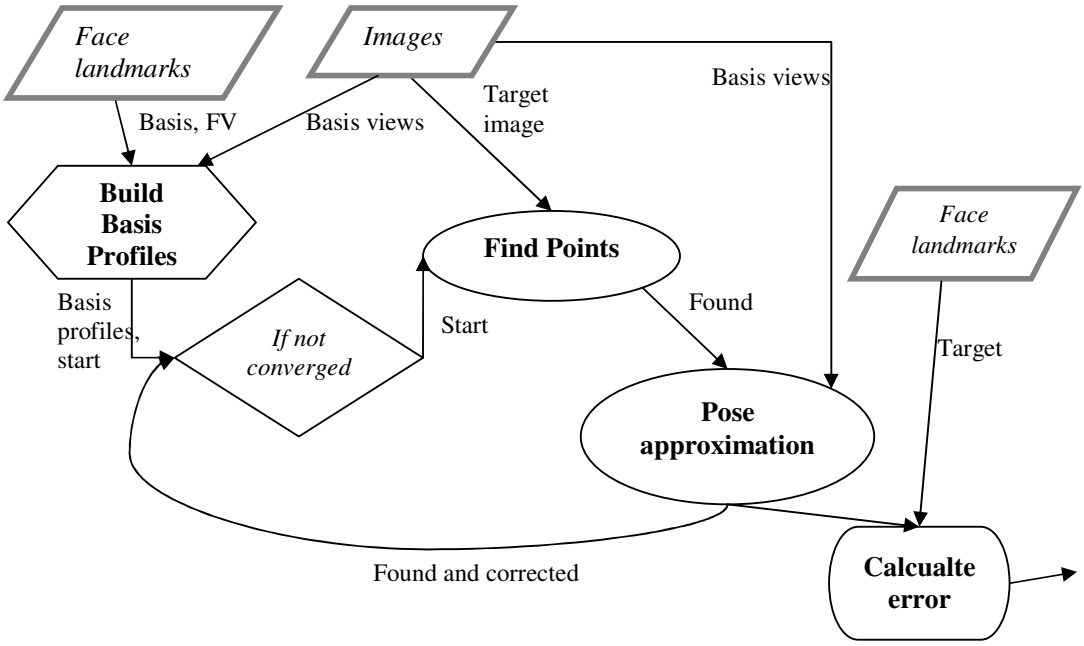


Figure 9-1. Data flow diagram for the system using grey-level images.

The processes ***Build Basis Profiles*** and ***Find Points*** are decomposed into subordinate data flow diagrams as follows:

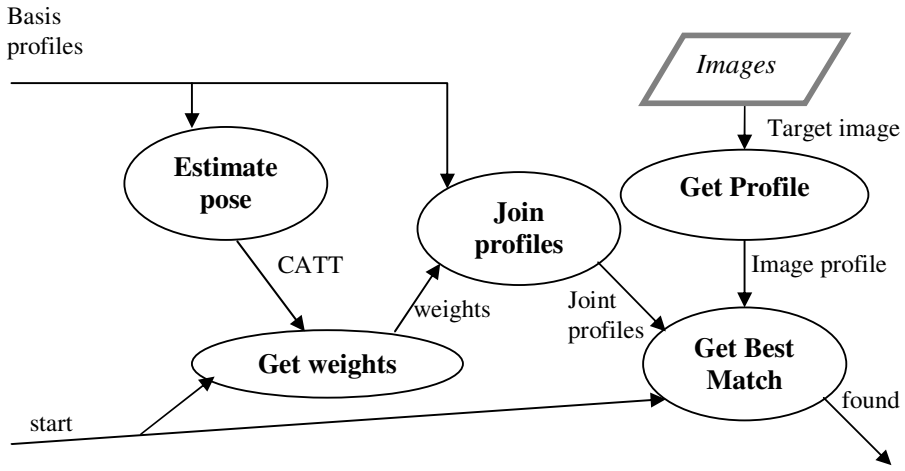


Figure 9-2. Data flow diagram for the process Find Points.

Error!

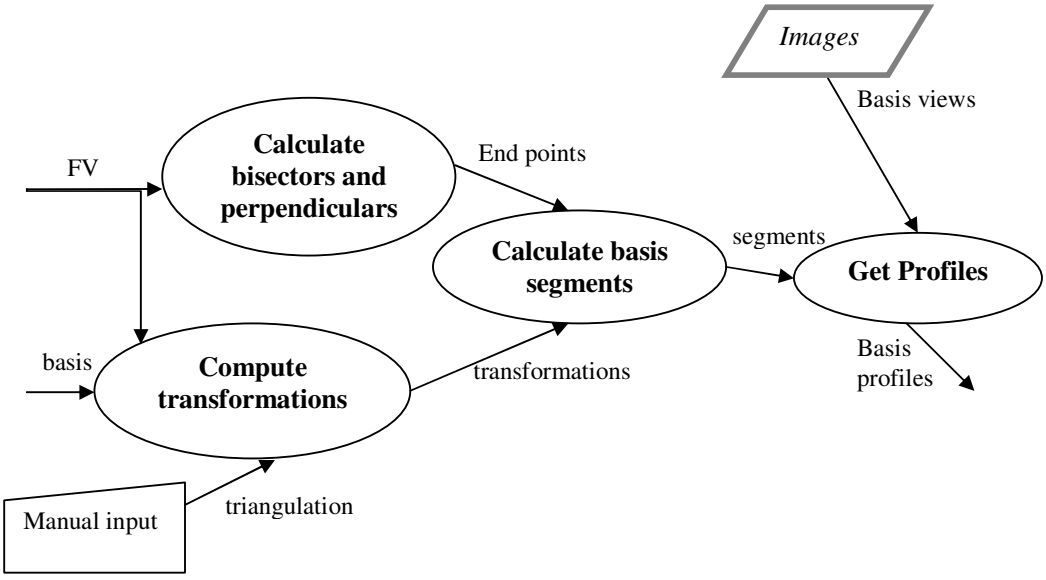


Figure 9-3. Data flow diagram for the process Build Basis Profiles.

9.2 Functions hierarchy

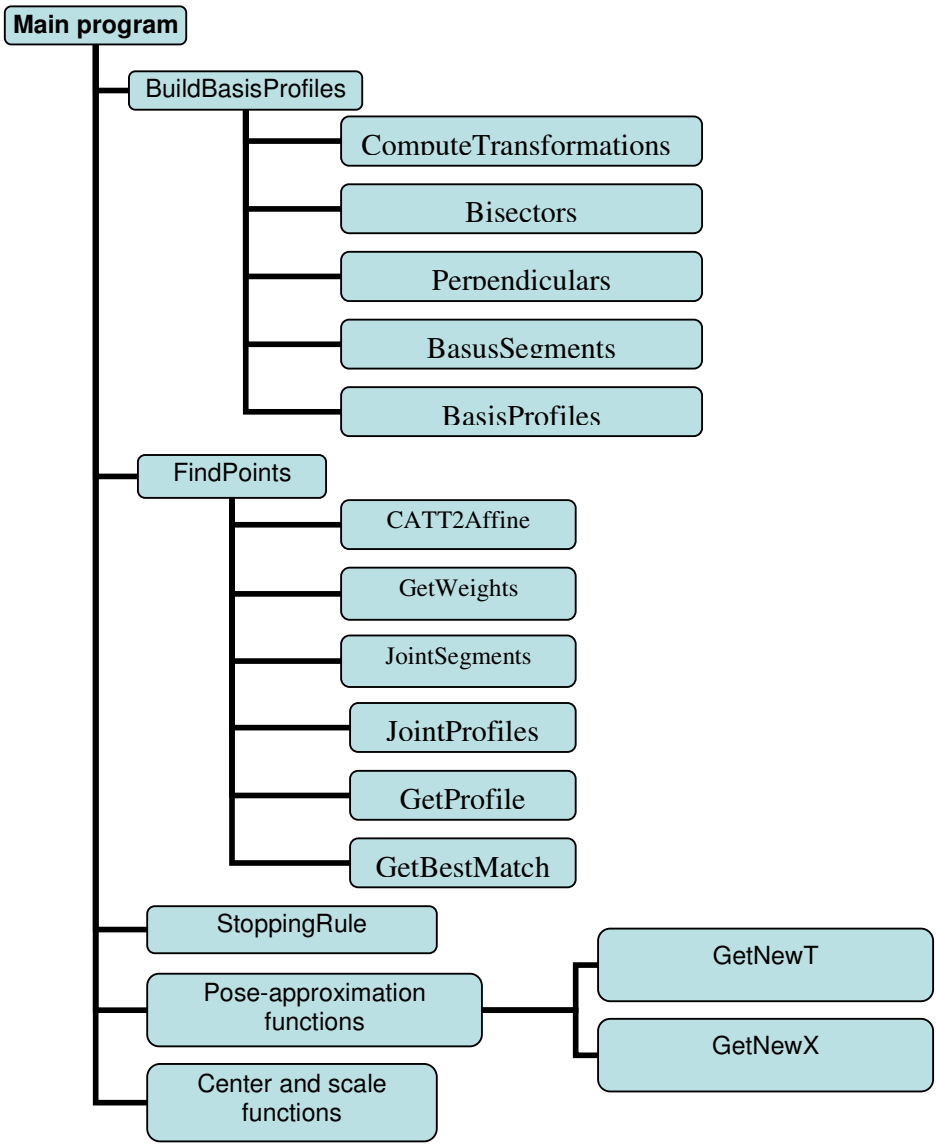


Figure 9-4. Functions hierarchy for the grey-level approach.

10 Testing for the grey-level approach

In this chapter we briefly outline the testing of the grey-level search procedures described earlier

10.1 Functionality tests

After the implementation of each system's functionality some tests were carried out to test their validity using some simple inputs for which we know the answer. These were some of the tests carried out:

- Weight calculation. Given the parameters of the transformation representing the current pose, the weights that indicate how much each basis view is "mixed-in" the transformation are returned as output. This was tested by giving as input the parameters representing the basis view's pose, and checked that the output was 1 and 0. Also, parameters corresponding to the frontal view pose were given as input and checked that the weights were close to 0.5.
- To test the piecewise mapping (used only in the second approach), the frontal view was given as input as the starting point and as the basis views, i.e. all three were the same set of points. The result expected is therefore null mappings for all the triangles according to a matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$. The mappings obtained were acceptably close to this ideal result.
- The previous test was extended to verify the directions of the profiles, as they should lie on the frontal view's bisectors/perpendiculars.
- As mentioned in section 8.2.1., there are two ways of building the joint profile:
    a. computing them after full reconstruction of the joint image
    b. combining the basis profiles by weighting the intensities

  A further test was performed, calculating the RMS error between these two ways of calculating the joint profile. The result is shown in appendix A.

10.2 First approach

The plan was to test the system in its integrity with all of the training images (seen in figure 8-1, section 8.1), but unfortunately the system failed with the very first test image: the frontal view.

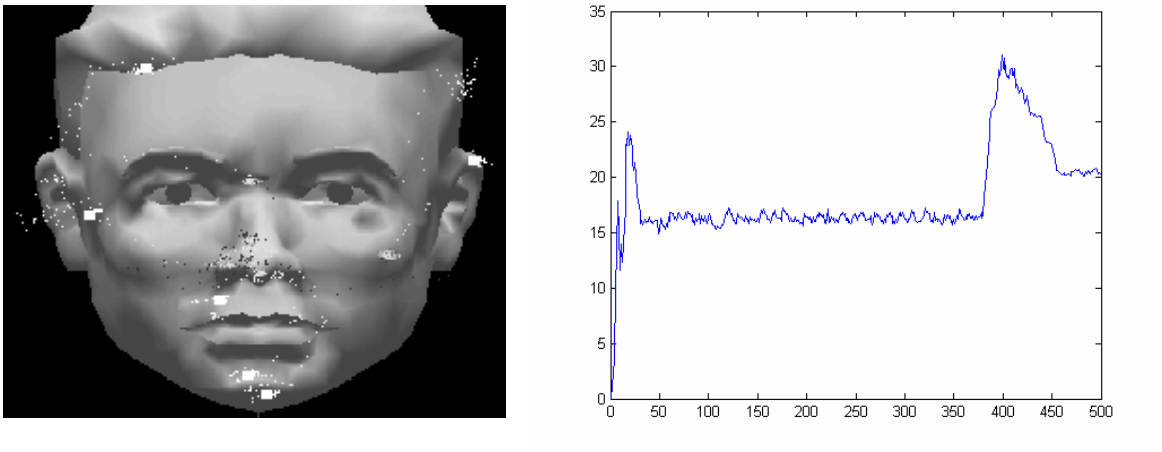


Figure 10-1. Result for the first approach using the frontal view as target image.

These results are pretty unfortunate. The search seems to go astray as far as the head limits. Furthermore, the search looks very unstable, reaching the maximum number of iterations allowed (500) without converging.

The testing was not taken any further, because the method was found to be incorrect, as described in section 8.2.2, and had to be completely redesigned.

10.3 Second approach

Again, we started by testing the newly designed system with the FV, obtaining good results, then we tried with the next immediate image (rotated 5 degrees):

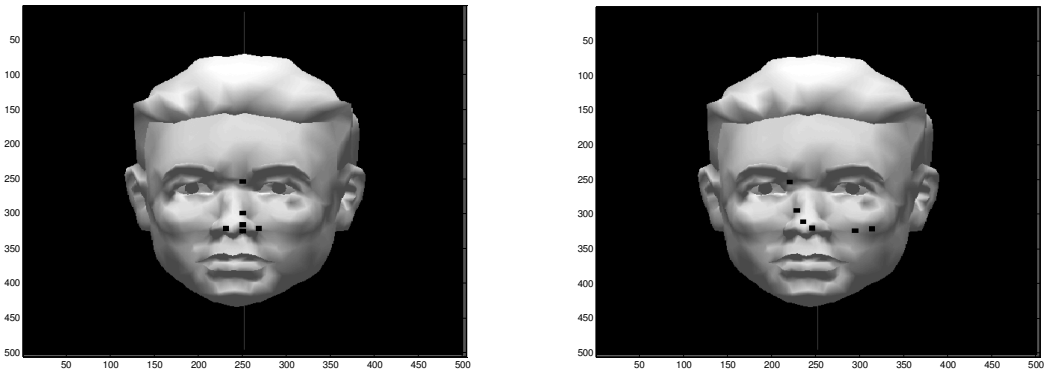


Figure 10-2. Starting search position (left) and points found after convergence (right).

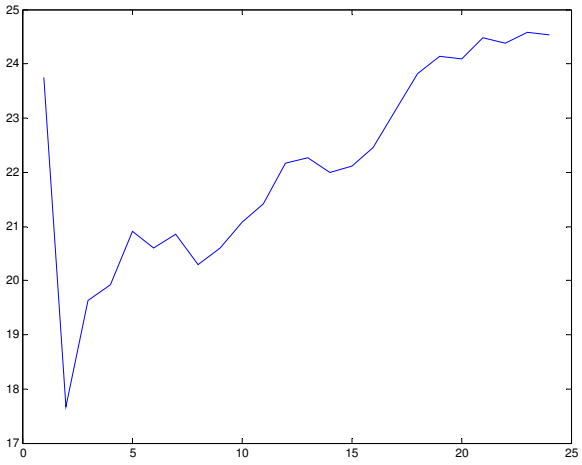


Figure 10-3. Error plot for the second approach using the test image rotated 5 degrees.

Again, the results are unfortunate, although not as bad as before. The points found throughout the search move further and further away from the solution. Similar or worse results were obtained using other target images (see appendix A).

The explanation to this outcome may be the way the triangulation is done. When dividing the image into triangles, the corners of the whole image is taken as control points of the most exterior triangles, so that these exterior triangles are very elongated, possibly causing some of the profiles lying on the outer triangles to be accordingly and erroneously elongated. This could be solved by taking other exterior control points closer to the area of interest, i.e. the nose. To test if the problem is due to this, the

whole joint image was built; the results showed that the nose area is correctly warped, so that this is not the cause of the problem.

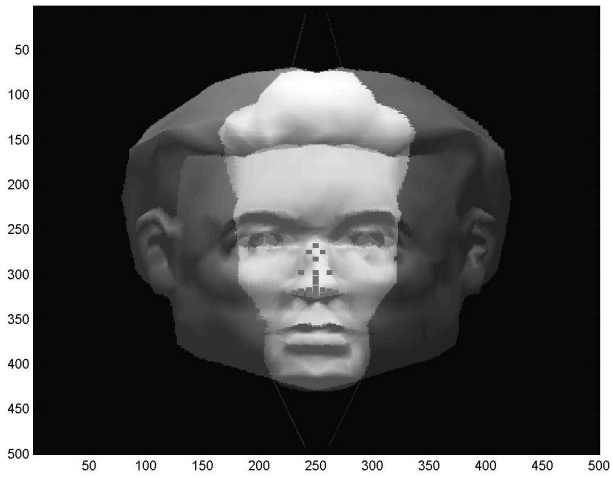


Figure 10-4. Joint image.

Among other alternatives, the search was tried without the intermediate subsidiary points, giving surprisingly much better results. These will be discussed in the next chapter.

11 Results for the grey-level approach

The results obtained when modifying the second approach by removing the subsidiary points are encouraging, but not completely satisfactory. The system performed very well when the target was not very far away from the frontal view. For example, when presented with an image taken from a viewpoint rotated 5 degrees with respect to the frontal view, this is the result:

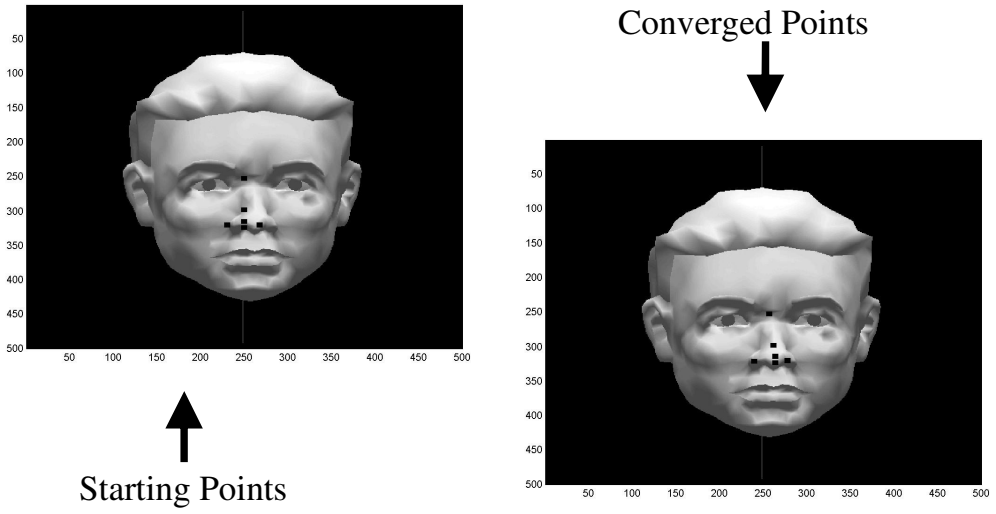


Figure 11-1. Result for the modified second approach when target is rotated 5 degrees.

Then, when presented with other test images with a larger viewpoint rotation, the method breaks down and does not converge. For example, when the system is presented with the next test image, which is rotated 10 degrees from the frontal view, the result is:

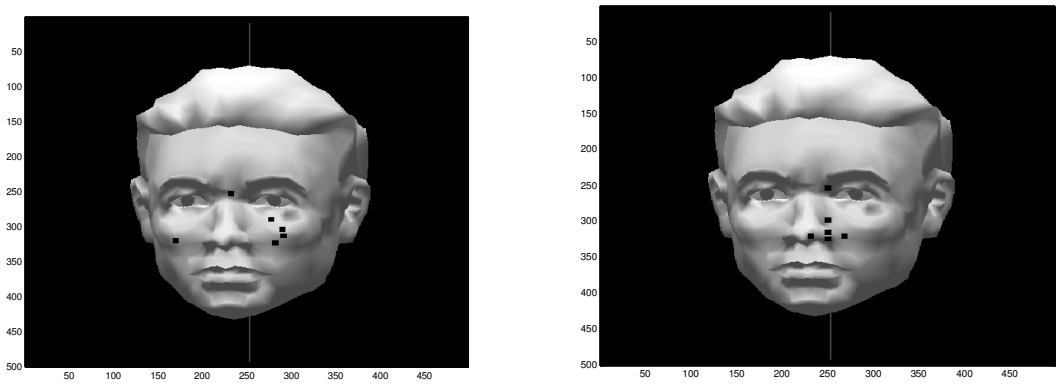


Figure 11-2. Starting search position (left) and points found after convergence (right).

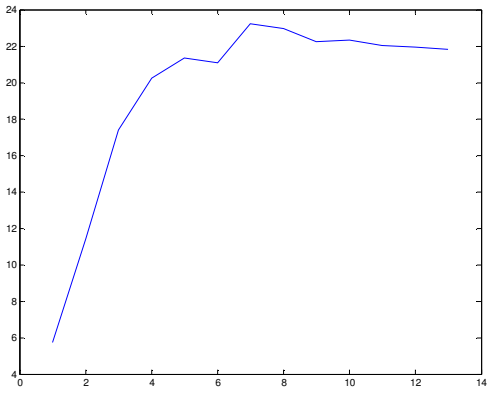


Figure 11-3. Error plot for the second approach without working with subsidiary points, using the test image rotated 10 degrees.

This disappointing result could be because the system does not deal with translations well or because it does not cope with large pose changes. To determine the cause, the starting points (taken from the frontal view) were manually taken closer to the nose in the target image. Obtaining, for example, for the most rotated test images (which is also one of the basis view):

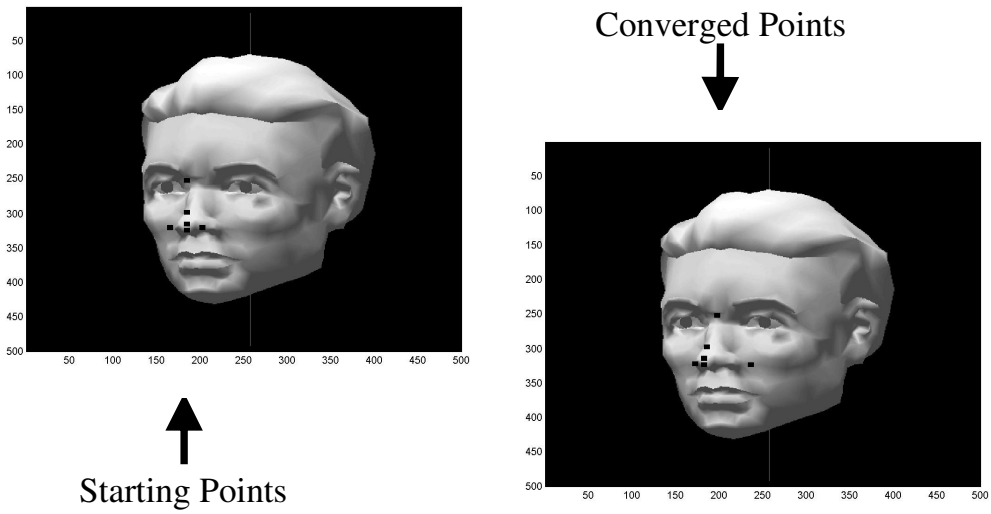


Figure 11-4. Result for the modified second approach using the one of the basis views as target image.

Although, in this case, one of the points is found quite far away from where it should be, the results were altogether good, which means that the method copes with pose changes, but not very well with translations.

The reason why working without subsidiary points gives better results, while in the line-drawing approach were so useful is not clear. One possible explanation for this is that the generation of the subsidiary points is not theoretically correct because a subsidiary point generated in three different poses (the frontal view and the two basis views), does not correspond correctly across the three views.

12 Evaluation, Conclusions and Further Work

The work here presented, successfully implements an iterative method that effectively searches and locates the landmark points of an object in novel images given only two basis views. This is done by modelling the grey-levels by means of one-dimensional profiles. Results are presented that shows that the method is successful both with line-drawing and, to some extent, with grey-level images. Constraining the search only with the CATT, the pose is also determined during the search.

The performance when using grey-levels could be greatly improved by coupling a feature tracker to the system. This way, the starting point of the search could be made to be near the target image, solving the main problem our method has: coping with position offsets.

The system can be improved also by changing some design details. For example, when building the grey-level profiles, we interrogate the image to get the intensity at some point I(x, y). The coordinates of this point can have floating point values. In our system, we just rounded the values to the nearest integer to get the pixel intensity. A bilinear interpolation between the four nearest neighbouring pixels could be performed to get more accurate information.

It is possible to test the system further, at least in the grey-level version, and try and tune some parameters that had been, otherwise, kept fix throughout the project. For example, the profiles have been always ten pixels long. A set of test could be done to see if some other size is more proper. Also, we would have liked to see how the system deals with real images instead of synthetic ones.

These kind of straightforward improvements have not been made in this project due to the short time frame.

The project allows for several major extensions:

- Appearance could be incorporated by building statistical grey-level models instead of just simple profile correlation. In this same line of development, the system could take into account possible lightning changes in the novel images and try to integrate intensity variation into the model by, maybe, normalising the intensity.
- Colour images provide three times more information than grey-levels. Thus, the colour can be also integrated into the model.
- One of the most interesting extensions is to build an ASM of other face features, such as the mouth, once the pose of the face is known. So that the mouth shape variations can be modelled independently from the face pose. The information provided by the ASM, could be further analysed to build a robust speech recognition system, or to animate faces of avatars.
- The system could be carefully re-implemented so as to use it in a real time application.

As it turns out, the work carried out on pose estimation and feature recognition has proved to be quite successful from a learning and also practical point of view. Personally, I am very satisfied with the progress and outcome of the project. We developed and evaluated two different versions of the system: for line-drawing and grey-level images, meeting all the goals we set ourselves at the beginning of the project.

Finally, I hope someone will benefit from the results of our research, in which case, the author would be very happy to hear about it.

BIBLIOGRAPHY

[1] A. Blake and A. Yuille, editors. *Active Vision*, MIT-Press, pages 59-68, 1992.

[2] L. Bretzner and T. Lindeberg. Use your hands as a 3-d mouse, or, relative orientation from extended sequences of sparse point and line correspondences using the affine trifocal tensor. In Proc. 5th ECCV, LNCS, volume 1406, pages 141 – 157. Springer Verlag, Berlin, June 1998.

[3] T. F. Cootes, G. J. Edwards, and C. J. Taylor. *Active appearance models. In Proc. European Conference on Computer Vision*, pages 484-498, June 1998.

[4] T. F. Cootes, G.J. Page, C.B. Jackson, and C.J.Taylor. Statistical Grey-Level Models for Object Location and Identification. Department of Medical Biophysics, University of Manchester. June 1998.

[5] T. F. Cootes and C.J.Taylor. Active Shape Model Search using Local Grey-Level Models : A quantitative Evaluation. Department of Medical Biophysics, University of Manchester. 1993.

[6] T. F. Cootes and C.J.Taylor. Active Shape Models – 'Smart Snakes'. Department of Medical Biophysics, University of Manchester. 1992.

[7] T. F. Cootes and C. J. Taylor. Statistical models of appearance for computer vision. http://www.wiau.man.ac.uk/~bim/Models/app_model.ps.gz, October 2001.

[8] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. A trainable method of parametric shape description. In Image and Vision Computing, volume 10, pages 289 – 294, 1992.

[9] T. F. Cootes, G. V. Wheeler, K. N. Walker, and C. J. Taylor. Coupled-view active appearance models. In Proc. BMVC, volume 1, pages 52 – 61, 2000.

[10] VIDEO REALISTIC TALKING HEADS USING HIERARCHICAL NON-LINEAR SPEECH-APPEARANCE MODELS *Darren Cosker, David Marshall, Paul Rosin and Yulia Hick.s* Cardiff University. *Proceedings of Mirage 2003, INRIA Rocquencourt, France, March, 10-11 2003*

[11] M. B. Dias and B. F. Buxton. Integrated shape and pose modelling. In Proc. BMVC, volume 2, pages 827 – 836, September 2002.

[12] M. B. Dias and B. F. Buxton. Enforcing a shape correspondence between two views of a 3d non-rigid object. In to appear in the Proc. 8th Iberoamerican Congress on Pattern Recognition (CIARP2003), Havana, Cuba, November 2003.

[13] M. B. Dias and B. F. Buxton. Separating shape and pose variations. Technical Report RN/03/01, University College London, Department of Computer Science, January 2003.

[14] B. F. Buxton and M. B. Dias,View Invariant, Image-Based Linear Flexible Shape Modelling, University College London (UCL), Department of Computer Science, Technical Report RN/03/11, August 2003.

[15] B. F. Buxton and M. B. Dias, The Principles of View Invariant, Image-Based Linear Flexible Shape Modelling, invited paper, submitted to the 5th International Conference on Advances in Pattern Recognition ICAPR 2003, December 10 - 13, 2003.

[16] G. H. Golub and C. F. Van Loan, Matrix Computations, Third Edition, John Hopkins University Press, ISBN 0-8018-5414-8, 1996.

[17] A. Goshtasby. Piecewise Linear Mapping Functions for Image Registration. *Pattern Recognition,* 19(6):459-466,1986,

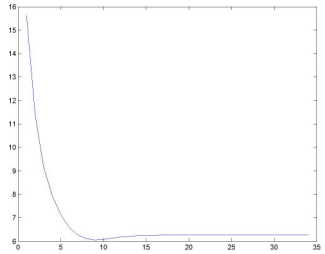
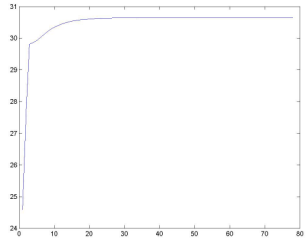
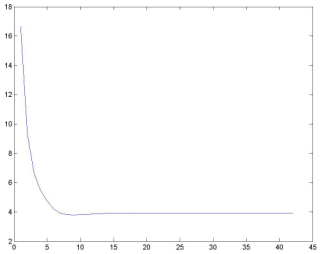
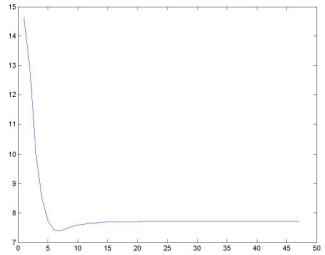
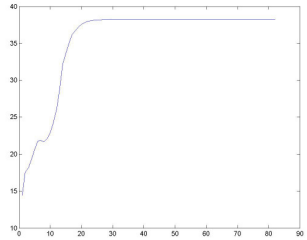
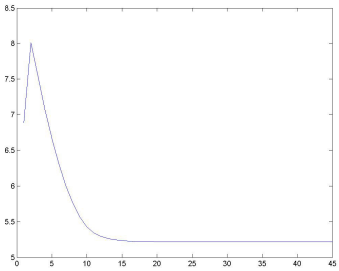
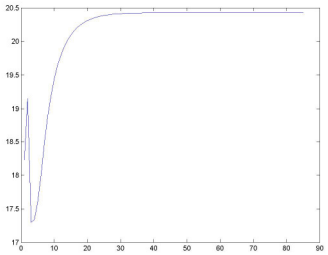
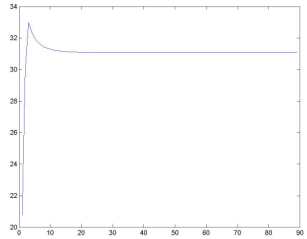
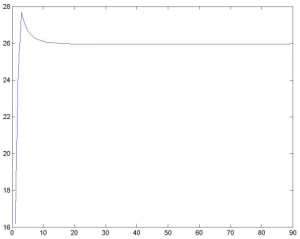
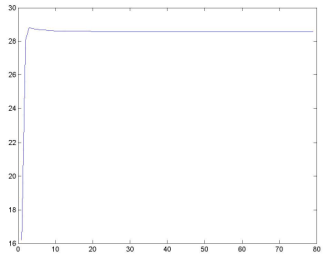
[18] M. E. Hansard and B. F. Buxton. Parametric view-synthesis. In Proc. 6th ECCV, volume 1, pages 191 – 202, 2000.
[19] R. Hartley and A. Zisserman. Multiple View Geometry In Computer Vision. Cambridge University Press, 2000.
[20] A. Hill, T.F. Cootes and C. J. Taylor. Active Shape Models and the Shape Approximation Problem. Department of Medical Biophysics, University of Manchester. 1995.
[21] D. M. Kennedy, B. F. Buxton, and J. H. Gibly. Application of the total least squares procedure to linear view interpolation. In Proc. BMVC, volume 1, pages 305 – 314, 1999.
[22] I. Koufakis and B. F. Buxton. Very low bit-rate face video compression using linear combination of 2d face views and principal component analysis. Image and Vision Computing, 17:1031 – 1051, 1998.
[23] S.Lucey. Audio-visual speech processing. PhD Thesis. Queensland University of Technology 2002
[24] J.Luettin, N.A.Thacker, S.W.Beet. Active Shape Models for Visual Speech Feature Extraction. University of Sheffield, UK. 1996.
[25] I. Matthews, T.Cootes, S.Cox, R.Harvey, and J.A.Bangham. Lipreading Using Shape, Shading and Scale. School of Information Systems, University of East Anglia, Norwich NR4 7TJ.
[26] A. Shashua. Trilinear tensor: The fundamental construct of multiple -view geometry and it's applications. In Proc. International Workshop on Algebraic Frames For The Perception-Action Cycle (AFPAC97), Kiel, Germany, September 1997.
[27] T. Thorhallsson and D. W. Murray. The tensors of three affine views. In Proc. International Conference on Computer Vision and Pattern Recognition, volume 1, pages 450 – 456, 1999.
[28] S. Ullman and R. Basri. Recognition by linear combinations of models. In IEEE Transactions on PAMI, volume 13, pages 992 – 1006, 1991.

APPENDIX A

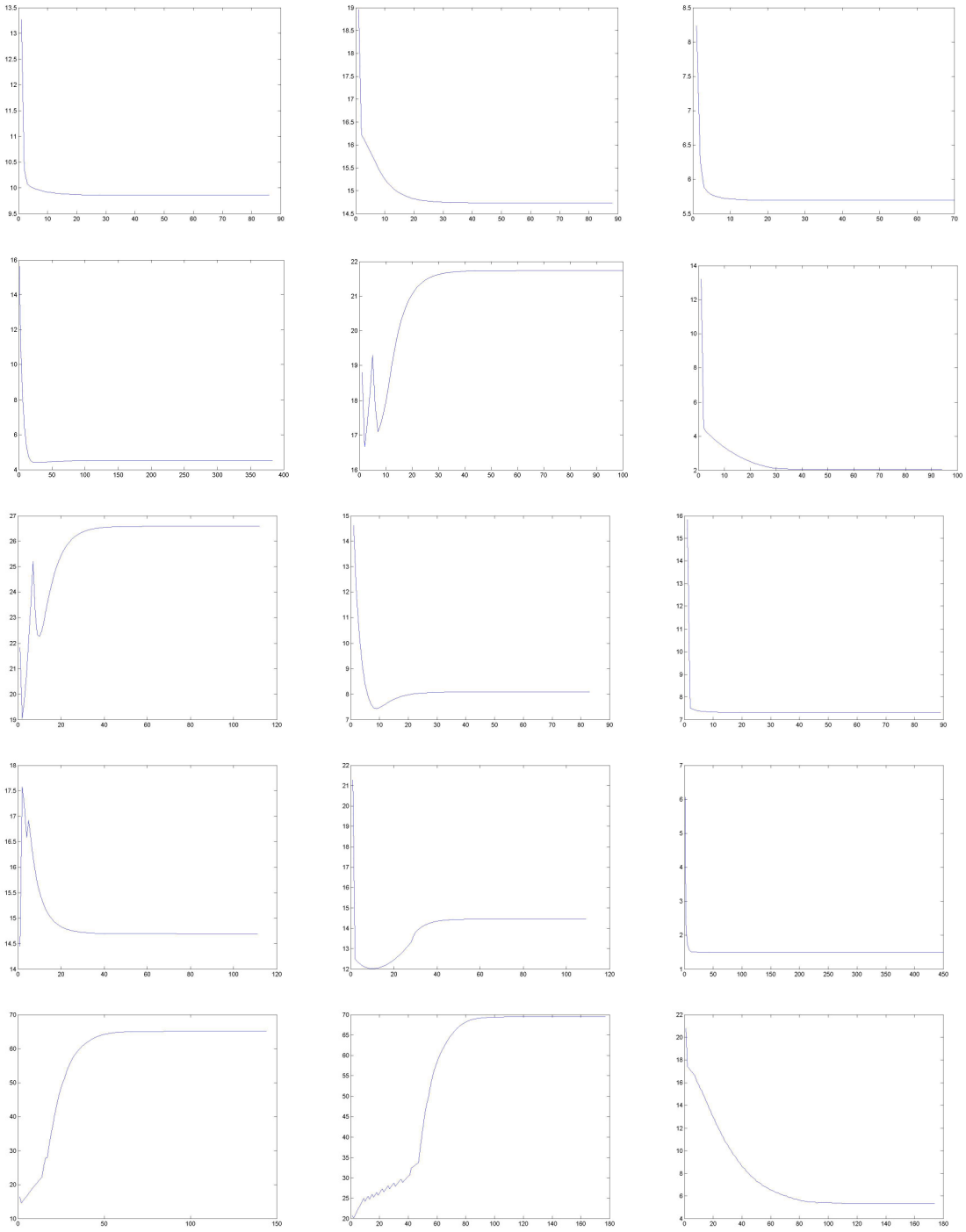
Testing for the geometrical approach

Using only geometrical information

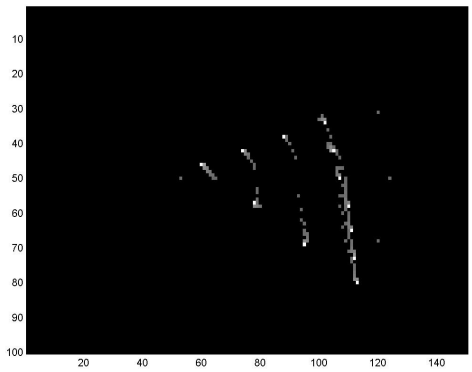
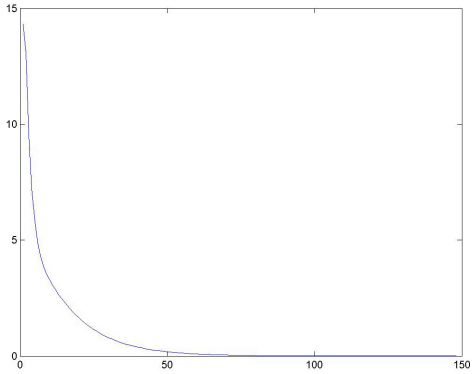
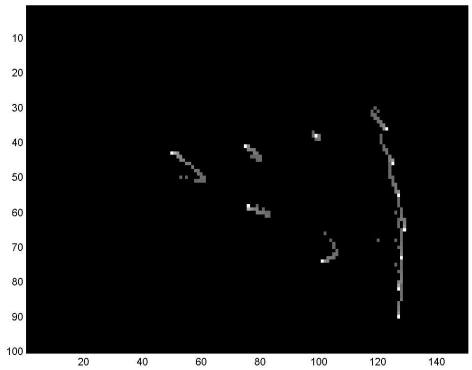
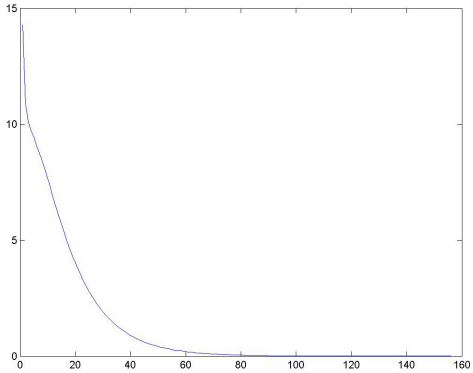
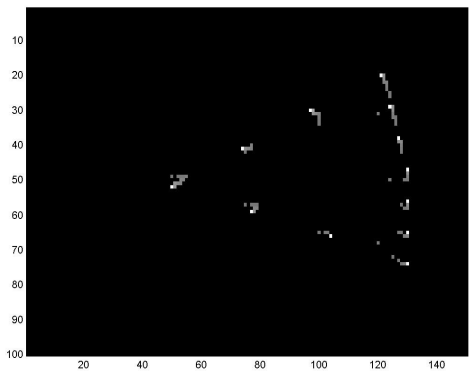
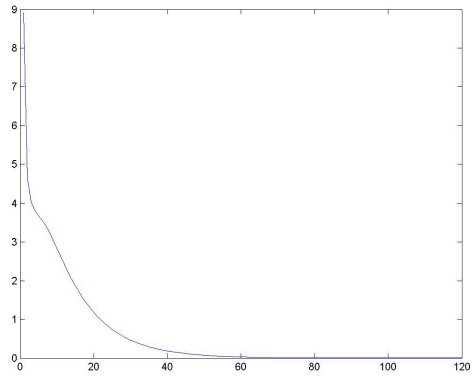
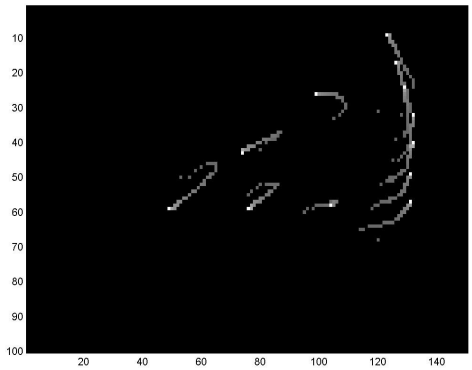
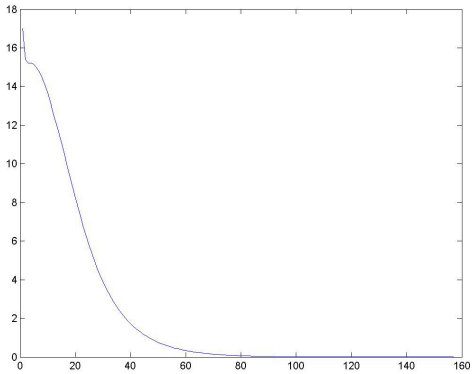
a. Using bisectors only

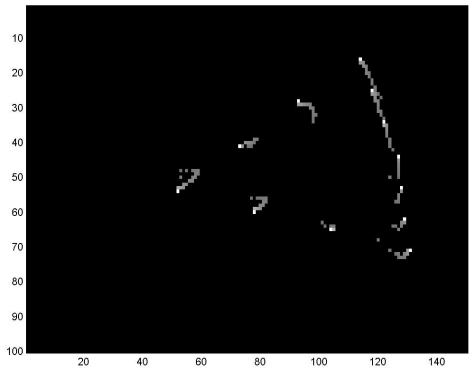











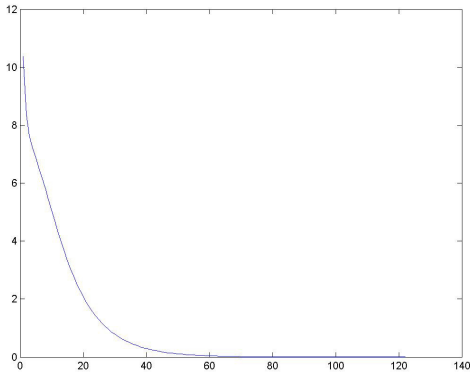
b. Using connections between adjacent corner points as an extra search direction

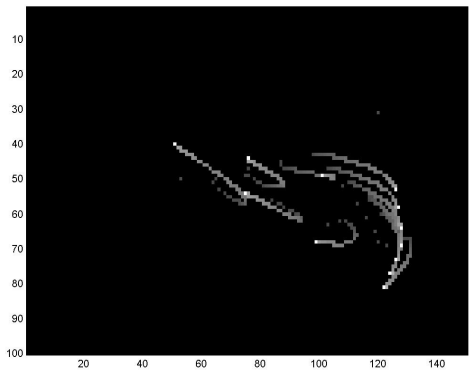


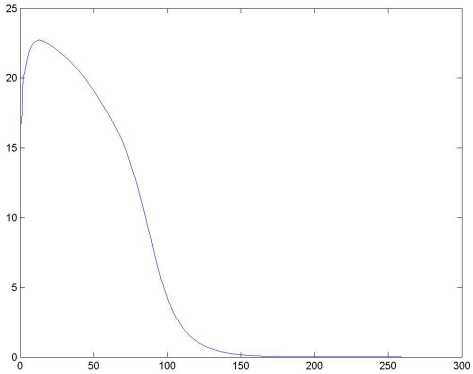
c. Including subsidiary landmark points

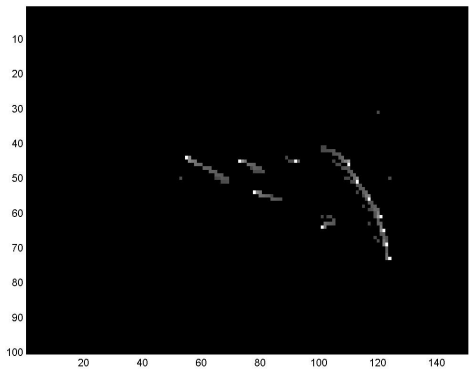









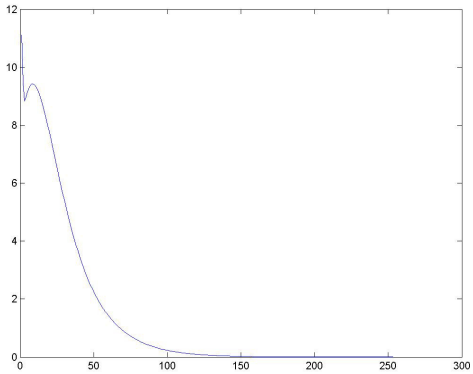

10
20
30
40
50
60
70
80
90
100
20
40
60
80
100
120
140

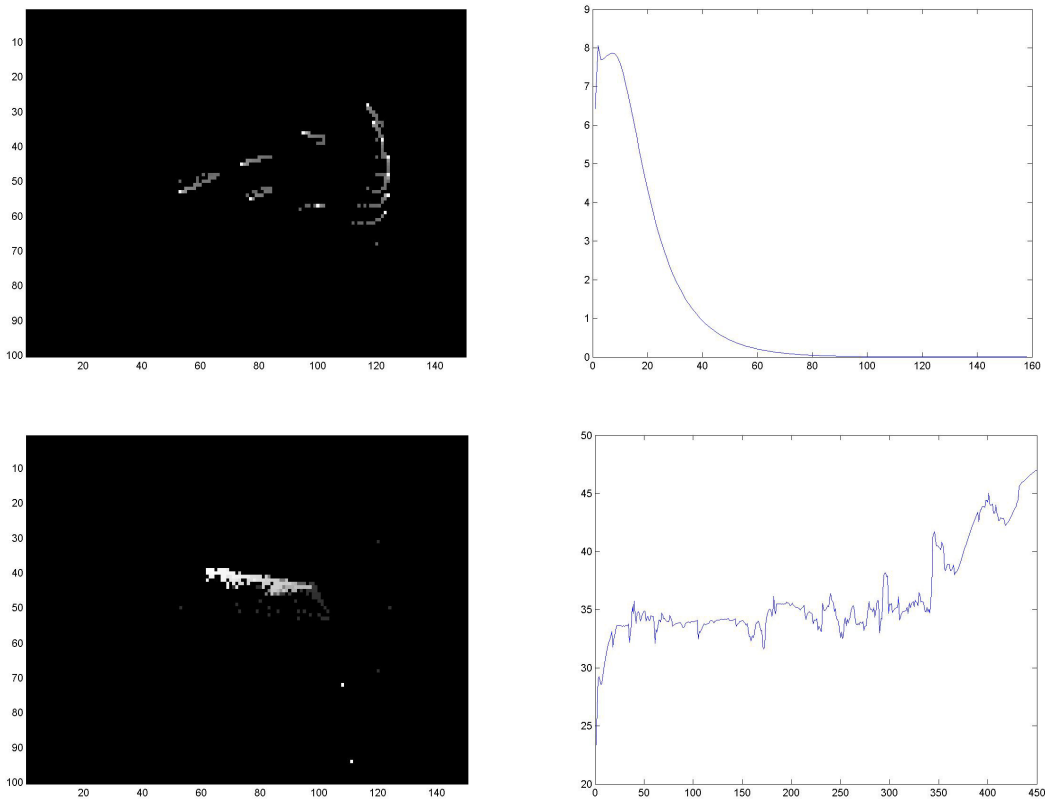

12
10
8
6
4
2
0
0
20
40
60
80
100
120
140


10
20
30
40
50
60
70
80
90
100
20
40
60
80
100
120
140


25
20
15
10
5
0
0
50
100
150
200
250
300


10
20
30
40
50
60
70
80
90
100
20
40
60
80
100
120
140


12
10
8
6
4
2
0
0
50
100
150
200
250
300

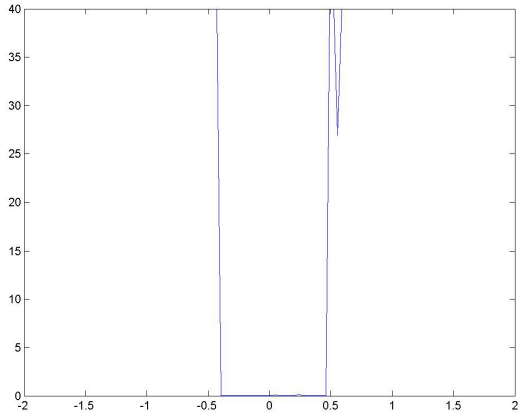


Minimal profile (line-drawings) under single-view transformations

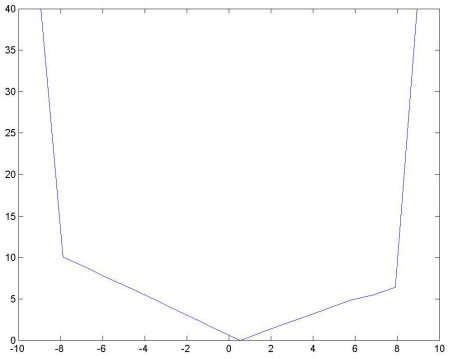
Without added noise

BASIS OF ATTRACTION – AFFINE TEST Tol 0.001 Max_Iter: 500 N=20
Range for translation [-10 10], for parameters [-3 3]

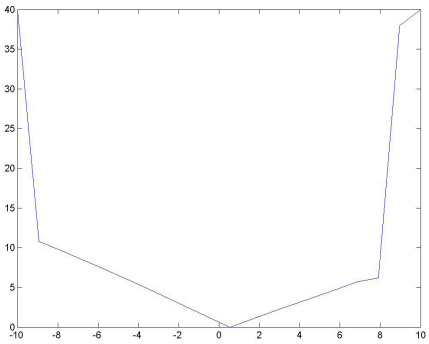
Rotation



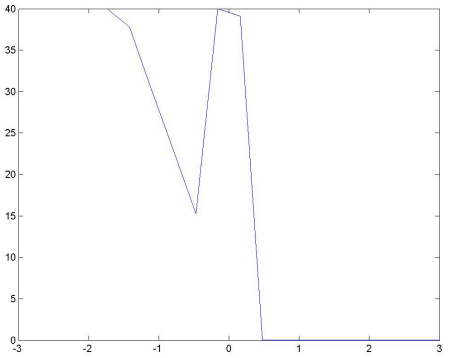
TRANSLATION: in x axis



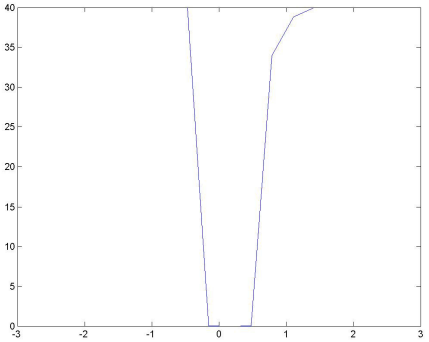
random direction



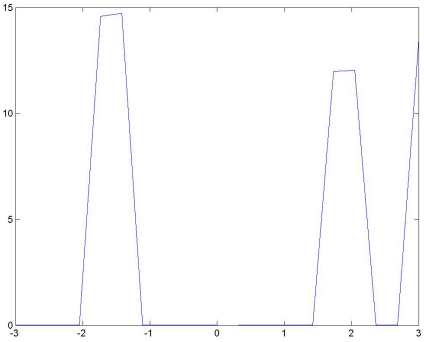
MODIFYING ‘a’



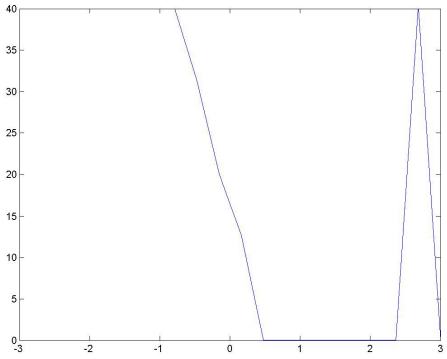
MODIFYING ‘b’



MODIFYING ‘c’

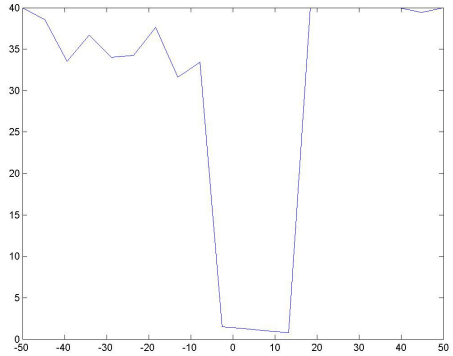


MODIFYING ‘d’

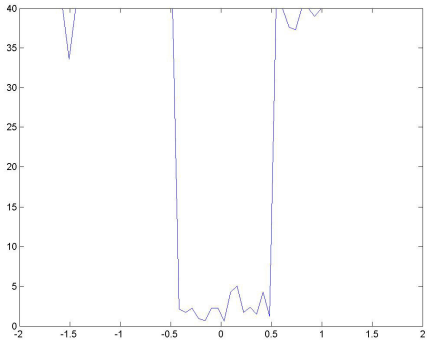


With added noise

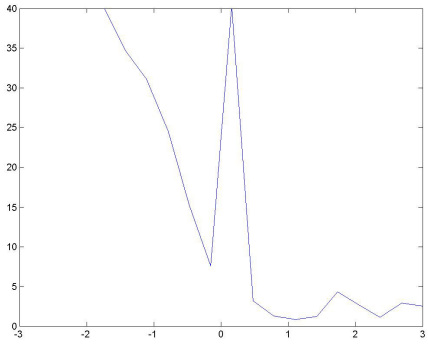
TRANSLATION rand



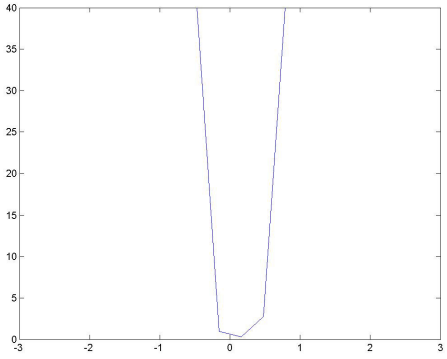
Rotation



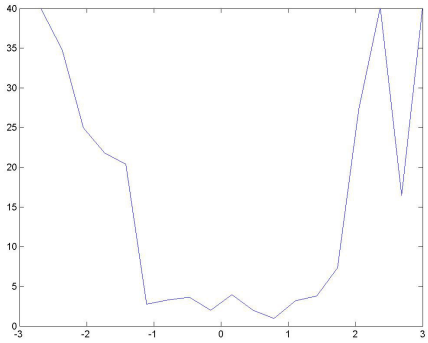
MODIFYING ‘a’



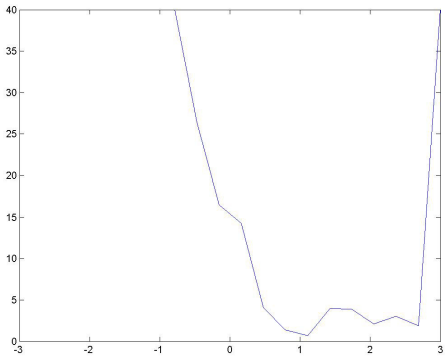
MODIFYING ‘b’:



MODIFYING ‘c’:



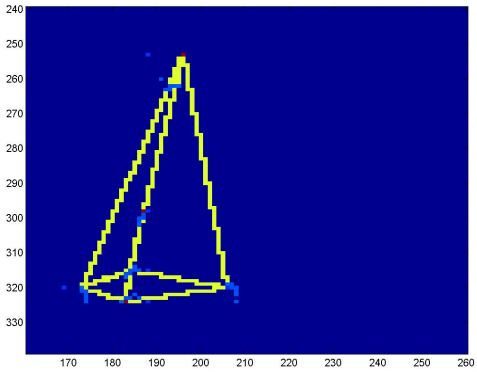
MODIFYING ‘d’

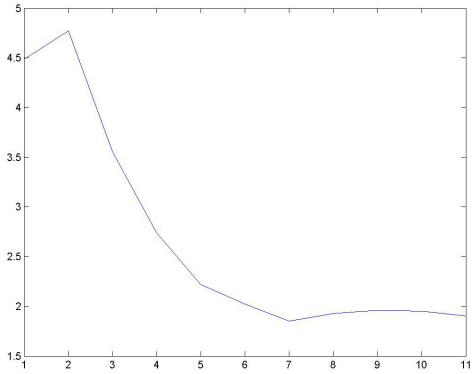


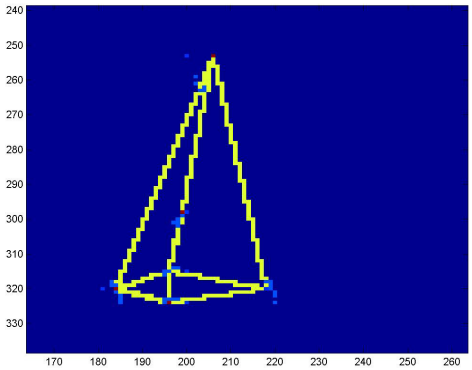
Minimal profile (line-drawings) under stereo transformations

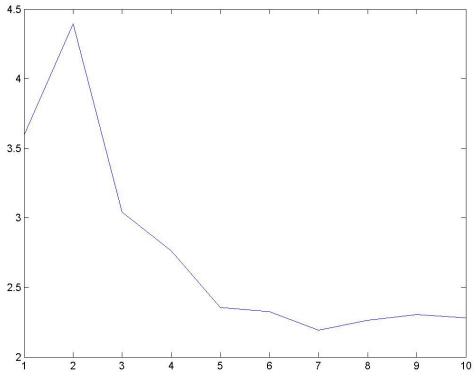
Affine approach

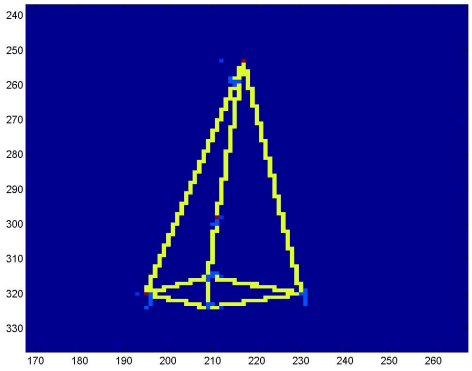
Known centre and scale of the target object.

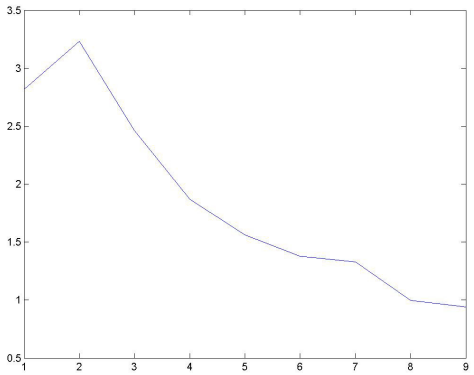


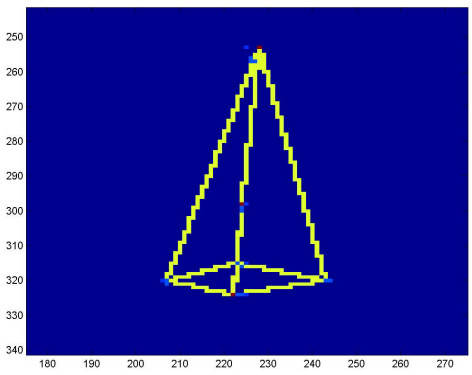


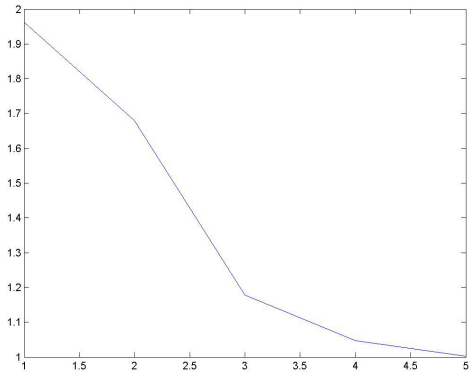


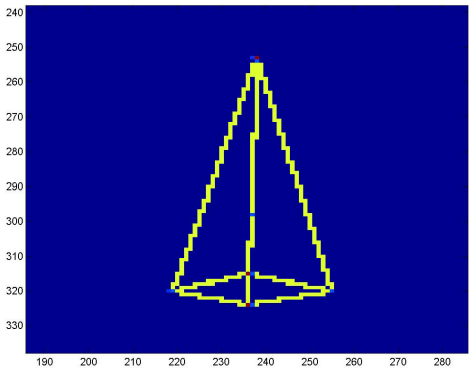


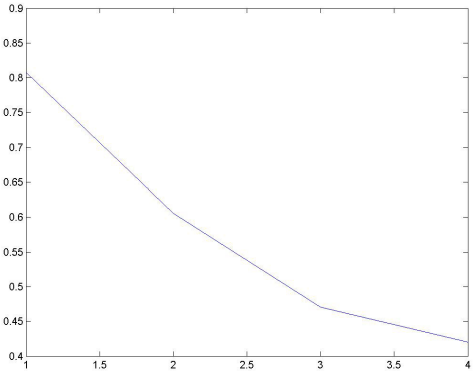




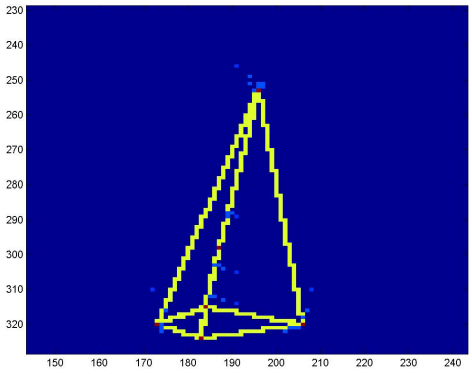


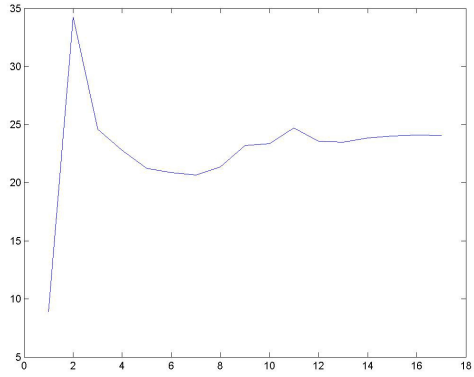


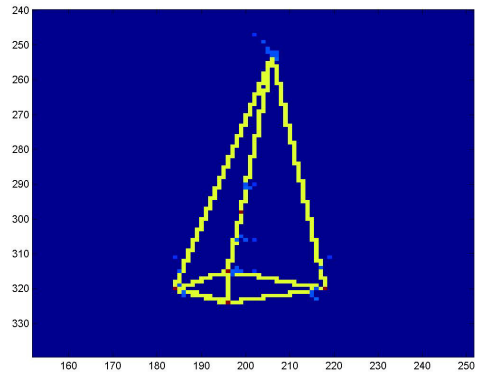


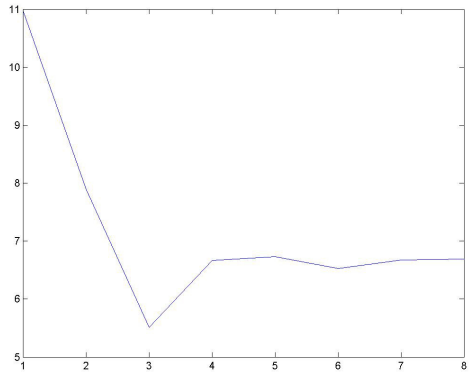


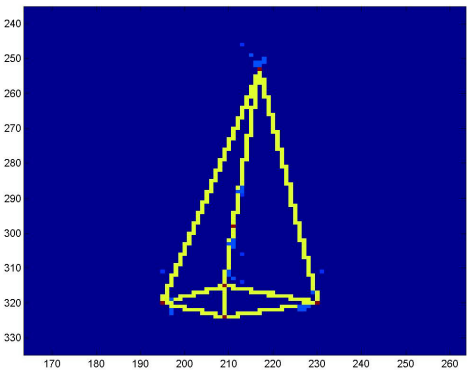
Using an estimate of the centre and scale of the target object

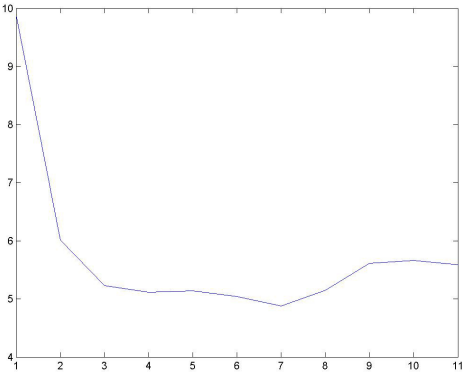


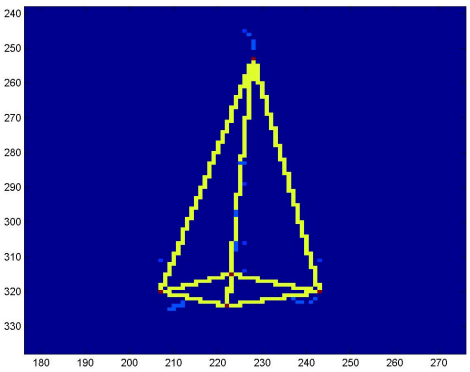


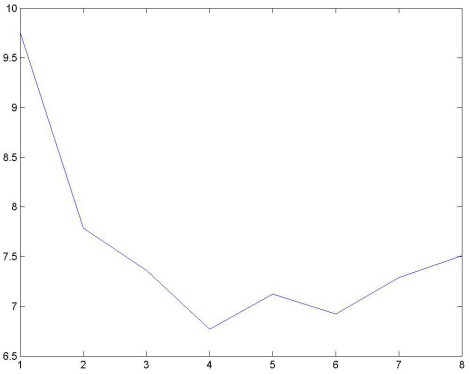

240
250
260
270
280
290
300
310
320
330
160 170 180 190 200 210 220 230 240 250

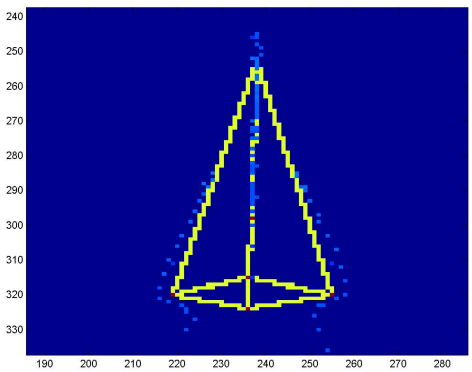

11
10
9
8
7
6
5
1 2 3 4 5 6 7 8

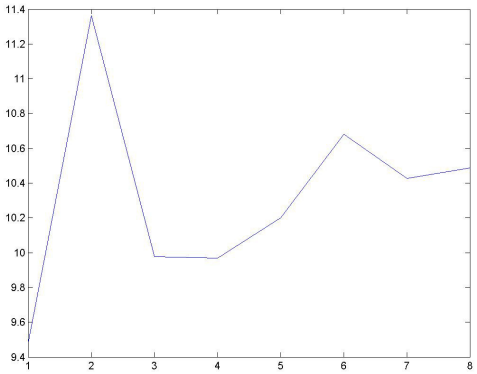

240
250
260
270
280
290
300
310
320
330
170 180 190 200 210 220 230 240 250 260

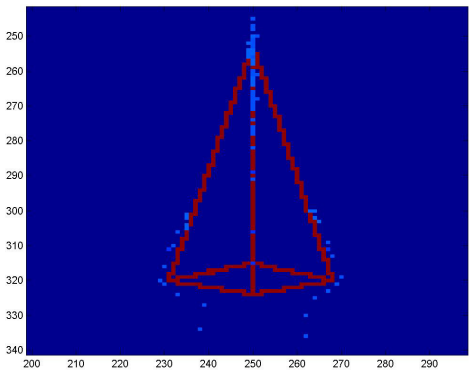

10
9
8
7
6
5
4
1 2 3 4 5 6 7 8 9 10 11

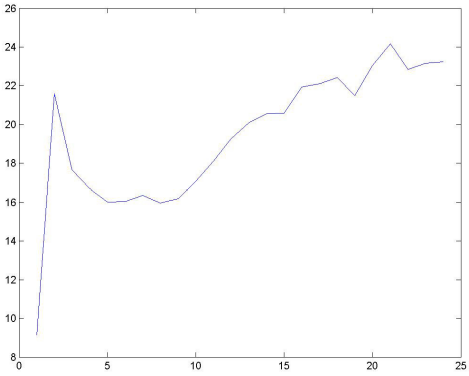

240
250
260
270
280
290
300
310
320
330
180 190 200 210 220 230 240 250 260 270


10
9.5
9
8.5
8
7.5
7
6.5
1 2 3 4 5 6 7 8



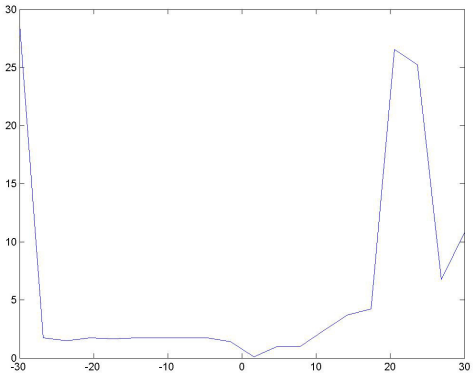




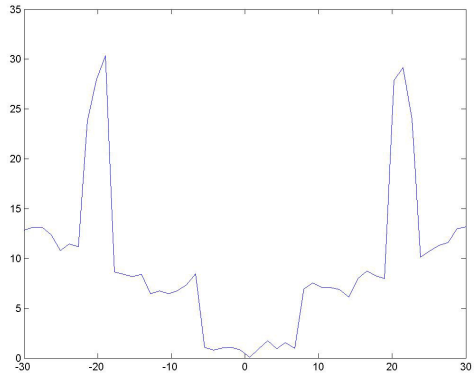


Basins of attraction

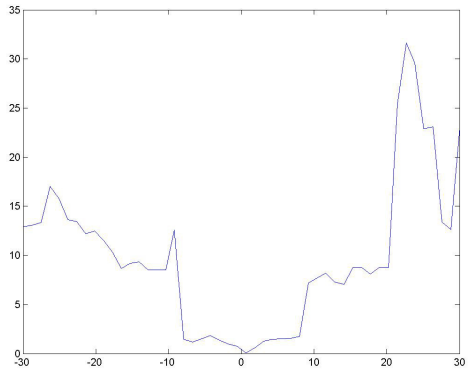
Trans3at56n 5n x



Trans3at56n 5n y

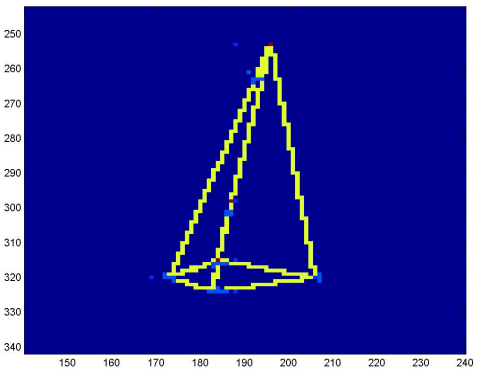


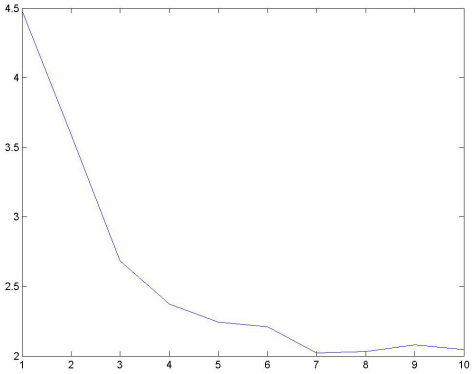
Trans3at56n 5n rand60 d5rect56n

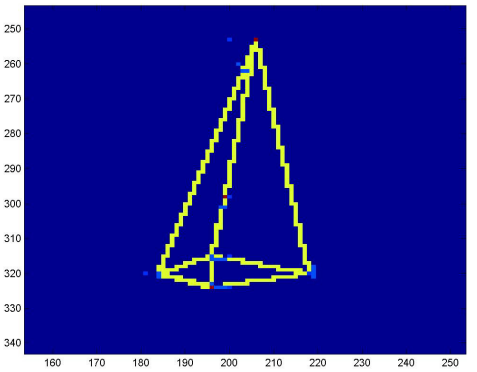


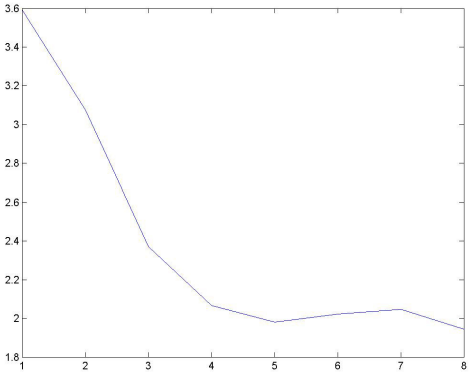
CATT approach

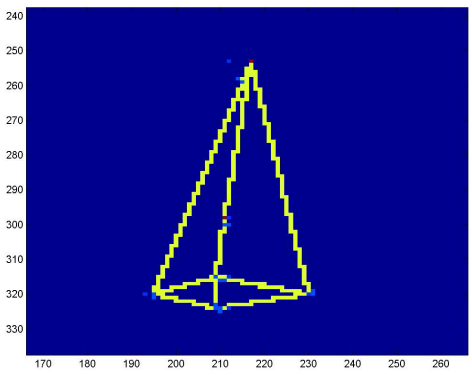
Known centre and scale of the target object.

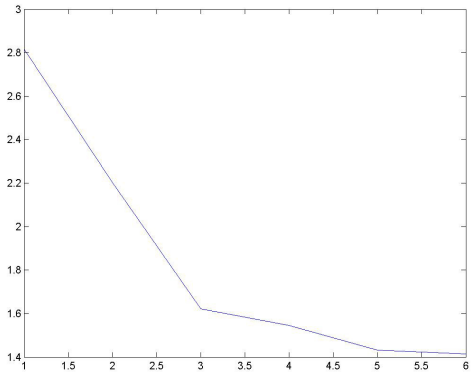


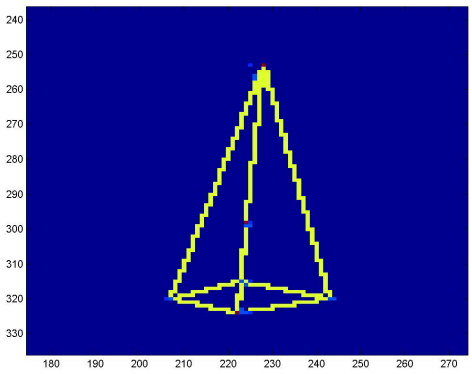


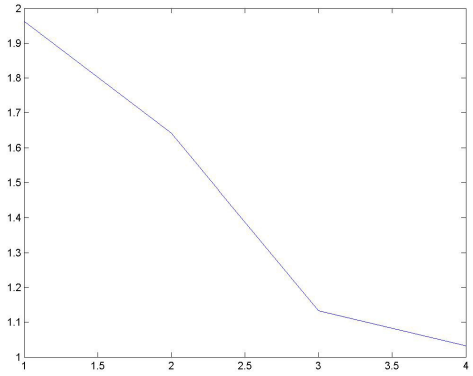


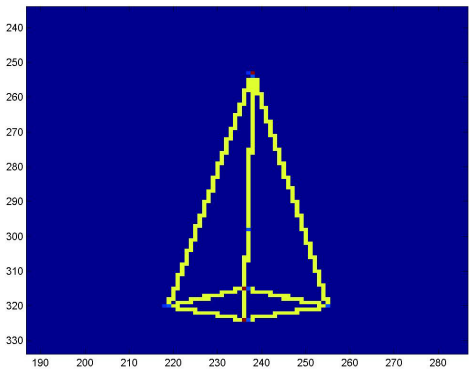


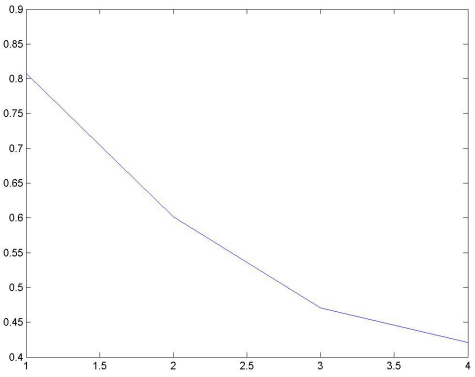

240
250
260
270
280
290
300
310
320
330
170 180 190 200 210 220 230 240 250 260


3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6

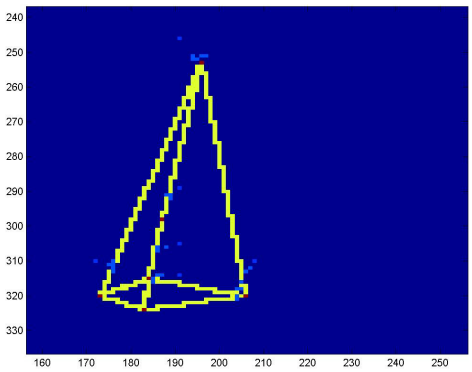
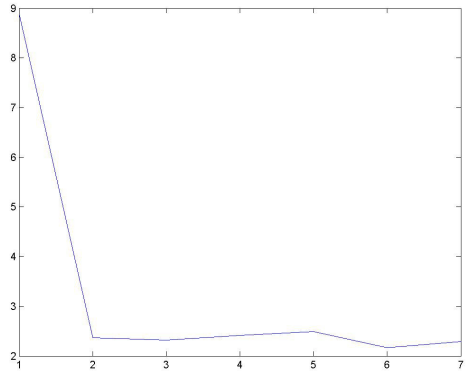
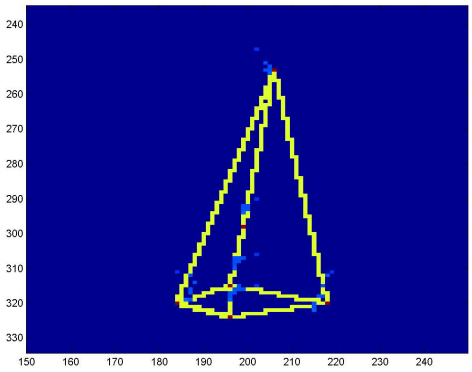
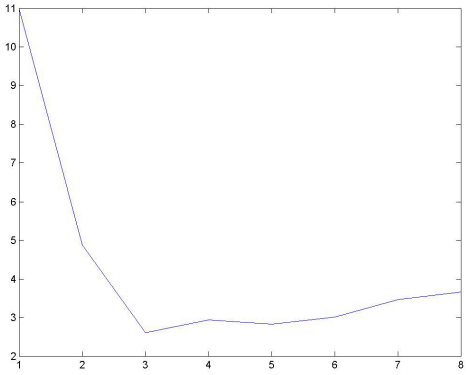
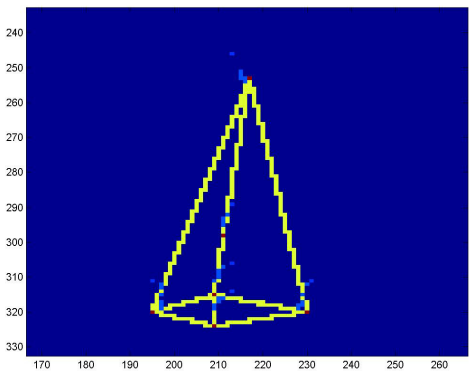
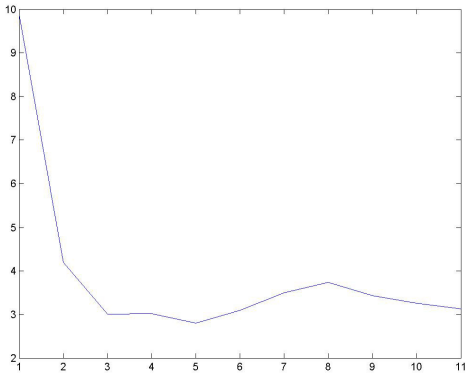
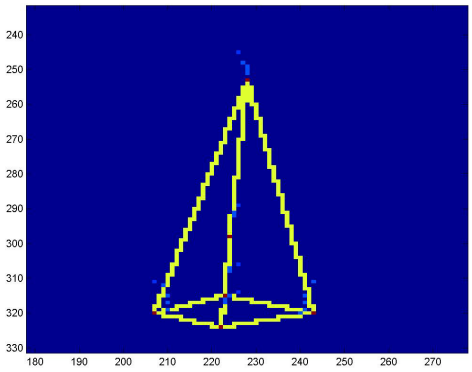
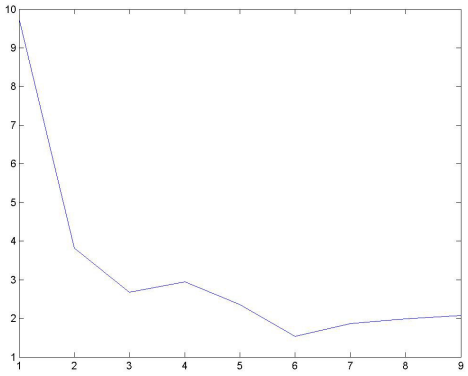

240
250
260
270
280
290
300
310
320
330
180 190 200 210 220 230 240 250 260 270

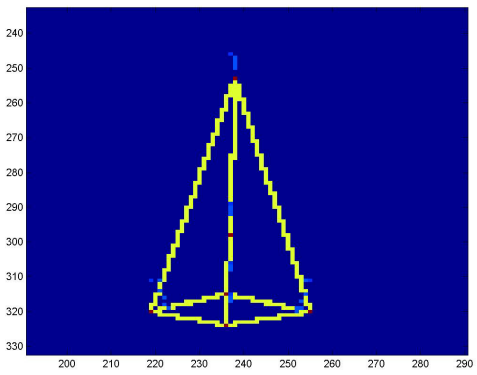

2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
1 1.5 2 2.5 3 3.5 4

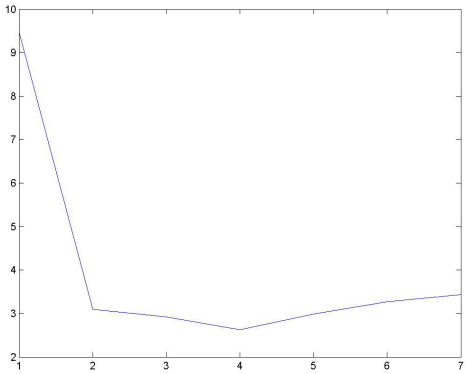

240
250
260
270
280
290
300
310
320
330
190 200 210 220 230 240 250 260 270 280

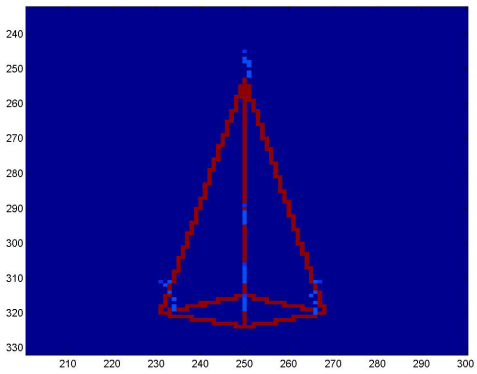

0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
1 1.5 2 2.5 3 3.5 4

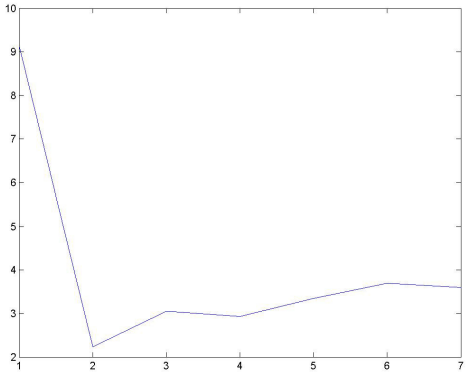
Using an estimate of the centre and scale of the target object.


240
250
260
270
280
290
300
310
320
330
200 210 220 230 240 250 260 270 280 290

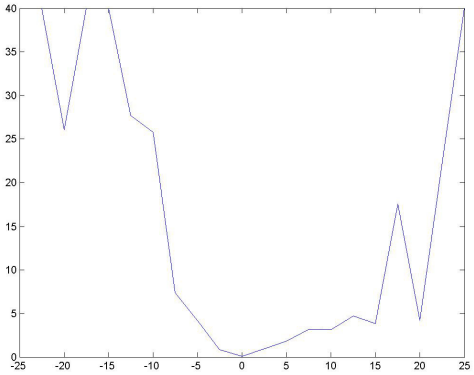

10
9
8
7
6
5
4
3
2
1 2 3 4 5 6 7


240
250
260
270
280
290
300
310
320
330
210 220 230 240 250 260 270 280 290 300

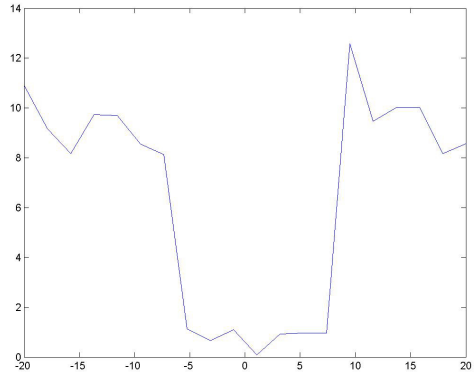

10
9
8
7
6
5
4
3
2
1 2 3 4 5 6 7

Basins of attraction

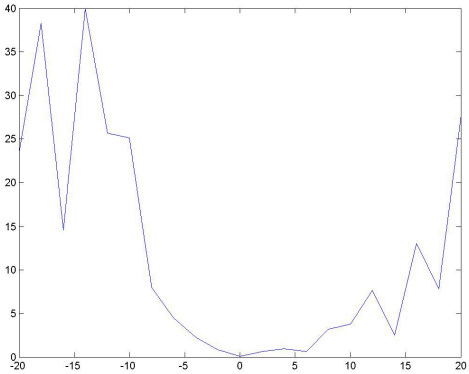
Trans3at56n 5n x d5rect56n



trrans3at56n 5n y d5resct56n

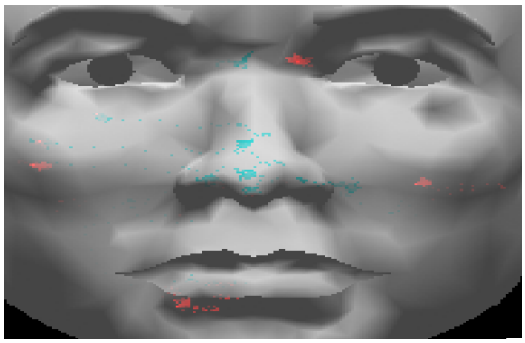


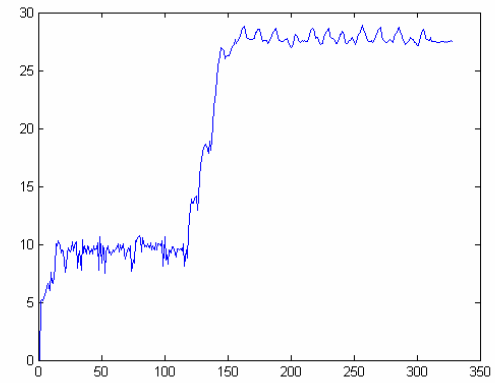
Trans3at56n 5n a rand603y ch6sen d5rect56n

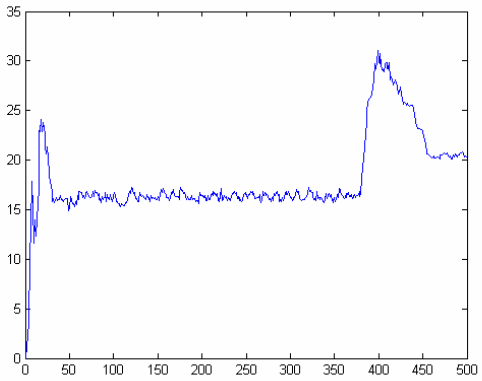


Grey-level approach

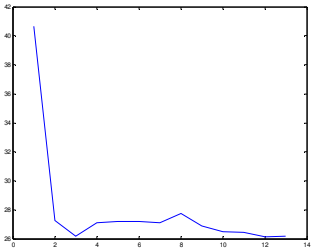
First approach

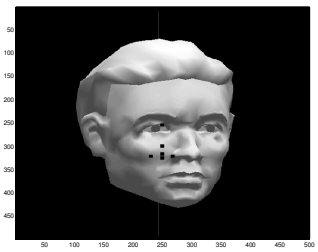


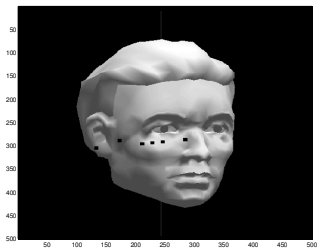


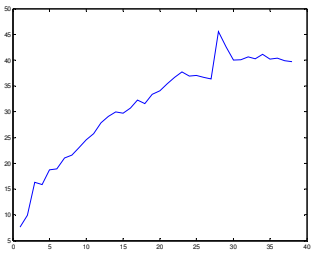


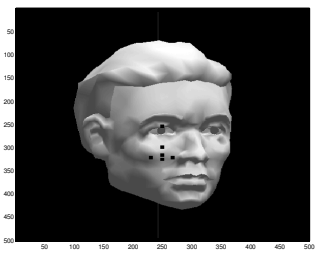
Second approach

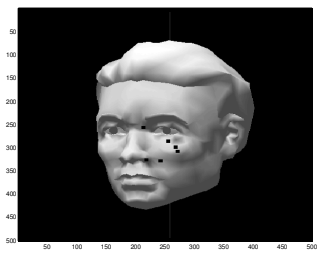


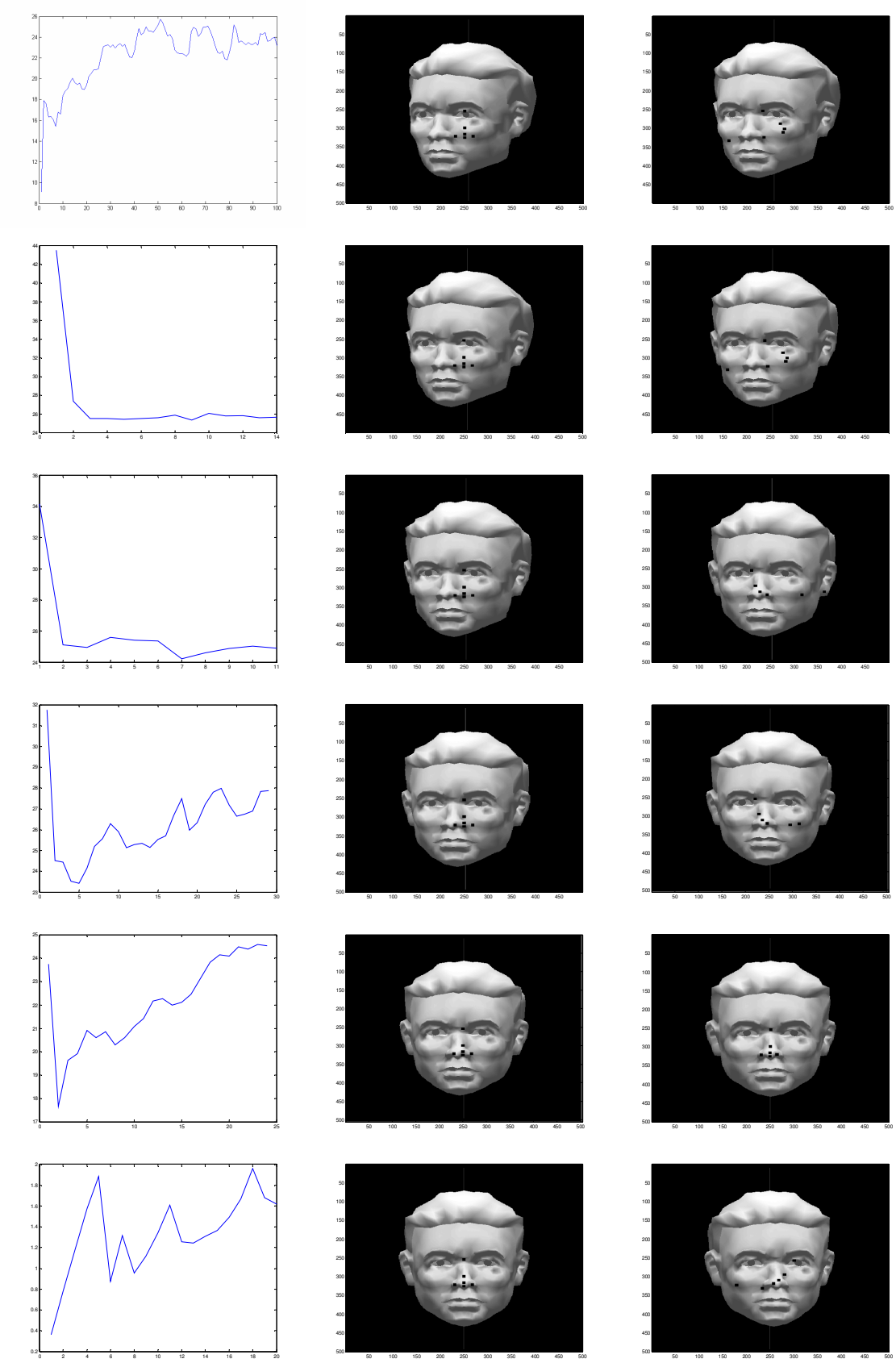


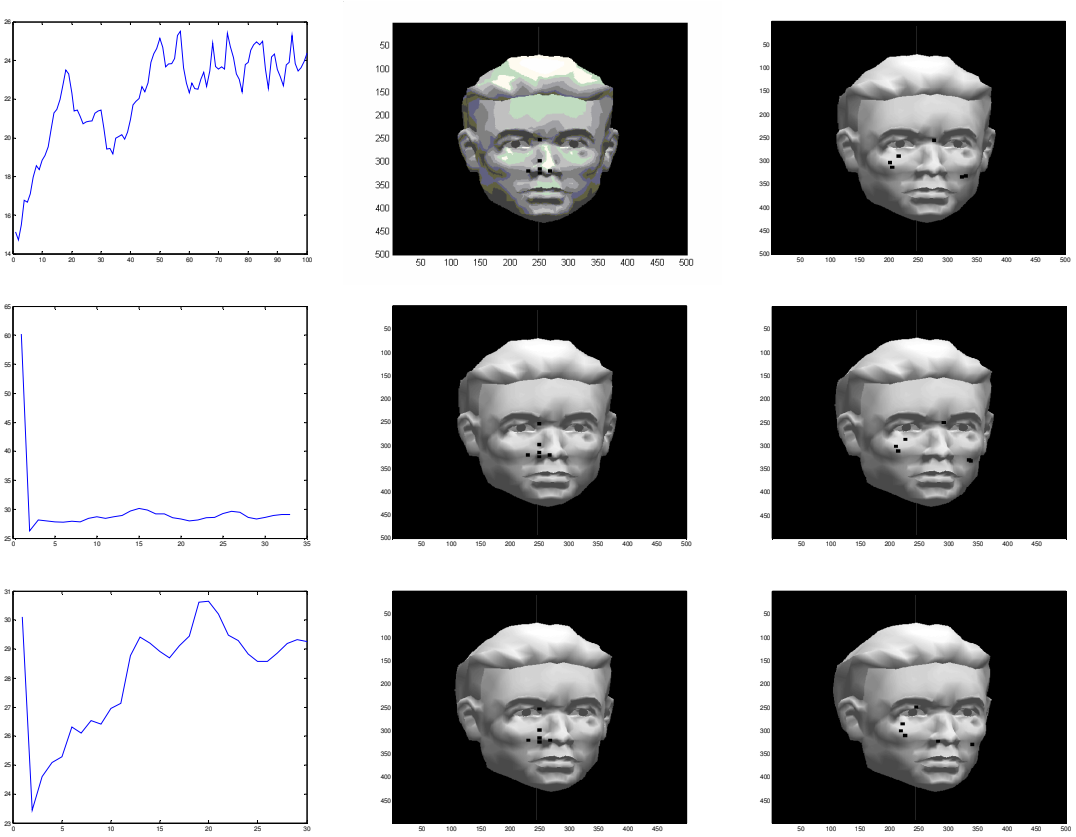




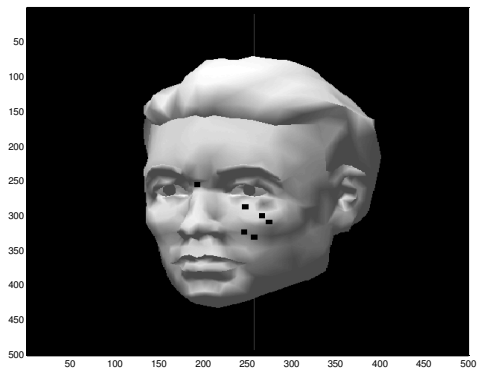


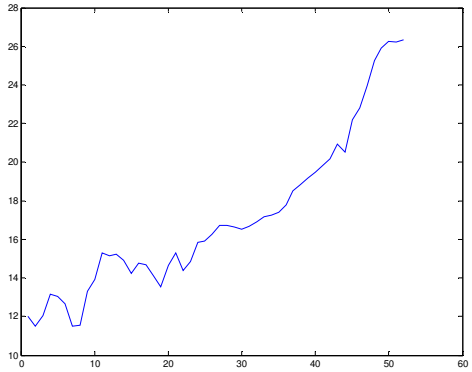


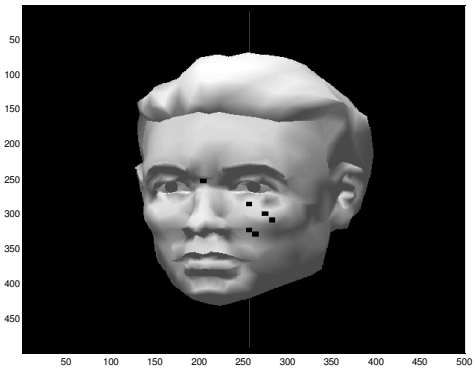


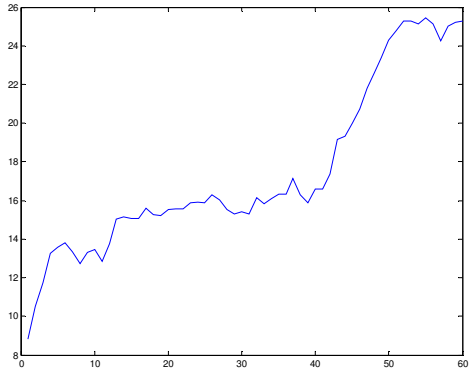


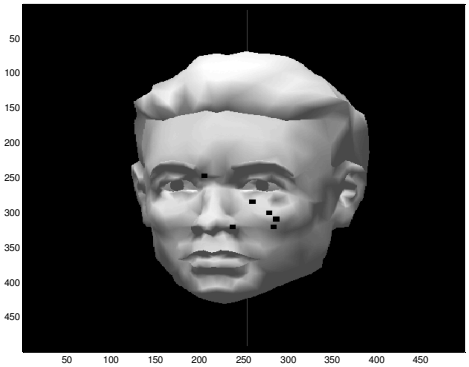
Second approach, removing the subsidiary points and using the correct centre and scale

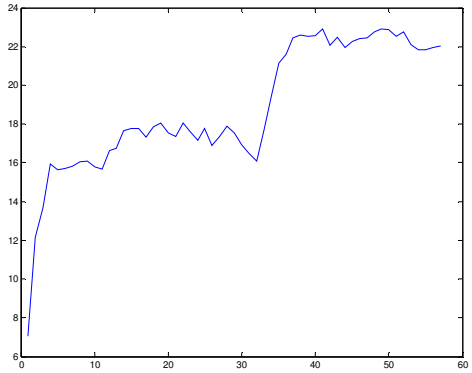


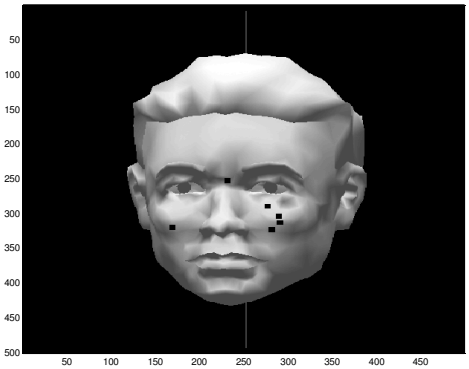


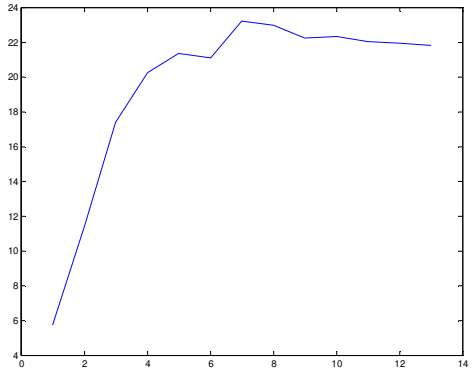


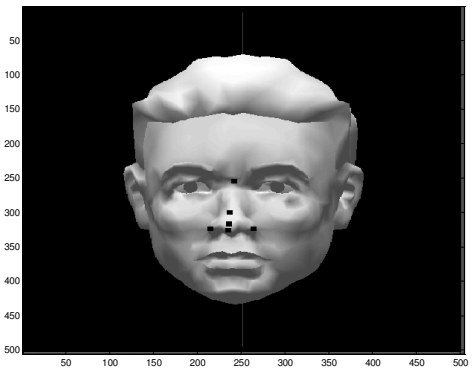


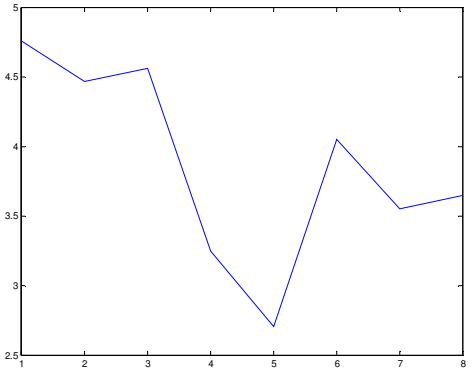


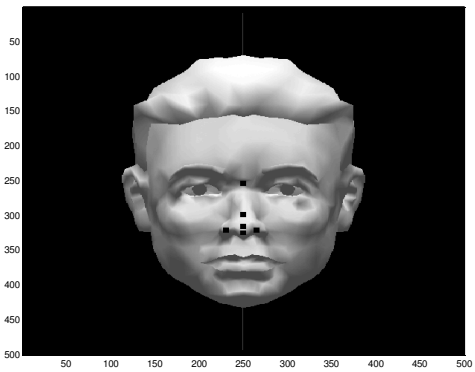
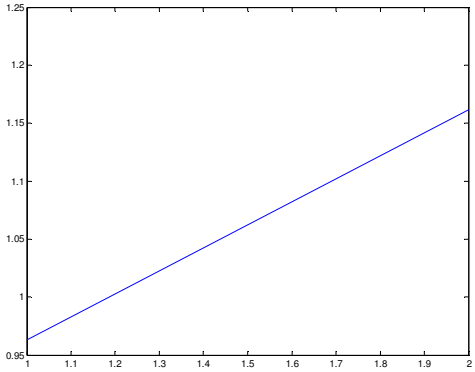
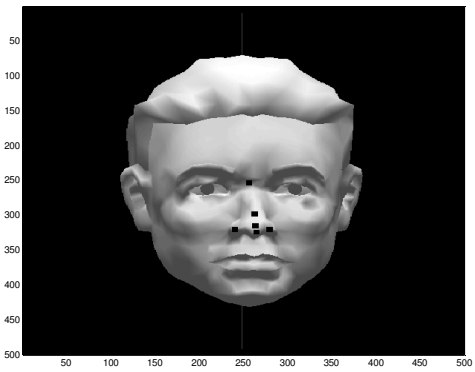
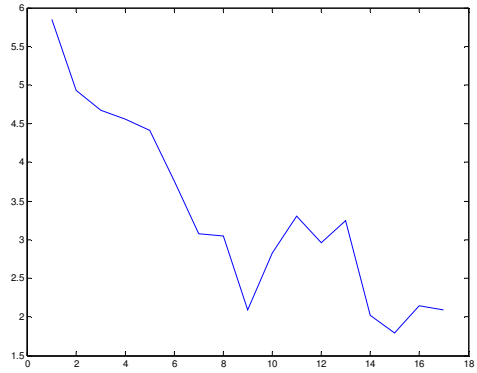
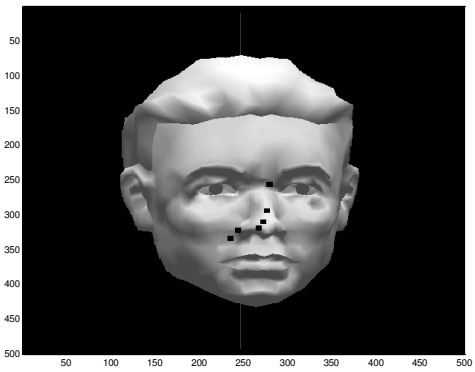
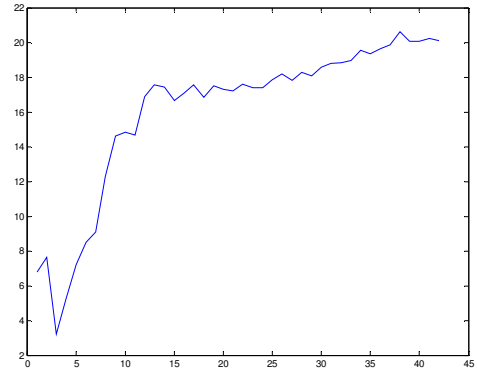
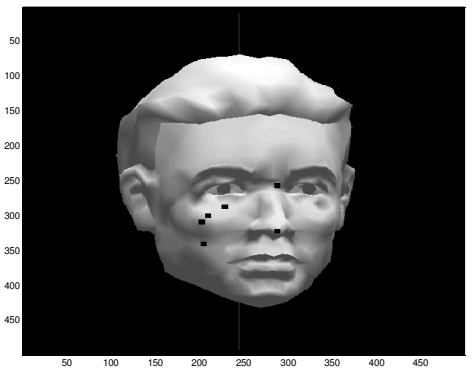
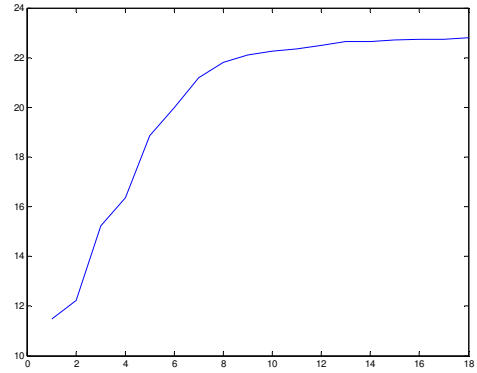


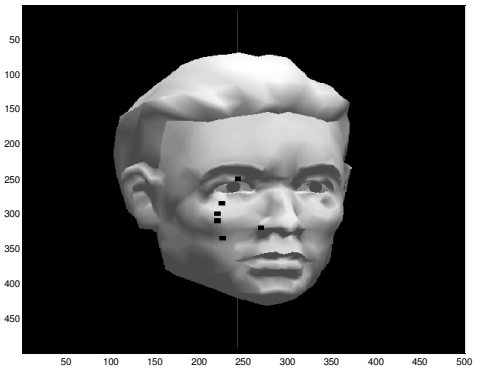


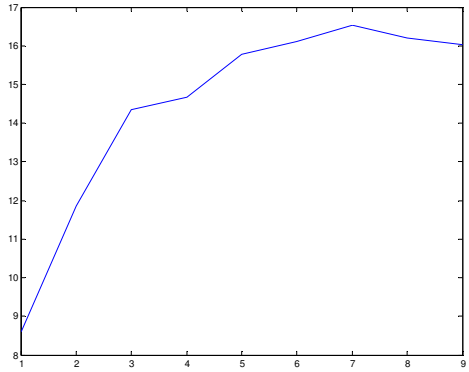


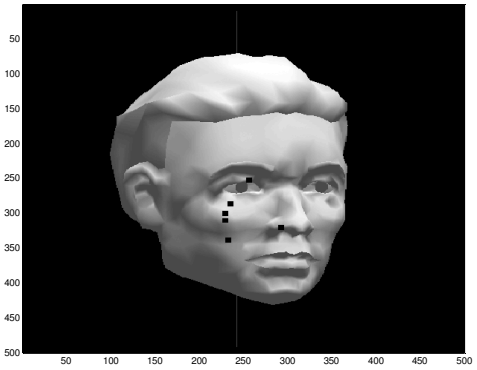


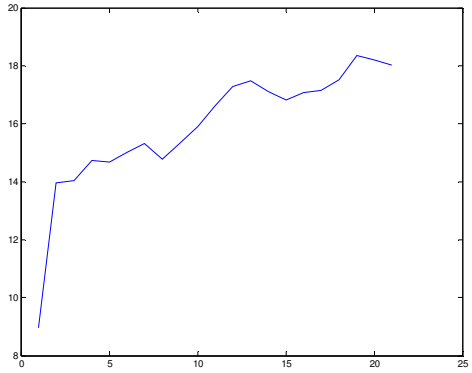


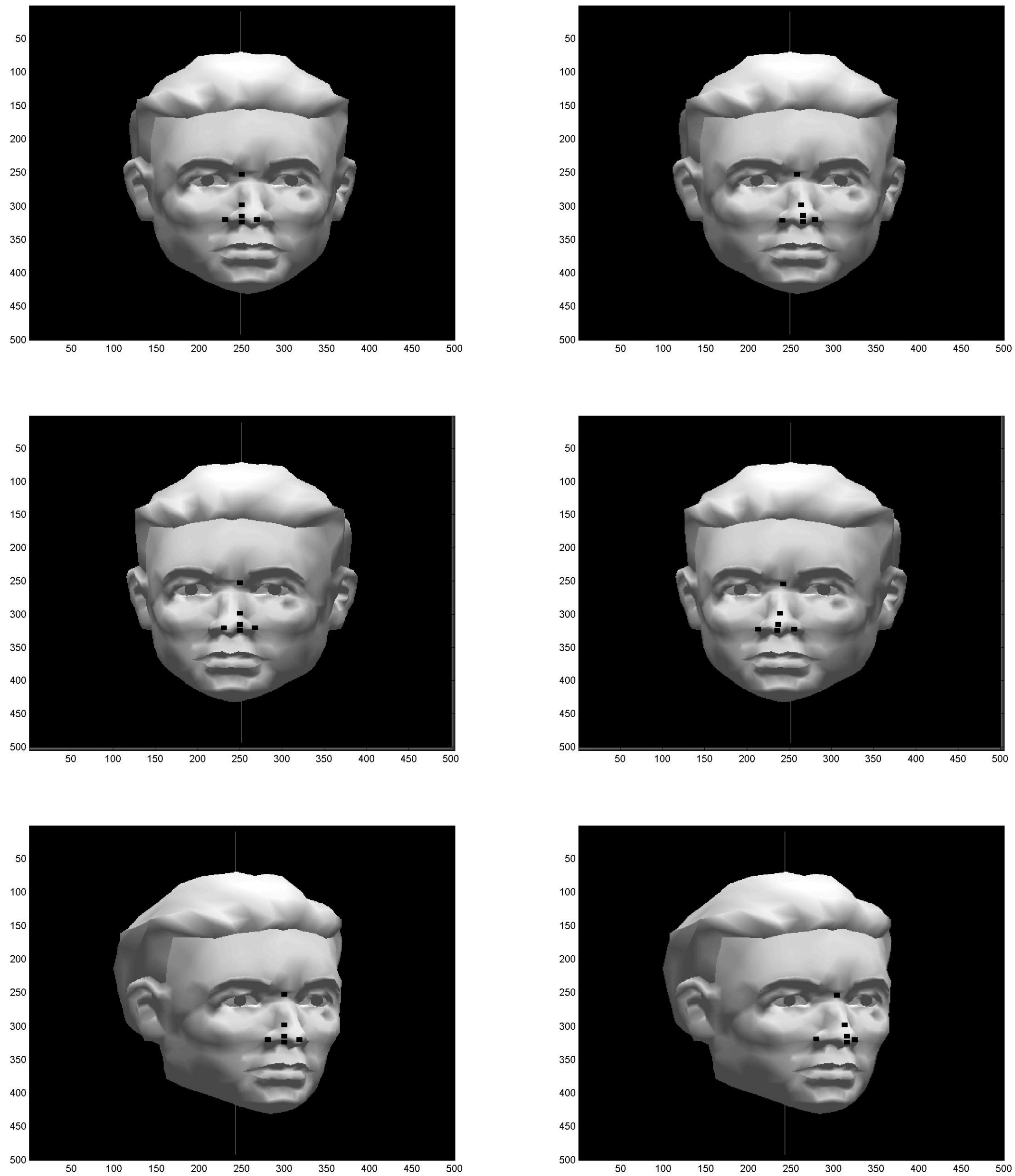

50
100
150
200
250
300
350
400
450
50 100 150 200 250 300 350 400 450 500

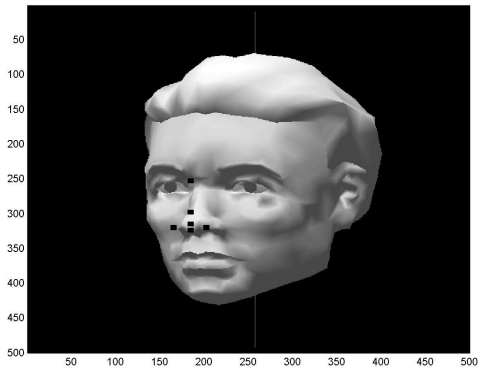

17
16
15
14
13
12
11
10
9
8
1 2 3 4 5 6 7 8 9

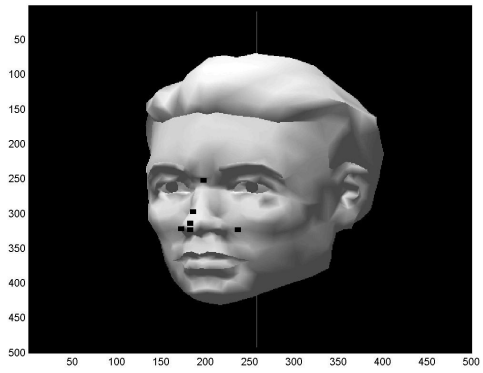

50
100
150
200
250
300
350
400
450
500
50 100 150 200 250 300 350 400 450 500

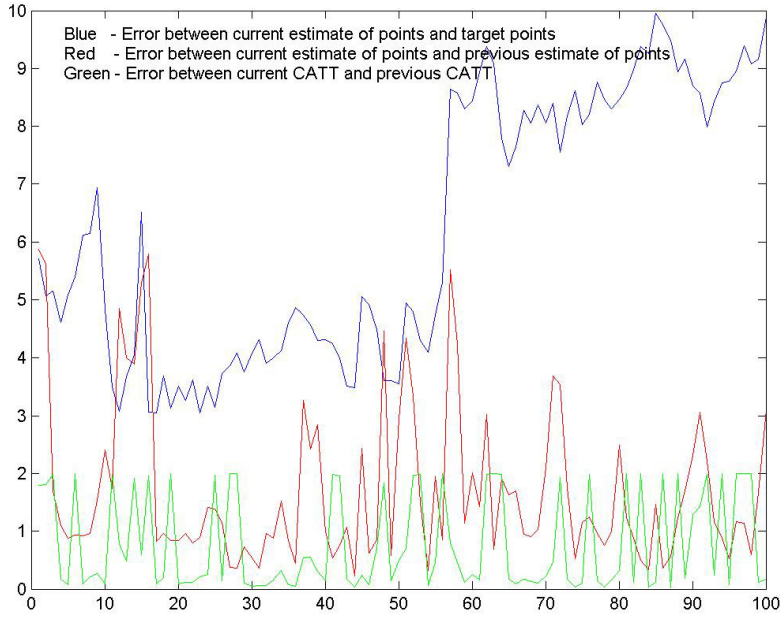

20
18
16
14
12
10
8
0 5 10 15 20 25

Second approach, removing the subsidiary points and but moving the centre and scale to place the starting points close to the target.




50
100
150
200
250
300
350
400
450
500
50 100 150 200 250 300 350 400 450 500


50
100
150
200
250
300
350
400
450
500
50 100 150 200 250 300 350 400 450 500


Blue - Error between current estimate of points and target points
Red - Error between current estimate of points and previous estimate of points
Green - Error between current CATT and previous CATT
0 1 2 3 4 5 6 7 8 9 10
0 10 20 30 40 50 60 70 80 90 100

APPENDIX B

Geometrical approach

Main loop

```
function [error,iterations] = Geometry(a,b,c,d,t1,t2,tol,MAX_NUM_ITER,target_image)

% ************************************************************************************************
% ************************************************************************************************
%
%      The set of given images should be in the following form,
%         saved in an MS Excel spreadsheet.
%
%  Row\Column      1         2         3      ..          n
%   1           x(1,1)    x(1,2)    x(1,3)   ...       x(1,n)
%   2           y(1,1)    y(1,2)    y(1,3)   ...       y(1,n)
%   :             :         :         :      :          :
% (2k-1)        x(k,1)    x(k,2)    x(k,3)   ...       x(k,n)
%   k           y(k,1)    y(k,2)    y(k,3)   ...       y(k,n)
%
%   Here {x(i,j),y(i,j)} is the jth landmark point on the ith image.
%
% ************************************************************************************************
% ************************************************************************************************

% DEFINING THE NOSE POINTS CONNECTIONS (in adjacent order -> for bisectors calculations)
connections=zeros(6,4); %6 points -- 4 possible connections
connections(1,1)=5;connections(1,2)=2;connections(1,3)=6;
connections(2,1)=1;connections(2,2)=3;
connections(3,1)=2;connections(3,2)=6;connections(3,3)=4;connections(3,4)=5;
connections(4,1)=5;connections(4,2)=3;connections(4,3)=6;
connections(5,1)=1;connections(5,2)=3;connections(5,3)=4;
connections(6,1)=1;connections(6,2)=3;connections(6,3)=4;

%-------------------------------------------------------------------------
%   Reading in Training Data
%-------------------------------------------------------------------------
% Reading in the original images from the MS Excel file
%     [fname,pname] = uigetfile(,'Reading in the original images from file ......\n\n');
%     filename = fullfile(pname,fname);
W = xlsread('Landmark Points for processing.xls');
sW = size(W);
k = sW(1)/2;  % The number of images in the set
n = sW(2);   % The number of landmark points in each image

index = 1;
for i = 1:k
   Xin1(1,:,i) = W(index,:); % FER - index could be 2*i -1
   index = index + 1;
   Xin1(2,:,i) = W(index,:); % FER - index could be 2*i
   index = index + 1;
end

clear W;
NosePoints = GetStablePoints(Xin1);

[NosePoints, Centers] = CentreThis(NosePoints);

[NosePoints, scales,MeanRefDist] = ReScale(NosePoints);

clear Xin1;

szx = size(NosePoints);
m = szx(3);


%       % ------------ test - check that image '6' can be recostructed from T -----------------
%       X=getNewX(NosePoints(:,:,1),NosePoints(:,:,m),T);
%       X=UnScale(X,scales(6));
```

```
%       X=UnCentre(X,Centers(6,:));
%       % make line drawing of generated image
%       I=zeros(1,400,400); % FER - CHANGE - size of image?
%       I(1,:,:)=DrawLine(X(:,1),X(:,2),I(1,:,:));
%       I(1,:,:)=DrawLine(X(:,1),X(:,5),I(1,:,:));
%       I(1,:,:)=DrawLine(X(:,1),X(:,6),I(1,:,:));
%       I(1,:,:)=DrawLine(X(:,2),X(:,3),I(1,:,:));
%       I(1,:,:)=DrawLine(X(:,3),X(:,4),I(1,:,:));
%       I(1,:,:)=DrawLine(X(:,3),X(:,5),I(1,:,:));
%       I(1,:,:)=DrawLine(X(:,3),X(:,6),I(1,:,:));
%       I(1,:,:)=DrawLine(X(:,4),X(:,5),I(1,:,:));
%       I(1,:,:)=DrawLine(X(:,4),X(:,6),I(1,:,:));
%       % Showing result
%       figure(666);
%       colormap(gray);
%       imagesc(squeeze(I(1,:,:)));

%---------------------------------- DRAWING - READ IMAGE AND COMPUTE CENTRES AND SCALE, just as a guess
I=imread(['drawing' num2str(target_image) '.bmp']);
[M,Points,centre,scale]=Image2Matrix(I,MeanRefDist); %we need MeanRefDist to scale
% % NOTE: Points in any order!!!!!!!!... connections inconsistent - Should not use Points - may use I or M
% %if we only had the image, we wouldnt be able to calculate the error

%this two following lines are to do the centred and scaled testing and the basins of attraction
centre=Centers(target_image,:); % test... remove this
scale=scales(target_image);     % test... remove this
%test for basis of attracton of AFFINE/CATT_translation in x ('y' in matlab)
centre(1)=centre(1)+t1;
centre(2)=centre(2)+t2;

target=NosePoints(:,:,target_image); %for error supervising purposes

%---------------------------------- FIND INITIAL POSE -------------------------

% get frontal view
%T=getNewT(NosePoints(:,:,1),NosePoints(:,:,m),NosePoints(:,:,6)); % using basis and frontal view (#6)
%FV=getNewX(NosePoints(:,:,1),NosePoints(:,:,m),T);
FV=NosePoints(:,:,6);

%---------------------------------- PREVIOUS TEST ---- <> ---------------

%FV=FV(:,[1 4 5 6]); %QUAD
szfv=size(FV);
num_corners=szfv(2);
% defining the quad point connections
connections_quad=zeros(num_corners,2); %4 points, 2 connections
connections_quad(1,1)=3; connections_quad(1,2)=4;
connections_quad(2,1)=3; connections_quad(2,2)=4;
connections_quad(3,1)=1; connections_quad(3,2)=2;
connections_quad(4,1)=1; connections_quad(4,2)=2;
%       connections=connections_quad; %THIS LINE TO TEST FOR THE KITE (QUAD)
global NUM_MIDDLE_POINTS; NUM_MIDDLE_POINTS=2; %DEFINITION
global SIZE_PROFILE; SIZE_PROFILE=4; %number of pixels in one wing of the profile (excluding the central one)

%warp starting point- SIMULATION
[FV,A]=AffineTrans(FV,a,b,c,d,t1,t2);

%testing with known inputs
%diamond within diamond
%        FV=[0 0 -20 20;20 -20 0 0];
%       target=[ 0 0 -10 10;10 -10 0 0];
%one shifted to the right
%       FV=[0 0 -20 20;20 -20 0 0];
%       target=[ 5 5 -15 25;10 -10 0 0];

%target=AddNoise(target);

% SEARCH - looking for the "correctly affined" from each new Frontal View
FV_old=FV+50; %just to start the first time
iterations=1;

%centre to image space and then uncentre back with the guessed centre, just to show the initial error
Y=UnScale(FV,scales(target_image));
Y=UnCentre(Y,Centers(target_image,:));
Y=UnCentre(Y,-centre);
Y=UnScale(Y,1/scale);
```

```
errors(iterations)=norm(Y-target,'fro')/sqrt(num_corners);

% %image show
% S=I;
% Y=UnScale(FV,scale);
% Y=UnCentre(Y,centre);
% S=DrawPoints(Y,S,40);

% Option 1: Add middle points here (once, and then work with them... only taking care when calculating bisectors). Stopping rule
may take middle points into account too
[FV,look_up]=AddMiddlePoints(FV,connections);
[basis1,look_up]=AddMiddlePoints(NosePoints(:,:,1),connections);
[basis2,look_up]=AddMiddlePoints(NosePoints(:,:,m),connections);

% Structure of FV [corner points    middle points between corner #1 and #2
while ((StoppingRule(FV_old(:,[1:num_corners]),FV(:,[1:num_corners]),tol) & (iterations < MAX_NUM_ITER)) | (iterations==1))
   FV_i=UnScale(FV,scale);   %taking FV to image space of the target image (guessed)
   FV_i=UnCentre(FV_i,centre); %taking FV to image space of the target image (guessed)
   points_found=FindPointsDrawing(FV_i,I,connections,look_up); %Drawing mode
   %remove points not found from basis and 'points_found'
   [basis1_aux,basis2_aux,points_found,num_corners_found]=RemoveNotFound(basis1,basis2,points_found); %useless with
greylevels
   if (num_corners_found<4) %for the AFFINE case we need 4 points IN TOTAL, for the CATT 3, we will generalise to 4
      fprintf('WARNING: less than 4 corner points found. EXITING\n');
      break; %
   elseif (num_corners_found<6) %some corner points not found
      %the centre and scale calculation will be very biased, so do not update them, use the old ones
      points_found=UnCentre(points_found,-centre); %centering
      points_found=UnScale(points_found,1/scale); %scaling
   else
      centre=UpdateCentre(points_found(:,[1:num_corners]));%updating centre with the corner points only, !not centering
      points_found=UnCentre(points_found,-centre); %centering ALL the points
      if ( (points_found(:,1)==[0 0]') & (points_found(:,2)==[0 0]') ) %points found are [0 0]
         fprintf('ERROR: points found are [0 0], unknown reason. EXITING\n');
         break;
      end
      scale=UpdateScale(points_found(:,[1:num_corners]),MeanRefDist);%updating scale with the corner points only, !not scaling
      points_found=UnScale(points_found,1/scale); %scaling
   end
   %           % AFFINE - Find the transformation newA that is a correct affine trasformation of FV from point_found
   %           %make homogenous (to obtain a 3x3 newA - Homography)
   %           %points_found(3,:)=1; %allow for scaling
   %           %FV(3,:)=1; %allow translation
   %           newA=points_found*pinv(FV);
   %           FV_old=FV([1 2],:); %remove homogenity
   %           FV=newA*FV;
   %           FV=FV([1 2],:); %remove homogenity

   %Affine 2:1
   %     [a,b]=GetNewab(basis1_aux,basis2_aux,points_found);
   %     FV_old=FV;
   %     FV=GetNewXAffine(basis1,basis2,a,b);

   %CATT
   T=GetNewT(basis1_aux,basis2_aux,points_found);
   FV_old=FV;
   FV=GetNewX(basis1,basis2,T);

   iterations=iterations+1;

   % % view FV_old FV - using centres and scales from '6', for example
   % if option 1, DOnt take into account middle points! (remove them temporarily)
   %     Y=UnScale(FV(:,[1:num_corners]),scale);
   %     %           Y=UnScale(FV,scales(6));
   %     %
   %     Y=UnCentre(Y,centre);
   %     %           % I=DrawPoints(X,I,150);
   %     S=DrawPoints(Y,S,ceil((100+iterations)/((100+MAX_NUM_ITER)/255)));%the operations are just to let the pixels have
fancy colours

   errors(iterations)=norm(FV(:,[1:num_corners])-target,'fro')/sqrt(num_corners);
end
error=errors(iterations);
```

Auxiliary functions

```
%cuts out the points not found ([666 666] in 'points_found'), from points_found and the basis views
%returns number of corner points found (to update centre and scale with the corner points found only.
%We suppose there are 6 corner points, could calculate it form num_points and NUM_MIDDLE_POINTS
function [basis1_aux,basis2_aux,points_found,num_corners_found]=RemoveNotFound(basis1,basis2,points_found)
indices=find(points_found(1,:)==666);
basis1_aux_1=basis1(1,:);
basis1_aux_1(indices)=[];
basis1_aux_2=basis1(2,:);
basis1_aux_2(indices)=[];
basis1_aux=[basis1_aux_1;basis1_aux_2];

basis2_aux_1=basis2(1,:);
basis2_aux_1(indices)=[];
basis2_aux_2=basis2(2,:);
basis2_aux_2(indices)=[];
basis2_aux=[basis2_aux_1;basis2_aux_2];

points_found_1=points_found(1,:);
points_found_1(indices)=[];
points_found_2=points_found(2,:);
points_found_2(indices)=[];
points_found=[points_found_1;points_found_2];

corners=find(indices<7);
num_corners_found=6-length(corners);

%OUT: look_up table of corner points involved in each connection number, used when calculating perpendicular lines to middle
points
function [added,look_up]=AddMiddlePoints(P,con)
global NUM_MIDDLE_POINTS;
szc=size(con);
num_points=szc(1);
num_con=szc(2);
added=P;
count=1;
%calculate middle points
for i=1:num_points
   for j=1:num_con
      if(con(i,j)==0) %no more connections for this point
         break;
      end
      if(i<con(i,j)) %asure only once per connection
         step=(P(:,i)-P(:,con(i,j)))/(NUM_MIDDLE_POINTS+1);
         for k=1:NUM_MIDDLE_POINTS
            added(:,num_points+count)=P(:,con(i,j))+step*k;
            count=count+1;
         end
         look_up((count-1)/NUM_MIDDLE_POINTS,1)=i;
         look_up((count-1)/NUM_MIDDLE_POINTS,2)=con(i,j);
         %implicit else - connection already dealt with
      end
   end
   %cannot do intersections here because cannot use point info anymore, should check all lines
end


% Not taking middle points into account (could, if option 1)
function bool=StoppingRule(old,new,tol)
szn=size(new);
num_points=szn(2);
dist=norm((old-new),'fro')/sqrt(num_points); % RMS
%dist=sum(sqrt((dif'*dif)));
if (dist<tol)
   bool=0; %stop
else
   bool=1; %continue
end

%*****************************************************************************************************
% FindQuad - Bisectors (DoBisector), Perpendicular, Intersections
(IntersectionsImage,CalculateIntersection,DoLine2Points,DoLine)
```

```
%*************************************************************************************************

function found=FindPointsDrawing(X,I,con,look_up)
bisectors=Bisectors(X,con); %own lines -> Bisectors2
bisectors=Perpendiculars(X,con,look_up,bisectors); %maybe dont need con
found=IntersectionsImage(bisectors,X,I); %X to know where to start

function found=FindPoints(X,P,con,look_up) %pure geometry - check to accept middle points
bisectors=Bisectors(X,con); %own lines -> Bisectors2
bisectors=Perpendiculars(X,con,look_up,bisectors); %maybe dont need con
found=Intersections(bisectors,X,P,con); %X necessary for distances, con necessary to calculate line equ for P

function found=FindPoints2(X,P,con) %neighbourhood alternative - pure geometry
linesP=DoLineEqsFromPoints(P,con);
szx=size(X);
num_points=szx(2);
found=zeros(2,num_points);
bisectors_calculated=0; %if one point fail, only calculate "bisector alternative" once
for i=1:num_points
   aux=Neighbour(X(:,i),linesP,P)'; %P necessary to limit intersections
   if ((aux(1)==0) & (aux(2)==0) & bisectors_calculated==0) %neighbour not found, first time
      found=FindPoints(X,P,con);
      bisectors_calculated=1;
      %break; %witout 'break' rest of points calculated by neighbourhood if possible
   elseif ((aux(1)~=0) | (aux(2)~=0)) %neighbourhood found
      found(:,i)=aux;
      %implicit else - neighbourhood not found, second time-> do nothing
   end
end

function closest=NeighbourImage(start,I)
closest=start;
step=1;
limit=100/step;
iter=1;
done=0;
while ((iter<limit) & (done==0))
   belongs=CheckImage([start(1)+step*iter start(2)],I);%P necessary to limit intersections
   if (belongs)
      done=1;
      closest=[start(1)+step*iter start(2)];
      break;
   end
   belongs=CheckImage([start(1)-step*iter start(2)],I);%P necessary to limit intersections
   if (belongs)
      done=1;
      closest=[start(1)-step*iter start(2)];
      break;
   end
   belongs=CheckImage([start(1) start(2)+step*iter],I);%P necessary to limit intersections
   if (belongs)
      done=1;
      closest=[start(1) start(2)+step*iter];
      break;
   end
   belongs=CheckImage([start(1) start(2)-step*iter],I);%P necessary to limit intersections
   if (belongs)
      done=1;
      closest=[start(1) start(2)-step*iter];
      break;
   end
   belongs=CheckImage([start(1)+step*iter start(2)+step*iter],I);%P necessary to limit intersections
   if (belongs)
      done=1;
      closest=[start(1)+step*iter start(2)+step*iter];
      break;
   end
   belongs=CheckImage([start(1)-step*iter start(2)+step*iter],I);%P necessary to limit intersections
   if (belongs)
      done=1;
      closest=[start(1)-step*iter start(2)+step*iter];
      break;
   end
   belongs=CheckImage([start(1)-step*iter start(2)+step*iter],I);%P necessary to limit intersections
   if (belongs)
      done=1;
```

```
        closest=[start(1)-step*iter start(2)+step*iter];
        break;
    end
    belongs=CheckImage([start(1)+step*iter start(2)-step*iter],I);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1)+step*iter start(2)-step*iter];
        break;
    end
    iter=iter+1;
end
if (done==0)
    fprintf('WARNING: Neighbour not found...  switching to bisectors!');
    closest=[0 0];
end


function closest=Neighbour(start,linesP,P)
closest=start;
step=1;
limit=100/step;
iter=1;
done=0;
while ((iter<limit) & (done==0))
    belongs=Check([start(1)+step*iter start(2)],linesP,P);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1)+step*iter start(2)];
        break;
    end
    belongs=Check([start(1)-step*iter start(2)],linesP,P);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1)-step*iter start(2)];
        break;
    end
    belongs=Check([start(1) start(2)+step*iter],linesP,P);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1) start(2)+step*iter];
        break;
    end
    belongs=Check([start(1) start(2)-step*iter],linesP,P);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1) start(2)-step*iter];
        break;
    end
    belongs=Check([start(1)+step*iter start(2)+step*iter],linesP,P);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1)+step*iter start(2)+step*iter];
        break;
    end
    belongs=Check([start(1)-step*iter start(2)+step*iter],linesP,P);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1)-step*iter start(2)+step*iter];
        break;
    end
    belongs=Check([start(1)-step*iter start(2)+step*iter],linesP,P);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1)-step*iter start(2)+step*iter];
        break;
    end
    belongs=Check([start(1)+step*iter start(2)-step*iter],linesP,P);%P necessary to limit intersections
    if (belongs)
        done=1;
        closest=[start(1)+step*iter start(2)-step*iter];
        break;
    end
    iter=iter+1;
end
if (done==0)
    fprintf('WARNING: Neighbour not found...  switching to bisectors!');
```

```
   closest=[0 0];
end

function belongs=CheckImage(x,I)  %line drawing
belongs=I(round(x(1)),round(x(2)));

function belongs=Check(x,linesP,P) %pure geometry
belongs=0;
szl=size(linesP);
num_points=szl(1);
%loop through top part of line-matrix
for k=1:num_points-1
   for l=k+1:num_points
      if((linesP(k,l,1)~=0) & (belongs==0)) %there is a line between these points
         inside=LimitIntersection(x,P(:,k),P(:,l));
         if (inside)
            if    (abs(linesP(k,l,1)*x(1)+linesP(k,l,2)*x(2)+linesP(k,l,3))<=abs(linesP(k,l,1)*0.5+linesP(k,l,2)*0.5))    %    a*0.5+b*0.5
maximum error with step 1
               belongs=1;
               break;
            end
         end
      end
   end
end % for - lines

%%%%%%%%%%%%%%%%%                                                                                                    Intersections
(DoLine,DoLine2Poinnts,CalculateIntersection,DoLineEqusFromPoints,Intersections,LimitIntersection,IntersectionsImage)

%check the image along the line, starting from x, in a 4-connected way (not to miss a 8-connected drawing)
function int=CalculateIntersectionImage(x,line,I) %cheking in both directions (and with image limit control)
v(1)=-line(2);
v(2)=line(1);
%find the ratio (how many units we must go along one dimension for 1 unit of the other)     NOTE: maintaining the signs!
if (v(1)==0)
   ratio=[0 sign(v(2))]; %vertical line [0 1] [0 -1]
else
   ratio=[sign(v(1)) v(2)/abs(v(1))]; % [+/-1 +/-xxx] xxx can be '0' and 0.xxx
   %we want the smallest to be 1 (and the other more than 1) (except in the case [1 0]), so...
   if ((ratio(2)~=0) & (abs(ratio(2))<1))
      ratio=[sign(ratio(1))/abs(ratio(2)) sign(ratio(2))]; % [+/-xxx +/-1]
   end
end
%round the larger component and calculate remainder (probability of adding an extra pixel to the larger component)
if (abs(ratio(1))==1) %second component larger    NOTE: if [1 0], second component will be classified as larger, and remainder will
be '0', this should be ok
   larger_component=2;
   int_part=floor(abs(ratio(2)));
   remainder=abs(ratio(2))-int_part; %probablility of adding an extra pixel to the largest component
   ratio(2)=int_part*sign(ratio(2));
else %first component larger
   larger_component=1;
   int_part=floor(abs(ratio(1)));
   remainder=abs(ratio(1))-int_part;
   ratio(1)=int_part*sign(ratio(1));
end

szi=size(I);
found=0;
image_end=0;
offset=[0 0];
x=round(x');
acum=remainder; %will tell if we have to add an extra pixel (if >0.5)
if ( max(x>szi) | max(x<[0 0]) )  %limit not reached in one side - else exit
   fprintf('ERROR:out of the image!!!\n');
   int=[666 666];
   break;
end

if (I(x(1),x(2))>0) % check starting point
   found=1;
   int=x+offset;
end
while ((found==0) & (image_end==0))
   for i=1:abs(ratio(1))
      offset(1)=offset(1)+sign(ratio(1));
```

```
      if (   ( min(x+offset<=szi) & min(x+offset>[0 0]) ) | ( min(x-offset<=szi) & min(x-offset>[0 0]) )   ) %limit not reached in one
side - else exit
         if ( min(x+offset<=szi) & min(x+offset>[0 0]) ) %limit not reached in this direction - else ignore this direction
            if (I(x(1)+offset(1),x(2)+offset(2)))
               found=1;
               int=x+offset;
               break;
            end
         end
         if ( min(x-offset<=szi) & min(x-offset>[0 0]) ) %limit not reached in this direction - else ignore this direction
            if (I(x(1)-offset(1),x(2)-offset(2)))
               found=1;
               int=x-offset;
               break;
            end
         end
      else
         image_end=1;
         break;
      end
   end %for
   if (found==0)
      for i=1:abs(ratio(2))
         offset(2)=offset(2)+sign(ratio(2));
         if (   ( min(x+offset<=szi) & min(x+offset>[0 0]) ) | ( min(x-offset<=szi) & min(x-offset>[0 0]) )   ) %limit not reached in
one side - else exit
            if ( min(x+offset<=szi) & min(x+offset>[0 0]) ) %limit not reached in this direction - else ignore this direction

               if (I(x(1)+offset(1),x(2)+offset(2)))
                  found=1;
                  int=x+offset;
                  break;
               end
            end
            if ( min(x-offset<=szi) & min(x-offset>[0 0]) ) %limit not reached in this direction - else ignore this direction
               if (I(x(1)-offset(1),x(2)-offset(2)))
                  found=1;
                  int=x-offset;
                  break;
               end
            end
         else
            image_end=1;
            break;
         end
      end %for
   end %if found
   %check 'acum' to add an extra pixel-search
   if (found==0)
      if(acum>=0.5) %go one more
         acum=acum+remainder-1;
         offset(larger_component)=offset(larger_component)+sign(ratio(larger_component));
         if (   ( min(x+offset<=szi) & min(x+offset>[0 0]) ) | ( min(x-offset<=szi) & min(x-offset>[0 0]) )   ) %limit not reached in
one side - else exit
            if ( min(x+offset<=szi) & min(x+offset>[0 0]) ) %limit not reached in this direction - else ignore this direction

               if (I(x(1)+offset(1),x(2)+offset(2)))
                  found=1;
                  int=x+offset;
                  break;
               end
            end
            if ( min(x-offset<=szi) & min(x-offset>[0 0]) ) %limit not reached in this direction - else ignore this direction
               if (I(x(1)-offset(1),x(2)-offset(2)))
                  found=1;
                  int=x-offset;
                  break;
               end
            end
         else
            image_end=1;
            break;
         end
      else %acum<1
         acum=acum+remainder;
      end %if acum
```

```
   end %if found
end %while
if (found==0)
   fprintf('WARNING: IntersectionImage not found!');
   int=[666 666]; %not found
end

%not found are returned as [666 666]
function found=IntersectionsImage(bis,X,I)
sz=size(bis);
num_points=sz(1);
num_bis=sz(2);
found=zeros(2,num_points);
szi=size(I);

%calculate intersections
for i=1:num_points
   closest=[666 666];
   closest_dist=99999;
   if ( max(X(:,i)>[szi(1) szi(2)]') | max(X(:,i)<[0 0]') )  %limit not reached in one side - else exit
      fprintf('ERROR:IntersectionsImage: starting point out of the image!!!');
   else
      for j=1:num_bis
         %calculate all intersections of current bisector with all lines, then get the minimum distance
         if ((bis(i,j,1)==0) & (bis(i,j,2)==0))%no more bisectors for this point
            break;
         end
         int=CalculateIntersectionImage(X(:,i),bis(i,j,:),I);
         % if point not found ... try with neighbour ... or do nothing and give less points to GetNewT
         %        if (int==[666 666]) %not found
         %           int=NeighbourImage(X(:,i),I);
         %        end
         if (int~=[666 666])
            %calculate distance between int and X(:,i) - necessary anyway because we might have several bisectors
            dist=sqrt((int(1)-X(1,i))^2 + (int(2)-X(2,i))^2);
            if (dist<closest_dist)
               closest_dist=dist;
               closest=int;
            end
         end
      end % for - bisectors
   end %if out of the image
   found(:,i)=closest';
end % for - points

function eq=DoLine(v,p)
eq(1)=v(2);
eq(2)=-v(1);
eq(3)=v(1)*p(2)-v(2)*p(1);

function eq=DoLine2Points(p1,p2)
eq=DoLine(p1-p2,p1);

function p=CalculateIntersection(line1,line2)
if (line1(2)*line2(1)-line1(1)*line2(2)==0) % no intersection
   p=[999999 999999]; %just to make sure it is not used
else
   p(2)=(line2(3)*line1(1)-line1(3)*line2(1))/(line1(2)*line2(1)-line1(1)*line2(2));
   if (line1(1)==0) %avoid dividing by zero
      p(1)= - (line2(2)*p(2)+line2(3))/line2(1);
   else
      p(1)= - (line1(2)*p(2)+line1(3))/line1(1);
   end
end

function linesP=DoLineEqsFromPoints(P,con)
szc=size(con);
num_points=szc(1);
num_con=szc(2);
%calculate line equations of P
linesP=zeros(num_points,num_points,3+2+2); %simetric matrix with line_eq in its elements. If '0'-> no connection. Contains
a,b,c,p1_x,p1_y,p2_x,p2_y
for i=1:num_points
   for j=1:num_con
      if(con(i,j)==0) %no more connections for this point
         break;
```

```
        end
        if(linesP(i,con(i,j),1)==0) %no line calculated yet (suppose 'a' in ax+by+c=0 is not '0', if it was, it will just be calculated again)
            linesP(i,con(i,j),:)=[DoLine2Points(P(:,i),P(:,con(i,j))) P(:,i)' P(:,con(i,j))'];
            linesP(con(i,j),i,:)=linesP(i,con(i,j),:); %to maintain symetry
            %implicit else - line already calculated
        end
    end
    %cannot do intersections here because cannot use point info anymore, should check all lines
end

function found=Intersections(bis,X,P,con)
sz=size(bis);
num_points=sz(1);
num_bis=sz(2);
szc=size(con);
num_corners=szc(1);
num_con=szc(2);
found=zeros(2,num_points);

linesP=DoLineEqsFromPoints(P,con);

%calculate intersections
for i=1:num_points
    closest=zeros(2);
    closest_dist=99999;
    for j=1:num_bis
        %calculate all intersections of current bisector with all lines, then get the minimum distance
        if ((bis(i,j,1)==0) & (bis(i,j,2)==0))%no more bisectors for this point
            break;
        end
        %loop through top part of line-matrix
        for k=1:num_corners-1
            for l=k+1:num_corners
                if(linesP(k,l,1)~=0) %there is a line between these points
                    int=CalculateIntersection(bis(i,j,:),linesP(k,l,:));
                    %check if it is inside the segment
                    inside=LimitIntersection(int,P(:,k),P(:,l));
                    if (inside)
                        %calculate distance between int and X(:,i)
                        dist=sqrt((int(1)-X(1,i))^2 + (int(2)-X(2,i))^2);
                        if (dist<closest_dist)
                            closest_dist=dist;
                            closest=int;
                        end
                    end
                end
            end
        end % for - lines
    end % for - bisectors
    found(1,i)=closest(1); found(2,i)=closest(2);
end % for - points

% checking in both coordinates (knowing that they must be colinear, checking for one coordinate is enough)
function inside = LimitIntersection(x,p1,p2)
inside=0;
if ((x(1)<=max(p1(1),p2(1))) & (x(1)>=min(p1(1),p2(1))) & (x(2)<=max(p1(2),p2(2))) & (x(2)>=min(p1(2),p2(2))) )
    inside=1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                    Bisectors
(DoBisector,Perpendiculars)

function eq=DoBisector(P,X1,X2)
%vectors
v1=X1-P;
v2=X2-P;
%normalise
v1=v1/(sqrt(v1(1)*v1(1)+v1(2)*v1(2)));
v2=v2/(sqrt(v2(1)*v2(1)+v2(2)*v2(2)));
%bisector vector
v=(v1+v2)/2;
if (v==[0 0]') %opposite vectors
    eq=DoLine([-v1(2) v1(1)],P); %perpendicular vector
else
    eq=DoLine(v,P);
end
```

```
function added=Perpendiculars(X,con,look_up,bisectors) %maybe dont need con
global NUM_MIDDLE_POINTS;
added=bisectors;
%get number of corners ... from con or bisectors or just pass it as parameter
szb=size(bisectors);
num_corners=szb(1);
szx=size(X);
num_points=szx(2);
for i=num_corners+1:num_points
   con_num=ceil((i-num_corners)/NUM_MIDDLE_POINTS);
   added(i,:,:)=0; %initialisation
   vector=X(:,look_up(con_num,1)) - X(:,look_up(con_num,2)); %vector between the corner points (more robust than using the
middle point)
   pvector(1)=-vector(2); %perpendicular vector
   pvector(2)=vector(1);
   added(i,1,:)=DoLine(pvector,X(:,i));
end


% option 2 (all possible combinations of adjacent connections to form bisectors)
% if '0' in con, no more connections for that point
% we want also the bisector of the last connection with the first (except when only 2 connections)
% % % % 'bisectors' structure:
% % % % bisectors(num_corners, num_bisectors (with the '0' at the end if no more), bisector equation (ax + by + c = 0)
% PRE: minimum 2 connections
function bisectors=Bisectors(X,con)
sz = size(con);
num_corners=sz(1);
num_con=sz(2);
if (num_con==2)
   bisectors=zeros(num_corners,1,3);
else
   bisectors=zeros(num_corners,num_con,3);
end

for i=1:num_corners
   do_last_one=1; %TRUE
   for j=2:num_con
      if (con(i,j)==0) %no more connections
         if(j==3) % only 2 connections
            do_last_one=0; %FALSE
         elseif (j<3) % j==2, something wrong
            fprintf('ERROR: only 1 connection !!???');
            do_last_one=0;
         else % 3 or more connections, do last bisector
            j=j-1; %to index correctly when doing the last one
            break;
         end
      else
         bisectors(i,j-1,:)=DoBisector(X(:,i),X(:,con(i,j-1)),X(:,con(i,j)));
      end % if - connections
   end %for - connections
   %do last one if necessary
   if ((do_last_one==1) & (num_con>2))
      bisectors(i,j,:)=DoBisector( X(:,i), X(:,con(i,j)), X(:,con(i,1)) );
   end
end %for - points

% option 2 (all possible combinations of adjacent connections to form bisectors) + OWN LINES
% if '0' in con, no more connections for that point
% we want also the bisector of the last connection with the first (except when only 2 connections)
% % % % 'bisectors' structure:
% % % % bisectors(num_points, num_bisectors (with the '0' at the end if no more), bisector equation (ax + by + c = 0)
% PRE: minimum 2 connections
function bisectors=Bisectors2(X,con)
sz = size(con);
num_points=sz(1);
num_con=sz(2);
if (num_con==2)
   bisectors=zeros(num_points,1 + num_con,3);
   last_bisector_index=1;
else
   bisectors=zeros(num_points,num_con*2,3);
   last_bisector_index=num_con;
end
```

```
for i=1:num_points
   do_last_one=1; %TRUE
   bisectors(i,last_bisector_index+1,:)=DoLine2Points(X(:,i),X(:,con(i,1))); %add first own line
   for j=2:num_con
      if (con(i,j)==0) %no more connections
         if(j==3) % only 2 connections
            do_last_one=0; %FALSE
         elseif (j<3) % j==2, something wrong
            fprintf('ERROR: only 1 connection !!???');
            do_last_one=0;
         else % 3 or more connections, do last bisector
            j=j-1; %to index correctly when doing the last one
            break;
         end
      else
         bisectors(i,j-1,:)=DoBisector(X(:,i),X(:,con(i,j-1)),X(:,con(i,j)));
         % add own lines
         bisectors(i,last_bisector_index+j,:)=DoLine2Points(X(:,i),X(:,con(i,j)));
      end % if - connections
   end %for - connections
   %do last one if necessary
   if ((do_last_one==1) & (num_con>2))
      bisectors(i,j,:)=DoBisector( X(:,i), X(:,con(i,j)), X(:,con(i,1)) );
   end
   %now the own lines

end %for - points

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of FindQuad- Bisectors (DoBisector) , Intersections (IntersectionImage,CalculateIntersection,DoLine2Points,DoLine)
%****************************************************************************************************************************

%------------------------------------------
%   Begining of "AffineTrans" and "AddNoise" function
%-----------------------------------------

function [Xout,A]=AffineTrans(Xin,a,b,c,d,t1,t2)
if ((a==0) & (d==0))
   %define the transformation randomly
   %A=rand(3,3)*2 - 1; %random numbers from -1 .. 1
   %A=rand(3,3)*4 - 2; %random numbers from -2 .. 2
   A=rand(3,3) - 0.5; %random numbers from -0.5 .. 0.5
   A(1,3)=0; A(2,3)=0; %no translation
   A(3,1)=0; A(3,2)=0; %no scaling
   A(3,3)=1;
   A(1,1)=A(1,1)+1; A(2,2)=A(2,2)+1; %from 0.5 .. 1.5
else
   %use the one given
   A=zeros(3,3);
   A(1,1)=a;A(1,2)=b;A(2,1)=c;A(2,2)=d;A(1,3)=t1;A(2,3)=t2;A(3,3)=1;
end
szx = size(Xin);
m = szx(2);
Xout=zeros(2,m);
for i=1:m
   w= A(3,1)*Xin(1,i) + A(3,2)*Xin(2,i) + A(3,3);
   Xout(1,i)= (A(1,1)*Xin(1,i) + A(1,2)*Xin(2,i) + A(1,3))/w;
   Xout(2,i)= (A(2,1)*Xin(1,i) + A(2,2)*Xin(2,i) + A(2,3))/w;
end
%why not matrix form? Xout=A*Xin;

function X=AddNoise(X)
szx=size(X);
m=szx(2);
noise=rand(2,m)*10-5;
X=X+noise;

%------------------------------------------
%   End of "AffineTrans" and "AddNoise" function
%-----------------------------------------

%------------------------------------------
%   Begining of "Image2Matrix" function - FER
%-----------------------------------------
```

```
% Calculate centre & scale, but results in image space
function [Xout,Points,centre,scale]=Image2Matrix(I,MeanRefDist)

szx = size(I);
m = szx(1);
n = szx(2);
totX=0; %to calculate centre
totY=0; %to calculate centre
[xp,yp]=find(I==150); % look for the 255 & 150 (sparse)
[xc,yc]=find(I==255); % look for the 255 & 150 (sparse)
x=[xc;xp];
y=[yc;yp];

num_pixels=length(x);
centre(1)=sum(x)/num_pixels;
centre(2)=sum(y)/num_pixels;

Xout=[x y]';
%centre to calculate scale
Xaux=[x-centre(1) y-centre(2)]'; %centre
Points=[xc yc]'; % Points=[xc-centre(1) yc-centre(2)]'; %centre

% calculate scale
dist=sum(sqrt(diag(Xaux'*Xaux)));
scale = MeanRefDist*num_pixels/dist;

% % scale calculated, now scale
% Xout=Xout*scale;
% Points=Points*scale;

%-------------------------------------------
%   End of "Image2Matrix" function - FER
%-------------------------------------------

%-------------------------------------------
%   Begining of "DrawLine" function - FER
%-------------------------------------------
function Iout=DrawLine(p1,p2,Iin)

Iout=Iin;
eq=DoLine2Points(p1,p2);

%%this gives better results, but slower:
for i = min(p1(1),p2(1)):0.01:max(p1(1),p2(1))
   for j = min(p1(2),p2(2)):0.01:max(p1(2),p2(2))
      if ( round(eq(1)*i+eq(2)*j+eq(3)) == 0 )

         % % this is faster
         % for i = min(p1(1),p2(1)):max(p1(1),p2(1))
         %    for j = min(p1(2),p2(2)):max(p1(2),p2(2))
         %       if ( abs(eq(1)*i+eq(2)*j+eq(3)) < 20 )

         Iout(1,round(i),round(j))=150; %FER - CHANGE - size of image/2
      end
   end
end
% draw points - for viewing
Iout(1,round(p1(1)),round(p1(2)))=255;
Iout(1,round(p2(1)),round(p2(2)))=255;

function Iout=DrawPoints(X,Iin,intensity)
Iout=Iin;
sz=size(X);
for i=1:sz(2)
   % draw points - for viewing
   Iout(round(X(1,i)),round(X(2,i)))=intensity;
   Iout(round(X(1,i)),round(X(2,i)))=intensity;
end

%---------------------------------------
%   End of "DrawLine" function - FER
%---------------------------------------


%-----------------------------------------------------
%   Begining of CENTRING AND SCALING FUNCTIONS
```

```
%---------------------------------------------------
function [Xout,centre]=Centre(Xin)
szx=size(Xin);
n=szx(2);
d = 0;
totX = 0;
totY = 0;
for j = 1:n
   d = d + 1;
   totX = totX + Xin(1,j);
   totY = totY + Xin(2,j);
end
centre(1) = totX/d; % FER -   d=n?
centre(2) = totY/d;

for j = 1:n
   Xout(:,j) = Xin(:,j) - centre';
end

function [Xout, Centers] = CentreThis(Xin)
szx = size(Xin);
m = szx(3);
for i = 1:m
   [Xout(:,:,i),Centers(i,:)]=Centre(Xin(:,:,i));
end

function centre=UpdateCentre(Xin)
szx=size(Xin);
n=szx(2);
d = 0;
totX = 0;
totY = 0;
for j = 1:n
   d = d + 1;
   totX = totX + Xin(1,j);
   totY = totY + Xin(2,j);
end
centre(1) = totX/d; % FER -   d=n?
centre(2) = totY/d;

function scale=UpdateScale(Xin,MeanRefDist)
szx = size(Xin);
n = szx(2);
r = 0;
dist = 0;
for i = 1:n
   dist = dist + sqrt((Xin(1,i)*Xin(1,i)) + (Xin(2,i)*Xin(2,i)));
end
if (dist==0)
   fprintf('\nERROR: distance 0!?\n');
end
scale = MeanRefDist*n/dist;

function [Xout,scale]=Scale(Xin,MeanRefDist)
szx = size(Xin);
n = szx(2);
r = 0;
dist = 0;
for i = 1:n
   dist = dist + sqrt((Xin(1,i)*Xin(1,i)) + (Xin(2,i)*Xin(2,i)));
end
scale = MeanRefDist*n/dist;
for i = 1:n
   Xout(1,i) = scale*Xin(1,i);
   Xout(2,i) = scale*Xin(2,i);
end

function [Xout, scales,MeanRefDist] = ReScale(Xin)
Xout = Xin;
szx = size(Xin);
n = szx(2);
m = szx(3);
RefDist = 0;
for i = 1:n
   RefDist = RefDist + sqrt((Xin(1,i,1)*Xin(1,i,1)) + (Xin(2,i,1)*Xin(2,i,1)));
end
```

```
MeanRefDist=RefDist/n;
scales(1) = 1;
for j = 2:m
   [Xout(:,:,j),scales(j)]=Scale(Xin(:,:,j),MeanRefDist);
end

function Xout=UnCentre(Xin,centre)
Xout=Xin;
Xout(1,:) = Xin(1,:) + centre(1);
Xout(2,:) = Xin(2,:) + centre(2);

function Xout=UnScale(Xin, scale)
Xout=Xin;
Xout(1,:) = Xin(1,:)/scale;
Xout(2,:) = Xin(2,:)/scale;

%---------------------------------------------------
%   End of CENTRING AND SCALING FUNCTIONS
%--------------------------------------------------

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    2:1    AFFINE    FUNCTIONS    %%%%%%%%%%%%%%%%%
CATT2Affine, getNewXAffine, GetNewab
function [a,b]=CATT2Affine(T)
aux=T(1)+T(4);
bux=T(5)+T(8);
cux=T(2)+T(3);
dux=T(7)+T(6);
a(1)=T(12)*bux - T(11)*dux;
a(2)=T(11)*bux - T(12)*dux;
a(3)=T(10)*bux - T(9)*dux;
a(4)=T(9)*bux - T(10)*dux;
b(1)=T(11)*cux - T(12)*aux;
b(2)=T(12)*cux - T(11)*aux;
b(3)=T(9)*cux - T(10)*aux;
b(4)=T(10)*cux - T(9)*aux;

function X=GetNewXAffine(basis1,basis2,a,b)
% szb=size(basis1); delete
% num_points=szb(2); delete
x1=[a(1) a(2)]*basis1+[a(3) a(4)]*basis2;
x2=[b(1) b(2)]*basis1+[b(3) b(4)]*basis2;
X=[x1;x2];

function [a,b]=GetNewab(X1,X2,X)
Y=[X1;X2];
AB=X*pinv(Y);
a=AB(1,:);
b=AB(2,:);
```

Grey level approach

Main loop

```
function [error,iterations] = Greylevels_fixed_without_middle()
tol=0.1; %tolerance of convergence (stopping rule)
MAX_NUM_ITER=500;

% DEFINING THE NOSE POINTS CONNECTIONS (in adjacent order -> for bisectors calculations)
connections=zeros(6,4); %6 points -- 4 possible connections
connections(1,1)=5;connections(1,2)=2;connections(1,3)=6;
connections(2,1)=1;connections(2,2)=3;
connections(3,1)=2;connections(3,2)=6;connections(3,3)=4;connections(3,4)=5;
connections(4,1)=5;connections(4,2)=3;connections(4,3)=6;
connections(5,1)=1;connections(5,2)=3;connections(5,3)=4;
connections(6,1)=1;connections(6,2)=3;connections(6,3)=4;

    W = xlsread('Landmark Points for processing.xls');
    sW = size(W);
    k = sW(1)/2;  % The number of images in the set
    n = sW(2);   % The number of landmark points in each image

    index = 1;
    for i = 1:k
        Xin1(1,:,i) = W(index,:); % FER - index could be 2*i -1
        index = index + 1;
        Xin1(2,:,i) = W(index,:); % FER - index could be 2*i
        index = index + 1;
    end

    clear W;
     NosePoints = GetStablePoints(Xin1); %FER - maybe we need to do this
    [NosePoints, Centers] = CentreThis(NosePoints);
    [NosePoints, scales,MeanRefDist] = ReScale(NosePoints);
    clear Xin1;
    szx = size(NosePoints);
    m = szx(3);

%---------------------------------- DRAWING - READ IMAGE AND COMPUTE CENTRES AND SCALE, just as a guess
target_image=6;
I=imread('images/0.bmp');
%guess centre and scale :   frontal view
centre=Centers(6,:); scale=scales(6);

target=NosePoints(:,:,target_image); %for error supervising purposes

FV=NosePoints(:,:,6);

szfv=size(FV);
num_corners=szfv(2);

global NUM_MIDDLE_POINTS; NUM_MIDDLE_POINTS=2; %number of middle points between pair of corner points
global SIZE_PROFILE; SIZE_PROFILE=4; %number of pixels in one wing of the profile (excluding the central one)

FV_old=FV+50; %just to start the first time

%centre to image space and then uncentre back with the guessed centre, just to show the initial error correctly
iterations=1;
Y=UnScale(FV,scales(target_image));
Y=UnCentre(Y,Centers(target_image,:));
Y=UnCentre(Y,-centre);
Y=UnScale(Y,1/scale);
errors(iterations)=norm(Y-target,'fro')/sqrt(num_corners);

% Option 1: Add middle points here (once, and then work with them). Stopping rule may take middle points into account or not
% look_up is a table indicating the corner points involved in each CONNECTION (not actual middle points!)
% [FV,look_up]=AddMiddlePoints(FV,connections);
% [basis1,look_up]=AddMiddlePoints(NosePoints(:,:,1),connections);
% [basis2,look_up]=AddMiddlePoints(NosePoints(:,:,m),connections);
basis1=NosePoints(:,:,1);
basis2=NosePoints(:,:,m);
look_up=1; %just to ignore it
% Basis views profiles
szi=size(I);
```

```
% here the basis profiles are built. profiles1wing2, for example, means profiles of basis view 1, wing corresponding to the second
segment (defined by segments1)
%the profilesXwingX are structured as follows: [point,bisector,profile], where the profile is of length SIZE_PROFILE + 1 (common
pixel included on both wings, and always first)
%the segmentsX are structured as follows: [point, bisector, (v_wing1,w_wing1,v_wing2,w_wing2)], where v,w are vectors centred
at the common pixel and ending where the profile should end
[profiles1wing1,profiles1wing2,segments1,profiles2wing1,profiles2wing2,segments2]=BuildBasisProfiles(basis1,basis2,scales,Cent
ers,FV,look_up,connections,m,szi);
T=GetNewT(basis1,basis2,FV); %intitialize T (to join profile segments and calculate weights)
% Structure of FV [corner points, middle points between corner #1 and #2, middle points bt #1 #5, and so on] the exact order can
be calculated using look_up
%%%%%%%%%%%%%%%%%%%% MAIN PROGRAM LOOP
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while ((StoppingRule(FV_old(:,[1:num_corners]),FV(:,[1:num_corners]),tol) & (iterations < MAX_NUM_ITER)) | (iterations==1))
   FV_i=UnScale(FV,scale);   %taking FV to image space of the target image (guessed)
   FV_i=UnCentre(FV_i,centre); %taking FV to image space of the target image (guessed)

points_found=FindPoints(FV_i,I,connections,look_up,profiles1wing1,profiles1wing2,profiles2wing1,profiles2wing2,segments1,seg
ments2,T); %Greylevels mode
   centre=UpdateCentre(points_found(:,[1:num_corners]));%updating centre with the corner points only, !not centering
   points_found=UnCentre(points_found,-centre); %centering ALL the points
   if ( (points_found(:,1)==[0 0]') & (points_found(:,2)==[0 0]') ) %points found are [0 0]
      fprintf('ERROR: points found are [0 0], unknown reason. EXITING\n');
      break;
   end
   scale=UpdateScale(points_found(:,[1:num_corners]),MeanRefDist);%updating scale with the corner points only, !not scaling
   points_found=UnScale(points_found,1/scale); %scaling
   % CATT
   T=GetNewT(basis1,basis2,points_found);
   FV_old=FV;
   %FV=GetNewXAffine(basis1,basis2,a,b);
   FV=GetNewX(basis1,basis2,T);

   iterations=iterations+1;

   % % view FV_old FV - using centres and scales from '6', for example
   % if option 1, DOnt take into account middle points! (remove them temporarily)
   Y=UnScale(FV(:,[1:num_corners]),scale);
   Y=UnCentre(Y,centre);
   S=DrawPoints(Y,S,ceil((100+iterations)/((100+450)/255)));%substract 200 to have the image "zoomed" in the area of interest

   errors(iterations)=norm(FV(:,[1:num_corners])-target,'fro')/sqrt(num_corners);
end
```

Auxiliary functions

Note that the functions that are common to the geometrical part are not listed again.

```
%*************************************************************************************************
%  FindQuad - Bisectors (DoBisector), Perpendicular, Intersections
(IntersectionsImage,CalculateIntersection,DoLine2Points,DoLine)
%*************************************************************************************************

%NOTE: if(errors(i)==999), do not take point into account (bisector didn't exist)
function best=GetBestMatch(point,fits,errors)
%OPTION 1 - The lowest error
[Y,I]=min(errors);
best=fits(:,I);
% OPTION 2 - closest - not implemented yet
% OPTION 3 - weighted interpolation  - not implemented yet

% this function does the image search for the greylevel case
function  found=FindPoints(X,I,con,look_up,profiles1wing1,profiles1wing2,profiles2wing1,profiles2wing2,segments1,segments2,T);
%Greylevels mode
global SIZE_PROFILE;
found=X; %initialization
[a,b]=CATT2Affine(T);
[w1,w2]=GetWeights(a,b);
segments=JointSegments(segments1,segments2,T); %calculate weighted profile segments
wing1=JointProfiles(profiles1wing1,profiles2wing1,w1,w2); % calculate weighted profiles
wing2=JointProfiles(profiles1wing2,profiles2wing2,w1,w2); % calculate weighted profiles
szs=size(segments);
num_points=szs(1);
```

```
num_bis=szs(2);
for i=1:num_points
   point=X(:,i);
   errors=zeros(1,num_bis)+999; %initialisation - fit error for each bisector
   best_fits=zeros(2,num_bis); %initialization - best point that fit for each bisector
   for j=1:num_bis
      error1=999; error2=999; % initialization - fit error of each of the wings
      if ((segments(i,j,1)==0) & (segments(i,j,2)==0))%no more bisectors for this point
         break;
      end
      %wing1
      for k=1:SIZE_PROFILE*3 %limit = 3xSIZE_PROFILE (one wing)
         v=squeeze(segments(i,j,[1:2])); %search vector
         %normalise... defining search step
         d=sqrt(v(1)^2+v(2)^2);
         vu=v*sqrt(2)/d; %search step is sqrt(2) (asure we advance one pixel)
         %match profiles(i,j,:) with image 'profile' (offseted)
         profile=GetProfile(I,point+vu*k,v);
         error=norm(double(profile)-squeeze(wing1(i,j,:))','fro')/sqrt(SIZE_PROFILE+1);
         if (error<error1)
            error1=error;
            best_fit1=point+vu*k;
         end
         %we must search in the other direction too
         profile=GetProfile(I,point-vu*k,v);
         error=norm(double(profile)-squeeze(wing1(i,j,:))','fro')/sqrt(SIZE_PROFILE+1);
         if (error<error1)
            error1=error;
            best_fit1=point-vu*k;
         end
      end %for search wing1
      %wing2
      for k=1:SIZE_PROFILE*3 %limit = 3xSIZE_PROFILE (one wing)
         v=squeeze(segments(i,j,[3:4])); %search vector
         %normalise... defining search step
         d=sqrt(v(1)^2+v(2)^2);
         vu=v*sqrt(2)/d; %search step is sqrt(2) (asure we advance one pixel)
         %match profiles(i,j,:) with image 'profile' (offseted)
         profile=GetProfile(I,point+vu*k,v);
         error=norm(double(profile)-squeeze(wing2(i,j,:))','fro')/sqrt(SIZE_PROFILE+1);
         if (error<error2)
            error2=error;
            best_fit2=point+vu*k;
         end
         %we must search in the other direction too
         profile=GetProfile(I,point-vu*k,v);
         error=norm(double(profile)-squeeze(wing2(i,j,:))','fro')/sqrt(SIZE_PROFILE+1);
         if (error<error2)
            error2=error;
            best_fit2=point-vu*k;
         end
      end %for search wing1
      %Join both points found together using error1 error2
      best_fits(:,j)=(error1*best_fit2 + error2*best_fit1)/(error1+error2); %weighted average
      errors(j)=2*error1*error2/(error1+error2); %weighted average
   end %bisctors
   %chose the point found amongst found in each bisector
   found(:,i)=GetBestMatch(point,best_fits,errors);
end %points

function centre=UpdateCentre(Xin)
szx=size(Xin);
n=szx(2);
d = 0;
totX = 0;
totY = 0;
for j = 1:n
   d = d + 1;
   totX = totX + Xin(1,j);
   totY = totY + Xin(2,j);
end
centre(1) = totX/d; % FER -   d=n?
centre(2) = totY/d;

function scale=UpdateScale(Xin,MeanRefDist)
szx = size(Xin);
```

```
n = szx(2);
r = 0;
dist = 0;
for i = 1:n
   dist = dist + sqrt((Xin(1,i)*Xin(1,i)) + (Xin(2,i)*Xin(2,i)));
end
if (dist==0)
   fprintf('\nERROR: distance 0!?\n');
end
scale = MeanRefDist*n/dist;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GREYLEVEL FUNCTIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%compute transformation for one triangle - return as one dimensional array
function T=ComputeTransformation(Origin,Target)
%put origin in homogeneous
Origin(3,:)=1;
Inv=pinv(Origin);
if (Inv==Inf) %was singular - I dont know if pinv check this itself
   Inv=pinv(Origin);
end
T=reshape(Target*Inv,1,6);

%compute affine transformations of all triangles of one view - return as a one dimensional array
function trans=ComputeTransformations(FV,basis,box,triangles);
szt=size(triangles);
num_triangles=szt(1);
trans=zeros(num_triangles,6);
for i=1:num_triangles
   tri=triangles(i,:); %get points of triangle (indices)
   a=tri(1);b=tri(2);c=tri(3);
   if (a<0) %point of the box
      A=box(:,abs(a));
      A_basis=box(:,abs(a));
   else %point of the nose
      A=FV(:,a);
      A_basis=basis(:,a);
   end
   if (b<0) %point of the box
      B=box(:,abs(b));
      B_basis=box(:,abs(b));
   else %point of the nose
      B=FV(:,b);
      B_basis=basis(:,b);
   end
   if (c<0) %point of the box
      C=box(:,abs(c));
      C_basis=box(:,abs(c));
   else %point of the nose
      C=FV(:,c);
      C_basis=basis(:,c);
   end
   trans(i,:)=ComputeTransformation([A B C],[A_basis B_basis C_basis]);
end

function inside=PointInsideTriangle(point,A,B,C)
ab=cross([(B-A)' 0],[(point-A)' 0]);
z_ab=ab(3);
bc=cross([(C-B)' 0],[(point-B)' 0]);
z_bc=bc(3);
ca=cross([(A-C)' 0],[(point-C)' 0]);
z_ca=ca(3);
if (  ((z_ab<=0)&(z_bc<=0)&(z_ca<=0)) | ((z_ab>=0)&(z_bc>=0)&(z_ca>=0))  ) %checking if all positive just in case the points
are not ordered clockwise
   inside=1;
else
   inside=0;
end

% checks for point being inside each of the triangels in 'possible' (only indices), triangles is the list of triangles to look for the points
involved (only indices of points); FV and boox to look for the acutal coordinates
function t=WhichTriangle(point,possible,triangles,FV,box);
num_possible=length(possible);
```

```
t=0;
for i=1:num_possible
   tri=triangles(possible(i),:); %get points of triangle (indices)
   a=tri(1);b=tri(2);c=tri(3);
   if (a<0) %point of the box
      A=box(:,abs(a));
   else %point of the nose
      A=FV(:,a);
   end
   if (b<0) %point of the box
      B=box(:,abs(b));
   else %point of the nose
      B=FV(:,b);
   end
   if (c<0) %point of the box
      C=box(:,abs(c));
   else %point of the nose
      C=FV(:,c);
   end
   if PointInsideTriangle(point,A,B,C) %found
      t=possible(i);
      break;
   end
end
if (t==0) %not inside any triangle - could be because end point gets out of triangle (in that case, use the closest)
   fprintf('WARNING: In WhichTriangle, point not inside expected triangles, getting closest instead\n');
   % first, find the triangle in wich the point really is
   all=[1 2 3 4 5 6 7 8 9 10 11 12]; %all the triangles
   rest=setxor(all,possible); %all except those already tried
   num_rest=length(rest);
   for i=1:num_rest
      tri=triangles(rest(i),:); %get points of triangle (indices)
      a=tri(1);b=tri(2);c=tri(3);
      if (a<0) %point of the box
         A=box(:,abs(a));
      else %point of the nose
         A=FV(:,a);
      end
      if (b<0) %point of the box
         B=box(:,abs(b));
      else %point of the nose
         B=FV(:,b);
      end
      if (c<0) %point of the box
         C=box(:,abs(c));
      else %point of the nose
         C=FV(:,c);
      end
      if PointInsideTriangle(point,A,B,C) %found
         t_aux=rest(i);
         break;
      end
   end %for rest
   if (t_aux==0)
      fprintf('ERROR: In WhichTriangle, point not inside ANY triangle!!!');
   end
   % point is inside 't_aux', but we want the closest among one of the possible. We suppose 't_aux' and the one we want are adjacent
   % if they are adjacent -> they share two vertex, no other triangle in 'possible' shares also two vertex (in our nose)
   for i=1:num_possible
      if (intersect(triangles(t_aux,:),triangles(possible(i),:)) == 2) % two common vertex
         t=i;
         break;
      end
   end
   if (t==0) % 't_aux' wasn't adjacent to any triangle in 'possible' -> return 't_aux'
      t=t_aux;
   end
end %t==0

%returns vector from basis to transformed end_point
function segment_vector=BasisEndVector(end_point,basis,trans,possible,triangles,FV,box)
t=WhichTriangle(end_point,possible,triangles,FV,box);
T=reshape(trans(t,:),2,3); %buiding transformation matrix
end_point(3)=1; %homogenous
basis_end=T*end_point; %affine transformation
segment_vector=basis_end-basis;
```

```
%Ben's asymetry - returns the two vectors going from each central landmark point to the segment end
function [segments1,segments2]=BasisSegments(FV,basis1,basis2,box,bis,trans1,trans2,look_up,look_up_triangles,triangles)
global SIZE_PROFILE;
% global NUM_MIDDLE_POINTS;
szb=size(bis);
num_points=szb(1);
num_bisectors=szb(2);
segments1=zeros(num_points,num_bisectors,4); %initialization - last component is 4 to contain both vector-segments
segments2=zeros(num_points,num_bisectors,4); %initialization - last component is 4 to contain both vector-segments
% szl=size(look_up);
% num_connections=szl(1);
% num_middle_points=NUM_MIDDLE_POINTS*num_connections;
% num_corners=num_points-num_middle_points;
for i=1:num_points
   %find possible triangles
%    if (i>num_corners) %middle point
%        con_num=ceil((i-num_corners)/NUM_MIDDLE_POINTS);
%        possible=intersect(look_up_triangles(look_up(con_num,1),:),look_up_triangles(look_up(con_num,2),:));
%    else
      possible=look_up_triangles(i,:);
%    end
   %remove zeros in 'possible'
   ind=find(possible==0);
   possible(ind)=[];
   %for each bisector get the two segment ends
   for j=1:num_bisectors
      if ((bis(i,j,1)==0) & (bis(i,j,2)==0))%no more bisectors for this point
         break;
      end
      v(1)=-bis(i,j,2);
      v(2)=bis(i,j,1);
      %normalise... again?
      d=sqrt(v(1)^2+v(2)^2);
      v=v/d;
      %one end of the segment - for both views
      end_point=FV(:,i)+v'*sqrt(2)*SIZE_PROFILE;
      segments1(i,j,1:2)=BasisEndVector(end_point,basis1(:,i),trans1,possible,triangles,FV,box);
      segments2(i,j,1:2)=BasisEndVector(end_point,basis2(:,i),trans2,possible,triangles,FV,box);
      %the other end of the segment - for both views
      end_point=FV(:,i)-v'*sqrt(2)*SIZE_PROFILE;
      segments1(i,j,3:4)=BasisEndVector(end_point,basis1(:,i),trans1,possible,triangles,FV,box);
      segments2(i,j,3:4)=BasisEndVector(end_point,basis2(:,i),trans2,possible,triangles,FV,box);
   end
end %for points

%gets the greylevels of one wing of a particular point
function profile=GetProfile(I,point,v)
global SIZE_PROFILE;
profile(1)=I(round(point(1)),round(point(2)));
for k=1:SIZE_PROFILE
   profile(k+1)=I(  round( point(1)+v(1)*k/SIZE_PROFILE ),round( point(2)+v(2)*k/SIZE_PROFILE )  );
end

%gets the greylevels of all the segments of all the points - both wings
function [wing1,wing2]=BasisProfiles(I,points,segments)
szs=size(segments);
num_points=szs(1);
num_seg=szs(2);
wing1=zeros(num_points,num_seg,5); %5 values counting the middle point
wing2=zeros(num_points,num_seg,5); %5 values counting the middle point
for i=1:num_points
   for j=1:num_seg
      if((segments(i,j,1)==0) & (segments(i,j,2)==0) & (segments(i,j,3)==0) & (segments(i,j,4)==0))  %no more segments for this
point
         break;
      end
      wing1(i,j,:)=GetProfile(I,points(:,i),segments(i,j,[1:2]));
      wing2(i,j,:)=GetProfile(I,points(:,i),segments(i,j,[3:4]));
   end
end

function
[profiles1wing1,profiles1wing2,segments1,profiles2wing1,profiles2wing2,segments2]=BuildBasisProfiles(basis1,basis2,scales,Cent
ers,FV,look_up,connections,m,szi)
```

```
basis1_uncentred=UnScale(basis1,scales(1));
basis1_uncentred=UnCentre(basis1_uncentred,Centers(1,:));
FV_uncentred=UnScale(FV,scales(6));
FV_uncentred=UnCentre(FV_uncentred,Centers(6,:));
basis2_uncentred=UnScale(basis2,scales(m));
basis2_uncentred=UnCentre(basis2_uncentred,Centers(m,:));
%defining outer limits (image box)
box=[0 0 szi(1) szi(1);0 szi(2) szi(2) 0];
% points belonging to each triangle (clockwise) - for all views - negative means outer limits (image box)
triangles=[1 -1 -2;1 -2 6;1 6 3;1 3 5; 1 5 -1;3 6 4;3 4 5;4 6 -3;4 -3 -4;4 -4 5;5 -4 -1;6 -2 -3];
% compute affine transformations to use for each triangle (both basis views)
trans1=ComputeTransformations(FV_uncentred,basis1_uncentred,box,triangles);
trans2=ComputeTransformations(FV_uncentred,basis2_uncentred,box,triangles);
% look up table to know which triangles are adjacent to which points (used to reduce number of possible triangles)
look_up_triangles=[1 2 3 4 5;3 4 0 0 0;3 4 6 7 0;6 7 8 9 10;4 5 7 10 11;2 3 6 8 12];
%bisectors for FV
bisectors_FV=Bisectors(FV_uncentred,connections); %own lines -> Bisectors2
% bisectors_FV=Perpendiculars(FV_uncentred,connections,look_up,bisectors_FV); %maybe dont need con
%get segments for both views
[segments1,segments2]=BasisSegments(FV_uncentred,basis1_uncentred,basis2_uncentred,box,bisectors_FV,trans1,trans2,look_up
,look_up_triangles,triangles);
%get profiles
BASIS1=imread('images/-25.bmp');
BASIS2=imread('images/25.bmp');
%each view has a profile wing for each segment-vector
[profiles1wing1,profiles1wing2]=BasisProfiles(BASIS1,basis1_uncentred,segments1);
[profiles2wing1,profiles2wing2]=BasisProfiles(BASIS2,basis2_uncentred,segments2);

function segments=JointSegments(segments1,segments2,T) %calculate weighted profile segments
segments=segments1; % initialization
szs=size(segments1);
num_points=szs(1);
num_bis=szs(2);
for i=1:num_points
   for j=1:num_bis
      if ((segments(i,j,1)==0) & (segments(i,j,2)==0))%no more bisectors for this point
         break;
      end
      segments_aux=GetNewX(reshape(segments1(i,j,:),2,2),reshape(segments2(i,j,:),2,2),T);
      segments(i,j,:)=reshape(segments_aux,1,4);
   end
end

function profiles=JointProfiles(profiles1,profiles2,w1,w2) % calculate weighted profiles
profiles=profiles1*w1+profiles2*w2;

function [w1,w2]=GetWeights(a,b)
d1=a(3)^2+a(4)^2+b(3)^2+b(4)^2;
d2=a(1)^2+a(2)^2+b(1)^2+b(2)^2;
w1=d2/(d1+d2);
w2=d1/(d1+d2);

%%%%
% END OF GREYLEVEL FUNCTIONS
%%%%%%%%%%%
```