FACIAL RECOGNITION FROM VIDEO SEQUENCES

Martyn W Booth

Computing (Industry)

Session (2006/2007)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from other sources may be considered as plagiarism.

(Signature of student).....

ABSTRACT

Face recognition has proven to be a difficult process to automate, even though humans are able to detect and identify faces with little or no effort. As a result of this, the research into automated face recognition has remained a focal point in the field of computer vision and artificial intelligence for over twenty five years. There have been many different attempts at solving the problem of automatic facial recognition (from static images and from video). In order to address this, a number of recent developments in the field of computer vision are examined. Some of these techniques are then applied to the design, implementation and test of a prototype facial recognition system utilising a live video feed. Furthermore, the system will use a number of video frames in order to attempt recognition. This is an attempt to improve recognition rates.

ACKNOWLEDEMENT

I would like to thank Professor David Hogg for his help, assistance and patience in helping me with some of the more complex areas of this project. Without his help, this report would be in an extremely poor state.

TABLE OF FIGURES

TABLE OF TABLES

TABLE OF EQUATIONS

Chapter 1 – REPORT INTRODUCTION

1.1 Introduction

The purpose of this chapter is to demonstrate the motivation for the study being undertaken. It will begin with a problem definition that will include how the problem has emerged followed by the scope of the study. The scope will include a section on the limitations and constraints of this project. The final part of this section will list the minimum requirements expected by the project and what exactly is expected to be achieved by undertaking this work.

Although the concept of recognizing someone from facial features is intuitive and automated in the human brain from an early age, facial recognition, as a biometric, makes human recognition a more automated, computerised process. What sets apart facial recognition from other biometrics is that it can be used for surveillance purposes. For example, public safety authorities want to locate certain individuals such as wanted criminals, suspected terrorists, and missing children. They also want to automate such areas as border checkpoints and automate access to secure areas. Facial recognition has obvious advantages in this area, even though commercial uses have already been made of this software; Newham (Meek, 2002) borough Council has installed 140 cameras on their high street and have linked these cameras to a complex facial recognition database that has resulted in a 'marked reduction in crime in the area.'

1.2 Problem Definition

Over recent years, Computer Vision has become a more interesting topic as its uses in practical scenarios becomes more apparent. Essentially, it is now more than just a theoretical exercise and instead offers real value if implemented correctly. This leap forward in possibility is due to the developments of technology and theory that now make it possible to implement Computer Vision systems effectively and efficiently.
There are numerous definitions for Computer Vision and Facial Recognition;

It has been suggested that Computer Vision is a broad field, which covers topics such as "image restoration, image enhancement, automated visual inspection, computer based image understanding of three dimensional scenes, and visual perception and cognition." (Vernon, 1991).

Whereas Computer Vision is different to Image processing because "image processing is the processing of an image, typically by a computer, to produce another image, while Computer Vision is about image acquisition, processing, classification, recognition, and, to be all embracing, decision making subsequent to recognition." (Low, 1991).

Facial Recognition is a subset area of Computer Vision and deals specifically with the processing of an image, the localisation of the face, the extraction of the features from the face and then the comparison of those features against some already held in some secondary storage medium to ascertain the likelihood of a match.

This report considers facial recognition in a very particular context. It considers identifying a particular person by storing attributes from the features on their faces and comparing these features to those extracted from a live video feed (in this context, the video source could be considered to be "pseudo-live" as it may not be possible to run the necessary algorithms in real time). This identification will be made based on locations of the eyes in particular and will utilise several frames of video to obtain a mean distance between features. This is to minimise the error detected between the various features in each individual frame and to increase the accuracy of the overall system.

A current problem in the area of Computer Vision is the successful implementation of a working facial recognition system. As mentioned above, this area has excellent commercial potential for areas such as border control and could replace key lock mechanisms. In this scenario, the door could automatically be opened as a person approaches it given that they have a face held on the database.

However, there are few facial recognition systems that work reliably and even fewer that are cost effective. Essentially, the field of facial recognition currently remains firmly in the realm of research. This will remain the case until certain problems have been overcome. This is not to say that considerable progress has not been made in recent years in some of the problem areas; a robust face detection system now exists that accurately locates a face in a complex background and in a variety of sizes (Viola & Jones, 2004). Further problems that are still under investigation exist and some of them are addressed in this study:

- Facial characteristics; Glasses, hair styles and partial facial occlusion can all affect the ability of the face detector to accurately localise the face. Some improvements have been made in this area and the Viola & Jones face detector (mentioned above) are now fairly difficult to fool.
- Facial Expressions; a person that is smiling or frowning may lead to incorrect measurements being taken from around the subject's mouth. This is due to the fact that many facial recognition systems use training data to 'train' a system to be able to recognise a particular face. During this training period, the faces used are normalised in a single position and are therefore susceptible to changes in facial expressions.
- Lighting conditions; Variation in lighting conditions can throw off a feature extraction algorithm because the 'templates' are essentially incorrect. It then becomes very difficult to locate features and a possible match may be missed.
- Face Position; for many legacy facial recognition systems, it was expected that the face would be offered to the face detector as a 'straight-on' representation of the face. Even now, variations on this are extremely difficult to deal with. Even though the Viola & Jones face detector has the ability to locate faces from a 'side-on' view it is difficult to do this in an automated fashion. From a technical perspective, the face may be partially occluded if not straight on and the features will appear slightly closer together (or further apart) even if the face is accurately detected but the pose is partially off-centre.

1.3 Study Scope

The study scope, in short, is to gain an understanding of the techniques used in current facial recognition systems and then use a number of those methods to create a basic facial recognition system prototype that can extract certain features from a face and use those to compare the 'live' face to 'test' faces stored in a database.

The techniques compared include:

- Elastic two-dimensional shape matching
- Bayesian learning approach
- Two-dimensional PCA
- Kernel-based Eigenfaces
- Support Vector Machines (SVMs)

The prototype system should reliably match faces in a database based on facial features under conditions that are not strictly controlled. For example; with people having different facial expressions, slight variations in pose and wildly varying lighting conditions. The main evaluation of the system will revolve around its ability to deal with the aforementioned conditions and still accurately locate the face and the necessary features as part of that detection.

For each of the areas being considered; face detection, 'individual' feature localisation and comparison methods. Only one technique from the literature survey will be utilised, however, the reasons why that technique has been decided upon will be clearly stated. The following assumptions are made about the subjects in the images:

- The subject's head will not rotate more than 10 degrees in any directions from a straight-on camera view for the duration of the video capture sequence. This constraint is due to limitations of the face detection algorithm.
- Lighting variations should be minimalised to 'normal' video noise levels rather than drastic changes for the duration of the video sequence.
- Facial expression should be changed only minimally and extremely slowly over the course of the video capture sequence.
- Background is assumed and handled by the face detector but multiple faces are not at this stage. (The solution itself will support detection and processing of multiple faces but due to real-time video processing concerns, it is assumed that a single face is present. This also helps the application to 'track' the face over multiple frames because no separate facial tracking software is being used.)

1.4 Study Objectives

The primary objectives of this study are to produce a comprehensive report, specifically aimed at people of mathematically and computing backgrounds as is required by the BCS (British Computing Society.) This includes gaining a good understanding of computer vision, its application in this area and sound reasoning for the methods chosen to be used in the prototype.

As this study deals with some complex subject matter and programming paradigms, a second objective is to gain a much better understanding of the high level language used to implement the prototype system (Java or C#).

The prototype system itself is a key objective of this study. It will accurately locate faces and facial features from a video sequence and will use these to attempt to classify the face being viewed. In addition, a graphical user interface will be provided to add new test data to the database and show the results of the video sequence. Finally, the system will use a 'fusion' of video frames in order to obtain a 'mean' face of the person being studied in an attempt to reduce the impact of erroneous feature detection and improve overall effectiveness of the application.

Finally, the system will be evaluated to ascertain its success or failure and then suggest improvements based on the findings of this appraisal.

1.5 Project Plan

Date	*Milestone*
October – December 2007	• Background Research conducted • Programming mock-ups conducted for suitably
January 2007	• Implement feature localisation in Matlab and in Java • Establish user requirements
January – Early March 2007	• Full Requirements gathering and specification • *Further feature extraction research*
March – April 2007	• Completion of prototype system • *Graphical interface design work*
April 2007	• *Changes to video handling to take advantage of threaded processing* • Full project report write-up

Table 1.5-1- Project Plan Table

1.5.1 Project Plan Revisions (Shown in italics above Table 1.5-1)

As the project progressed, it became necessary to make revisions to the schedule as unanticipated work presented itself and as deadlines became tighter. The main revisions are detailed; more research was conducted late on for other feature extraction techniques as it became apparent that the original ones (those summarised in the mid-project report) were inadequate in real-time. It was also decided that more work should be focussed on the design of a usable graphical interface as this was essentially the only part of the system the user would see.

Finally, the completion of development presented problems with the flow of execution. Considerable effort was put into correcting this and is therefore cited as a change from the planned schedule.

Chapter 2 – FACIAL RECOGNITION

2.1 About Facial Recognition

Over the past few decades there has been a research focus in the area of computerised identification of the human face. Rapid and prolonged technological advances over a more recent period result in the ability to move this area of research from theoretical, into practical. Technological advances include; rapid advancements to the speed and efficiency of computer hardware (processors, memory and even webcams etc.), advanced support for video and image processing in high-level languages, the realisation of robust real-time face detection software (Viola & Jones, 2004) and increased focus on feature localisation techniques such as the Bayesian approach to eye localisation (Everingham & Zisserman, 2006).

Specific uses of automated facial recognition and where it would be particularly useful are as follows:
The identification of people as they approach, and wish to enter, buildings that have restricted access; currently, security guards are responsible for the labour intensive task of checking people against identification cards and confirming that they are indeed entitled to the access that they seek. This requires a large investment of staff but because humans are susceptible to error, it is likely that a large number of mistakes will be made over the course of the checks. These errors are the result of human tiredness, fatigue and general lapses in concentration. An automated face recognition system could check access to building (or even specific rooms) twenty-four hours a day, seven days a week and could alert a single guard if something suspicious occurred. Additionally, the system would be able to keep detailed logs of person movement with little extra effort. Finally, the system is not biased and does not become familiar with subjects, as happens in human-based systems, so the rules do not become relaxed.

Many more tedious and difficult jobs could be removed by using face recognition systems; a key area for this in the current political climate is terrorist identification and alert. The same principal applies to the search for missing children, this could be handled by face recognition software as an alternative to the classic American ‘milk carton’ approach. These applications and many others are proposed in a recent report to the state of Virginia crime commission (Horn, Gatune, Thomas, & Woodward, 2003) along with several constraints. The main issues revolve around the difficulty of recognising faces in real-time especially when comparing to an ever increasing database of reference faces. The notion seems to imply that processing of the faces may need to be handled ‘off line’ rather than in real time. However, this paper does fail to realise that the most recent improvements in computer hardware technology has the potential to process the images in near real time speeds. Running on ordinary computers (Pentium 3 1 GHz) the face detector software can accurately locate a face at fifteen frames per second. Despite this, many of the problems

present at the end of the last decade have gone some way to be being alleviated; face detection in real-time, fast objection matching and image processing.

The reduction in cost of computer equipment mentioned above could render the password obsolete. A simple webcam could instead be used to properly identify a person when sitting in front of their computer. This also acts as a step to reduce the increasing problem of identity theft, as it becomes much more difficult to steal a person's face rather than just their name.

2.1.1 Difficulties with current face recognition systems

As mentioned in Chapter 1, the problems with facial recognition are slowly being eroded away. Recent advancements in many areas have made facial detection and recognition a reality and thus much work has begun in this field.

Many obstacles still exist to creating a working, reliable facial recognition system and are summarised as follows (Torres, 2004):
Pose variation (facial expression and facial position), illumination conditions, scale variability, images taken years apart, glasses, moustaches, beards, low quality image acquisition, partially occluded faces etc. A separate set of problems revolves around the fact that facial images (once acquired by the face detection system) are remarkably similar; they all have a mouth, a pair of eyes, a pair of ears and a nose etc.

The problem of image condition is being dealt with slowly however, the face detector mentioned previously (Viola & Jones, 2004) is robust under many different conditions. An existing problem with face detection was the 'noise' (areas around the face that may throw off the detector,) this has been almost alleviated by the Viola and Jones face detector. It is now possible to locate a number of faces against complex backgrounds. In Figure 2.1-1 below, the faces are accurately localised, with a single false positive result. All of the below faces have regular facial characteristics (no beards, moustaches etc.) and this may have influenced such positive results.

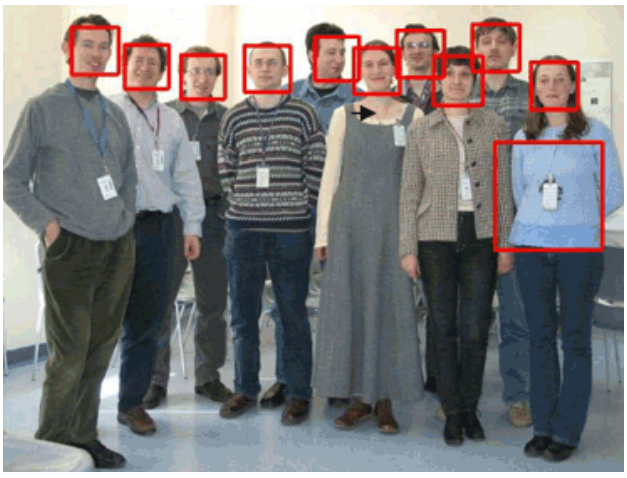

Figure 2.1-1 – Face detection using Viola & Jones face detector (Bornet, 2005)

Another problem linked to the condition of the acquired image is the lighting of that image. Changes in the lighting around the person being identified can throw the matching process off. This problem has been to locate the facial features. This is especially true when using a learning algorithm in order to do the comparison because the 'mean feature' will have a particular lighting variation.

Facial expression is an existing problem in recognition systems. Variations in facial recognition; a face frowning as opposed to a face smiling for example essentially changes the proportions of the face and can cause the recognition system to make an incorrect match:

Figure 2.1-2 - Changes in facial expressions (Dyer, 2004)

Significant advances have been made to remove this problem; a system is proposed to accurately map a face even though the expression may be in a state of constant change (Kittler & Nixon, 2003), their system builds a three dimensional model of the face and utilises Principal Component Analysis (PCA) to allow the face to be measured even though its basic dimensions may have changed. This work is based on further studies into using PCA to recognise faces which is invariant to facial expressions (Turk & Pentland, 1991). Eigenfaces refer to this PCA analysis and are the collection of Eigenvectors measuring the distance between various facial components. They are now considered to be a defacto standard in facial recognition and have been used successfully in the aforementioned cases to track faces in an expression invariant manner.

In summary, the following problems have existed for computer vision and specifically facial recognition since it came into research focus (Torres, 2004):

- Pose variation (facial expression and facial position)
- Illumination conditions,
- Scale variability,
- Images taken years apart,
- Glasses,
- Moustaches,
- Beards,

- Low quality image acquisition,
- Partially occluded faces
- Facial images are remarkably similar; they all have a mouth, a pair of eyes, a pair of ears and a nose etc.

Many of these problems have slowly been addressed over recent years; Lighting, Scale, Pose, Low quality images and partial occlusion are now dealt with effectively by a number of face detectors and PCA-based feature extraction algorithms.

2.2 Human interpretation of Faces

Humans can identify a large number of faces with ease. In contrast, computer systems find it extremely difficult to process a face containing an image and accurately match it to another face.

It is reasonable to assume that humans do not simply use pattern matching techniques in order to recognise a person's face but that they use their innate instincts in order to do it successfully. Humans can even recognise people long after they last saw them because of other non-visual queues, such as gestures or the way they talk. It is also not unreasonable to assume that if a person only had a short glimpse of another person (similar to facial recognition systems) they would not be able to differentiate when presented with thousands of subjects. This is one of the reasons that make automated facial recognition so hard, especially given that a system must usually assume that the face is unfamiliar and look for features which could indicate that they are the same person. However, given time to learn the face, humans can know up to seven hundred faces on a personal level and even potentially be able to recognise thousands of others (Samal & Iyengar, 1992); computers too should have this ability but a database that large invariably incurs a performance problem.

Faces in human memory can be thought of as being viewed in multidimensional space, the size of this space increases through increased exposure to the faces in the space (Pezdek, Blandon-Gitlin, & Moore, 2003).

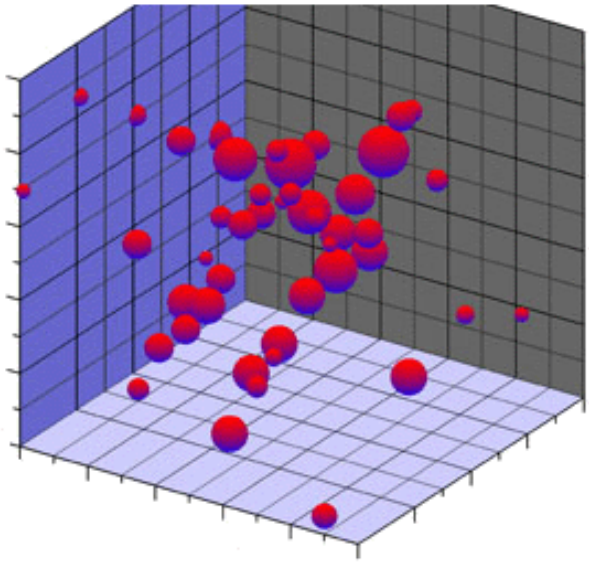

Figure 2.2-1 - Representation of the child face space. Inferred from (Pezdek, Blandon-Gitlin, & Moore, 2003)

Figure 2.2-1 above shows how children (and adults) perceive facial images. Faces are essentially split between distinctive and common. Common faces lie close to the centre of the space where there is less variation and therefore they are more difficult to distinguish from one another. More distinctive faces lie closer to the boundaries of the area and are therefore easier to differentiate. Given this information and the fact that, by definition, common faces are more frequent than distinctive ones, it is perhaps understandable that computers struggle to differentiate between the faces because they are so similar.

The following chapter examines some of the more prominent methods of face acquisition and feature extraction.

2.3 Conclusion

Automated facial recognition systems would have many uses in the real world amongst public and private sectors users; law enforcement and building security are a few of the more obvious ones. In order to realise the use of facial recognition as a reliable tool, a number of problems need to be overcome and researchers in this area are working hard to overcome the issues (a number of which, as mentioned, have already been over come to some extent.)

In addition to the technical advances being made in this field, it is important that the legalities are not overlooked and that suitable decisions are made about the objects that these systems guard. It is important that no facial recognition system impede any individuals' rights with regard to data protection and personal liberties. Extreme caution should be taken when deploying systems and when classifying reference individuals. In addition, a key threshold should be established to ensure that the system has a higher false positive rate or false negative rate depending on the type of application. For instance, access to a high security facility by means of facial identification

should have a higher false negative rate to prevent erroneous access. If a legitimate subject were refused entry, this could be easily overcome with less severe consequences than the alternative.

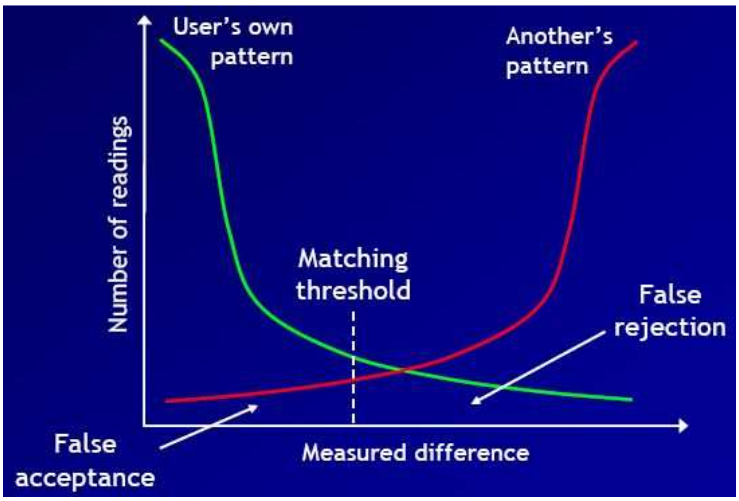

Figure 2.3-1 - False acceptance, false rejection rate example (Efford, 2006)

Finally, when designing a system to achieve facial recognition, serious thought should be attributed to the difficulties identified above (Torres, 2004) and methods chosen which are invariant, or at least partially invariant, to as many of them as possible.

Chapter 3 – LITERATURE SURVEY ON FACIAL RECOGNITION TECHNIQUES

3.1 Introduction

This chapter is designed to give a detailed overview of the different methods currently being used for facial acquisition and feature extraction that can be used for automated facial recognition systems. The chapter will give a brief overview of each method, how it works and the disadvantages and advantages that it offers. The discussion will cover popular methods and recent important research focuses in the field that may be of interest to future systems. The chapter will conclude with a comparison of some of the various methods.

3.2 Introduction to automated Face Recognition systems

Research into the field of computing is increasing as more efficient and effective methods become available to the public domain. This section looks at some of these methods and how successful they have been.

The actual process of recognising a face can be split into a number of different processes; the detection of a face amongst a cluttered background, the extraction of a number of features from the face area collected above and finally the comparison of those features against a number of reference faces held in secondary storage in order to produce a match. In this report, the location of the face in the video frame is not predetermined and must be established by the program itself using one of the mentioned methods. The

process following this could be to take the face as a single component and attempt to match it to a reference face in that way, however the failings of this method are well documented (Heisele, 2006). Therefore, the process to be used instead is to extract an as yet undetermined set of features and then use those for comparison.

A common set of steps is usually undertaken by most systems in order to attempt to recognise a face (Samal & Iyengar, 1992):

1. Determine a set of features to represent the subject's face
2. Transfer the features identified in step 1 to a database or other secondary storage
3. Determine a set of features (the same features as step 1) for a previously unseen face
4. Use an algorithm to match the features in steps 1 and 3 held in step 2 and system memory respectively

Some background information is needed about the steps above and is presented in sections 3.7.1 through to 3.7.5.

3.3 Feature Extraction

Although many traditional facial identification systems work by matching whole faces using templates or some other geometrics, it has become much more common for systems to attempt to locate any given number of facial features and compare the measurements of those features in relation to one another. Once located, performing the necessary functions on the features becomes relatively straight forward. Therefore, most design effort should be placed on the effective localisation of the features.

The parts of the face vary in importance with regard to how significant they are to the distinctiveness of any particular human face. In no particular order, the following parts of the human face are often described as being most significant to the identification of that face (Zhao, Chellappa, Rosenfeld, & Phillips, 2007):

- Hair
- Face Outline
- Eyes
- Mouth

Extracting the aforementioned features efficiently and effectively is crucial to creating a reliable facial recognition system. The nose plays a very insignificant role in the recognition of a face and as its shape does not lend itself to pattern matching, so should typically be ignored. As a side point; the upper portion of the face is more important than the lower portion and this fact could be utilised to increase performance (or at least reassign processing from lower to upper accordingly.) Studies have also shown that the attractiveness of faces affects their ability to be recognised easily. However this, at least at our current technology level, is impossible to translate into a computer algorithm.

Because of the difficulty of extracting features from a face it is important to focus efforts on those features than can be gathered most easily and which are most invariant to the conditions mentioned in section 2.1.1. The following prerequisites are inferred from a recent computer vision conference for the effective use of facial features (Jiao, Gao, Chen, Cui, & Shan, 2002):

- Estimating the area where the features are most likely to occur must be as simple as possible
- Changes in facial expression should have a small effect on the overall location of the feature
- The information posed by the feature should be crucial to the overall recognition of the face
- Light conditions should have as small an effect on the feature as possible

It is critical that the features used are quick to locate but that they also conform to the above standards where possible. If the feature is sensitive to some of the problems cited in section 2.1.1 then the feature may not be useful in an unnormalised environment. This calls into question the viability of an expression variant mouth in the above stated scenario.

The main methods used for the duration of this report are the Bayesian learning approaches and edge detection approaches. These are discussed in the following paragraphs.

The Bayesian learning approach refers, in essence, to an algorithm that uses training data to create a 'template' which can then be compared to every part of the test image to find a match. The match should be the centre of the patch that most closely resembles the feature being searched for.

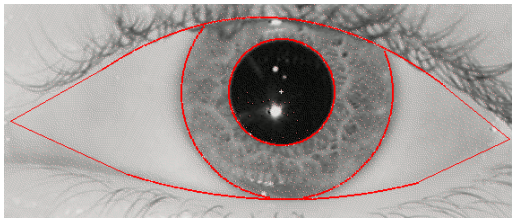

Figure 3.3-1 – Average eye image (acting as template)

Figure 3.3-2 – Eyes localised on test image (Zhao & Grigat, 2006)

The above images show the uses of template based matching that can be used in facial recognition systems. The red line in the image on the left, above (Figure 3.3-1) is the visual representation of an 'average' eye. This is created by looking at a number of eyes and creating a mean model over the entire set. That model is then compared to every section of the test image and the most similar two results, in theory, would be the eyes. The results are shown above right (Figure 3.3-2). A similar technique can be used to localise the other features.

The template image is compared to every part of the test image in a sliding window fashion and an algorithm is applied to ascertain how similar the template is to the patch being investigated.

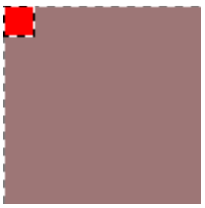

Figure 3.3-3 - Test patch

Figure 3.3-4 - Sliding Window test image

The above figures show how the sliding window approach works. The red square (size x by y Figure 3.3-3) represents a subset of the test image, a window within a picture. That subset is slid across the test image (Figure 3.3-4) so that every set of pixels (size x by y) is processed in turn.

The second main method used to localise features is the edge detection method; either by detection of edges or by detection of blobs and then using some sort of principal component analysis to establish which blobs are collection of detected edges correspond to each individual facial features. The image below illustrates the use of edge detection and blob detection (Kawaguchi, Rizon, & Hidaka, 2005) algorithm:

Figure 3.3-5 - Results of blob detection

As you can see from the above image, it is fairly easy to locate the eyes, eye brows and hairline once the other information has been eliminated. From there it is straightforward to estimate the location of the nose and mouth from the remaining blobs, as it is always located directly below the mouth.

3.4 Feature matching

Matching the selected feature to the feature on the test face is a critical part of facial recognition and if done incorrectly, invalidates the entire recognition process.

Four commonly used methods are cited (Zelek & Ersi, 2006):

- Correlation: Templates of a known face and of an unknown face are compared to ascertained differences.
- Feature cluster: The degree of similarity between two faces is ascertained by 'clustering' certain areas of the face.
- Set feature/aspect partitioning: certain features on the test face will have vectors that are very different to those on the training images. Therefore, using a process of elimination, those features can be removed and faces categorised as non-matching.

- Euclidean Distance: A comparison of the feature vectors on a training set of images against a test image reveals how closely related the faces are and therefore the smallest number should yield a facial recognition match. This is the most commonly used matching scheme.

The approach to be used depends on the data being utilised but some of the methods are more common than others.

3.5 Image Normalisation

When an image is captured from the capture device, whether it is a static photograph, a video segment or a live video feed, the image is in an unnormalised state. This means that the images could have any pose, any lighting conditions, and the background may have unwanted clutter. All of these conditions as mentioned above could have a detrimental effect on the accuracy of an automated face recognition system.

Normalisation was an important process to legacy facial recognition systems because of their brittleness. If a face was not exactly in the same position as the reference faces used to create the model, then a match would be almost impossible to obtain.

The process of normalisation, in a traditional sense, refers to a sequence of steps that are completed prior to any matching taking place. The objective is to remove variations in position, scale, and rotation of the facial image (Brunelli & Poggio, 1993). Normalisation takes place during the learning phase and on test data before any attempted match is made. This makes it possible to match static faces that are at different angles to the camera, under different lighting conditions or if the face was closer to or farther away from the camera than when the reference face was acquired. Traditionally, two points on the face may be chosen in order to ascertain the scale and position of the face prior to matching. This process relies on accurate feature extraction. Obviously, this does not take into account the effect of lighting; in order to standardise the lighting of an image so that it may be used for recognition purposes a series of filters may need to be used to standardise the lighting at each pixel location (Heusch, Cardinaux, & Marcel, 2003).

In the area of automated facial recognition from video, normalisation is difficult to perform in real-time. Serious thought must be given to the way in which images are normalised. It is assumed for the purposes of this report that the training data (if used) will be fully normalised but that the test data will be input from an unpredictable live video source and as such will not be normalised. Although, the pose of the head will be within certain limits due to the fact that a ‘front on’ face recognition system is being developed, not a ‘profile’ system.

3.6 Face Detection Techniques

The face detection itself is quite a distinct activity form the recognition of a face and is performed before any feature extraction can take place. The process is important because it reduces the search space from an entire image to just a facial area and reduces the likelihood of errors during feature extraction. The process needs to be fast, so that it can be performed in near real time and ideally should be invariant to many of the obstacles mentioned in section 2.1.1.

3.6.1 Viola and Jones Face Detector

The Viola and Jones face detector (Viola & Jones, 2004) represents a leap forward in facial detection techniques. It can perform accurate facial detection at a rate of fifteen frames per second on standard home architecture (Pentium 3). The detection process is split into 3 main stages; each stage is based upon work previously done in the field. The uses for a well designed, efficient and robust face detection system are extensive. Some of these uses are summarised in Table 3.6-1 below:

Areas	*Specific Applications*
Biometrics	Drivers' licenses
	Immigration, National ID, Passports, Voter Registration
	Welfare Fraud
Information Security	Desktop Logon
	Application Security Database Security, File Encryption
	Intranet Security, Internet Access, Medical Records
	Secure Trading Terminals
Law Enforcement and Surveillance	Advanced Video Surveillance, CCTV Control
	Portal Control, Post Event Analysis
	Shoplifting; suspect tracking and investigation
Smart Cards	Stored Value Security, User Authentication
Access Control	Facility Access, Vehicular Access

Table 3.6-1 - Face Detection system uses (Zhao, Chellappa, Rosenfeld, & Phillips, 2007)

The detection system uses an AdaBoost learning algorithm:
In an AdaBoost algorithm a number of weak classifiers are combined to create a smaller number of strong classifiers.

The simple learning algorithm is a weak learner; this is because even the best classifiers do not classify the data particularly well. In order to boost the classifier, it must first be made to solve a series of learning problems. After the first round of learning, the training set is re-assessed with weights according to the accuracy of the classification. Hence, this is where it learns the information it needs to better classify the data in subsequent training cycles.

The final strong classifier is a weighted combination of weak classifiers followed by a threshold. AdaBoost essentially finds a small number of good features, which have significant variety.

In the Viola and Jones' face detector, the AdaBoost algorithm is used to minimise the number of Haar-like features being computed. It does this to increase the computational efficiency of the detector by eliminating features that are unlikely to be of any help to the overall classification. Each week classifier is constrained to a single feature for the purposes of this classification.

Viola and Jones used several other important methods to build their state-of-the-art facial detection system:

- Focus of attention approach:
  This approach allows the system to locate areas of an image where the face is most likely to occur in order to avoid over-processing an area where a face definitely will not occur.

- The integral image:
  The integral image is an intermediate representation of the original image which contains a subset of that image; at point (x,y) the integral image is all of the points above and to the left of point (x,y).

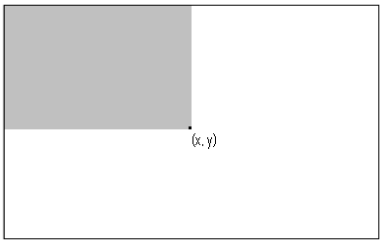

Figure 3.6-1 - The integral image example

This provides the basis for the cascading detection used to quickly filter out unnecessary data.

In the Viola and Jones implementation, the integral image is an image that ignores the intensity of the original. Integral images can be computed extremely quickly and the results have the benefit of being able to compute haar-like features at any scale or location in constant time. The system does fully realise that the rectangular feature matching system that it uses is imperfect for matching to variable shapes.

However, the extreme computational efficiency more than makes up for this short fall and fully explains why alternatives like Steerable filters are not used.

Using classifiers, the focus of attention operator discards 50% of an image at a cost of approximately 1% of genuine faces. The classifiers are evaluated in 20 simple operations per location.

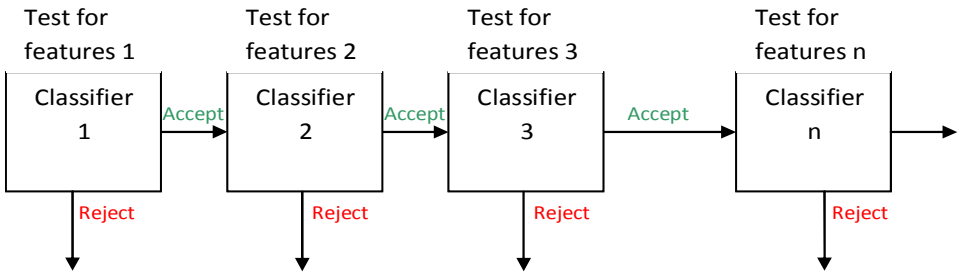

Figure 3.6-2 - Weak classifiers used by Face Detector

The classifier cascade described above removes sub windows that do not contain faces form the original image. Each classifier eliminates more non-faces whilst retaining nearly 100% of the actual faces.

The Viola and Jones face detection system performs fifteen times faster than the next most efficient face detector constructed by (Rowley, Baluja, & Kanade, Neural Network-Based Face Detection, 1998).

Using features for classification, as opposed to pixels saves a great deal of wasted computational efficiency, the system is therefore much more effective than its predecessors given the same time frame. This speed is proportional to the number of features evaluated. A balance has to be found where the minimal number of features is evaluated in order to obtain a 'good' rate of successful facial detections.

The Viola and Jones system was tested on a well-studied facial test set. The results against this were excellent and the fact that the set was extremely complex proves the reliability of the system.

The system itself is slightly less accurate than previous best efforts because the weak classifiers, by definition, can cause important information to be lost without being properly evaluated. This is an accepted risk of the system and its accuracy is well within acceptable limits. In addition, this slight loss in accuracy vastly improves the speed of the algorithm making it a much more viable choice for real-time face detection.

3.6.2 Sparse Network of Winnows (SNoW) Face Detection

Another type of face detector has been created in the past utilising an approach known as the Sparse Network of Winnows (SNoW) approach (Roth, Yang, & Ahuja, 2000).

The sparse network of winnows approach is a learning architecture consisting of a sparse network of linear units over an incrementally learned feature space. Each linear unit represents a face or a non-face pattern. The face image (or non face image for learning purposes) is offered as an input layer to the linear units (the input layer consists of a number of values representing the features contained within the input image[1]) and these are used to augment the linear units accordingly.

[1] The values representing the features can be Boolean (indicating presence or lack thereof) or real (indicating the strength of the given feature)

Once the linear units have received all of their input data, they are linked via weighted edges to input features to build a model of the face. The presence of a face is dependent on the presence (or lack) of active a number of active features. Each individual feature is active if its weighted edge value is greater than a threshold value. The threshold value is set during the learning phase. Also during the learning phase, the algorithm must be offered positive (face) examples and negative (non-face) examples to learn from, these must be labelled at input time.

The algorithm can be shown as:

$$\sum_{i \epsilon At} w_i^t > \theta_t$$

Equation 3.6-1- SNoW-based face detection model building

Where $\sum_{i \epsilon At}$ represents the set of features in the image being tested is, w_i^t is the weight of the edge connecting the i[th] feature (in the set) to t (the linear unit) and θ_t is the threshold to determine which values should be taken into consideration.

The learning algorithm is mistake driven. It therefore utilises an update rule, in this scenario, this is the Winnow update rule. The linear units are thus updated as the algorithm learns; $\alpha > 1$ is used as a promotion parameter when the algorithm works correctly and $0 < \beta < 1$ is used as a demotion parameter when the algorithm is incorrect. This is how the learning feature works and weights correct responses accordingly. The key to the Winnow update rule (the only rule applied in this detector) is that the number of examples it requires to learn a linear feature grows linearly with the number of relevant features and grows logarithmically with the total number of features being used.

The SNoW-based face detection algorithm is extremely accurate in detecting faces given appropriate learning material. In addition it can accurately locate a face in a number of poses, scales and under varying lighting conditions. It is therefore extremely robust. However, when compared to the algorithm above (Viola & Jones, 2004) it has a slight advantage in terms of accuracy but cannot compete in terms of speed of processing. This is due to the fact that the algorithm uses more costly processing and evaluates all areas of the image, not saving any processing power by utilising a weak classifier prior to any other processing. Indeed, this is not suitable for this algorithm. Also, the SNoW-based face detector cannot currently detect rotated faces. This, therefore, creates the need to impose difficult to enforce normalisation rules on the video image[2].

Finally, no information is available with regard to this approaches ability with processing video frames. Although, every indication is that the algorithm could process multiple frames per second, it is likely to perform much less effectively than the aforementioned method.

3.7 Feature Extraction Techniques

Following the successful acquisition of a human face from a cluttered background, it is necessary to extract several of the features from that face quickly and accurately in order to process those features and

[2] A solution does exist to this problem (Rowley, Baluja, & Kanade, Rotation Invariant Neural Network-based Face Detection, 1998) but offers even further reduced performance.

attempt to recognise the face. There are many proposed methods for the successful acquisition of facial features; some of the most popular and more recent advancements are described in the following sections.

3.7.1 Bayesian approach to feature localisation

This is an approach, in part, attributed to a paper prepared on eye localisation (Everingham & Zisserman, 2006). According to which, is more successful and affective than some other approaches mentioned in the same paper. It is described below.

The accuracy of facial recognition using Principle Components Analysis (PCA) or Linear Discriminant Analysis (LDA) is known to degrade with poor eye localisation. This approach offers a robust method to locate the eyes reliably, thus improving the overall accuracy of facial recognition systems using the above stated methods. Essentially, the eyes are considered to be amongst the most important features of the face and it is important that they are located using a reliable method.
The system attempts to classify patches as either being not an eye or a patch containing an eye.
Three approaches to the successful localisation of the eyes are explorer. Each one is different and has varying degrees of success:

- Regression method:
  - Parameters are learnt by minimising the Euclidean distance between true and predicted eye positions. This type of learning approach should offer promising results over a large dataset.
- Bayesian Method:
  - The Bayesian method is essentially an algorithm that gives a positive output if the patch does contain an eye or a negative output if the patch does not contain an eye. This makes it easy to classify if any particular patch does, or does not, contain an eye. A number of algorithms are proposed in order to implement this method.
- Sliding Windows:
  - Bayesian and discriminative approaches are used to locate the eyes. Square patches are extracted from the test image along a line in a 'sliding window' fashion. The classifier is then applied to a number of pixels, using each pixel in the square area as the centre of the constant value γ has the effect of expanding the distribution to make the matrix useable.

These approaches are compared in Figure 3.7-1 below:

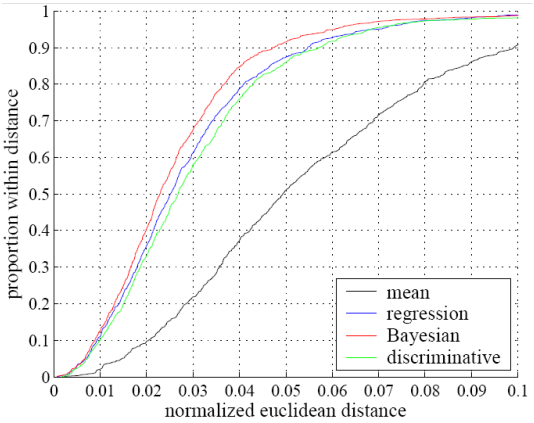

Figure 3.7-1- Results of errors of classification and regression eye localisation methods performed on FERET database (Everingham & Zisserman, 2006)

According to the published results for these methods (Everingham & Zisserman, 2006) and as briefly mentioned above the Bayesian approach offers the most reliable results amongst the popular methods offered by this paper. Figure 3.7-1 above shows how successful the Bayesian (classification) approach actually is. Given that it is also the most simple and computationally efficient algorithm to implement, it is should be very effective effective.

In 90% of images, the eye was localised to within 2.04 pixels during extensive tests. The total difference between the two eyes is therefore up to 4.08 pixels per pair of eyes. These are excellent results. However, there is potentially an identification problem with this type of detection error because it could fool the matching scheme into believing the subject is somebody else with similar eye spacing distance. Conversely, this is likely to be the best algorithm available for accuracy and computational efficiency.

Because this method was decided upon for use in the final prototype, the exact details about its implementation are presented in section 5.2. This section is meant as an overview to the method for consistency.

3.7.2 Two-dimensional PCA-based feature localisation

Principal Components Analysis (PCA) has been used extensively and successfully in the area of facial recognition. The process has been in use since the early nineties with many examples of its use recorded (Kirby & Sirovich, 1990).

PCA is used commonly in the field of computer vision as a cluster analysis tool. It is designed to capture the variance in a dataset in terms of principle components. The dataset refers to the images being used for comparison (in this instance the face training images.) The PCA process attempts to reduce the amount of data in the images whilst retaining the defining parts and simultaneously filtering out noisy data, such as background and other non-facial data. This model is illustrated in Figure 3.7-2. Normalisation can remove some of the variance in the data making the PCA process less effective.

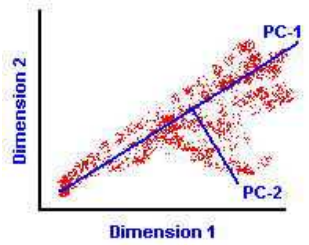

Figure 3.7-2 - Principal Component Analysis example (University College London; Oncology, 2006)

The blue lines above represent two separate principal components (PC-1 and PC-2). In reality, a face would have many more principal components. PCA is based upon covariances and the information they give about multiple dimensions of data. The key to principal components are that they work better with increasing dimensions of data as they take advantage of compression by removing noisy data that is existence in covariance. The principal components are selected from the single-dimensional eigenvalues with the highest values (Smith, 2002).

The above paragraph should explain why PCA is so useful to use with complex facial data.

PCA has many advantages but is becoming an old method and improvements have been made. The single biggest problem with PCA is that it is single dimensional. Improvements have been suggested (Yang, Zhang, Frangi, & Yang, 2004).

The optimal projection vectors of two-dimensional PCA $x_1, \dots, x_d$ are used for feature extraction in a similar way to traditional PCA. Given a sample image **A:**

$$Y_k = \boldsymbol{AX_k} k = 1,2, \dots, d$$

Equation 3.7-1 - 2DPCA Optimal Projection Vectors

Then, we have a family of projected feature vectors, $\boldsymbol{Y_1, \dots, Y_d}$ which then form the principal components vectors[3] of the given sample image (in this case **A**). From these vectors, a matrix is created called the feature matrix. The feature matrix is then compared to its neighbours to assess their similarities in much the same way as PCA does.

Method	*Recognition Rate*
Fisherfaces	94.5%
ICA	85.0%
Kernel Eigenfaces	94.0%
2DPCA	**96.0%**

Table 3.7-1 - Comparison of face recognition methods, adapted from (Yang, Zhang, Frangi, & Yang, 2004)

Perhaps unsurprisingly, the results of this approach offer better recognition rates than standard PCA (and in fact many other results too, see Table 3.7-1 above.) However, something which is surprising about this method; it uses less principal components than standard PCA offering a performance boast (see Figure 3.7-3 below.) Despite this, PCA is still a lengthy process and the processing time involved in 2DPCA prior to testing is extensive.

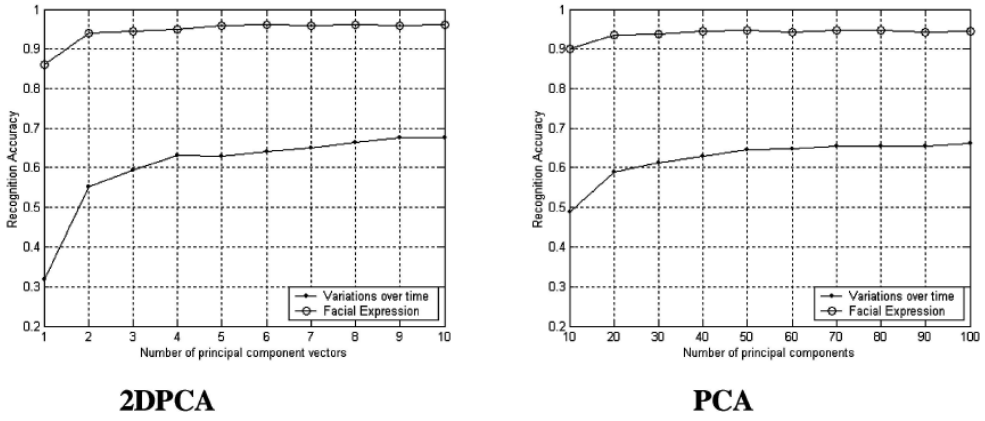

Figure 3.7-3 - Comparison of no. of PCs in PCA & 2DPCA (Yang, Zhang, Frangi, & Yang, 2004)

[3] In 2DPCA these values are vectors, in PCA they are scalars

In addition, because the PCA process attempts to extract a subspace where the variance is maximised (and the reconstruction error is minimised) changes in lighting and facial expression can cause unreliable results. Especially given that these changes in lighting and pose cause greater variations in the images than a change in facial identity, making identification extremely difficult.

3.7.3 Edge-based shape detection

The method now being considered is an edge-based shape detection algorithm. It is likely that whilst attempting to recognise objects, humans (at least partially) look at the edges of that object to ascertain its shape and whether that edge belongs to the object at all. Essentially the approach looks to extract full shapes rather than parts of an edge map. This has several advantages over the edge map approach, outlines below.

A recent edge-based shape detector is proposed (Moon, Chellappa, & Rosenfeld, 2002):

The approach considers the generalised Hough transform as a means of detecting features in test images. However, it also recognises the approaches limitations; it is rotation invariant and relies on salient points in order to work properly. Therefore, this particular approach leads to poor feature localisation. Fitting a polynomial curve to the outline of a face has also been considered in much the same way as an active-shape model, but the shape is too approximate to be of any use. Traditional edge detection algorithms suffer from too much emphasis on localised information and can discard edges without evaluating all of the relevant information.

In order to ascertain the presence of a particular shape, the pixels around the test pixel are evaluated for changes to intensity rather than examining the pixel itself. In this method, we must define the shape matched operator by expanding the most favourable step edge operator (which minimises the noise and mean squared error) along the speculative object boundary contour so that responses from intensity differences along the boundary are collected at the same time in order to improve performance. The responses to this process are gathered at the centre of the operator where a log-likelihood ratio determines whether a shape is present or not.

Because facial features are easy to model as two-dimensional shapes, they can be located using this method; circular irises, intensity differences at the corners of the eyes, and the shade between the upper and lower lips. To locate an eye, a circular shape can match to the eyes whilst a line could approximate the eyelids directly above. A straight-line segment can be used to approximate the position of the mouth and this shape can be bent by moving a pixel in the middle of that line to approximate facial expression, see Figure 3.7-4 for a visual example.

Figure 3.7-4 - Feature detection using shape matching

This method offers an extremely simple method for the accurate location of facial features by matching basic shapes. This model could truly work in a real-time environment and offers some resilience to facial expression changes. However, the model is susceptible to changes in pose (outside the bounds of regular movement) and the fact that the algorithm relies on intensity changes in certain places, significant changes in image illumination could have a negative impact on this process.

The method proposed here utilises edge detection and model fitting to basic shapes. The likelihood of a match is a probability function extremely similar to the one cited in section 3.7.1 (Everingham & Zisserman, 2006).

3.7.4 Kernel Eigenfaces and Fisherfaces approach comparison

Principal Components Analysis (3.7.2) and Fisher linear Discriminant methods have been successful in the area of facial recognition. However, they only look at lower level statistical dependencies between pixels and do not address higher levels such as the relationships between a number of pixels (three or more).

In order to address this problem the use of Kernel Eigenfaces and Kernel Fisherfaces is proposed here (Yang M.-H. , 2002).

The idea of this method is to expand PCA, Eigenfaces and Fisherfaces approaches so that they may take account of higher order statistics and make use of a greater order of surrounding pixels because it is believed that this extra information is the future of facial recognition. Kernel Eigenfaces and Fisherfaces as presented here, not only attempt to extract higher order statistical information but also maximise the class separation when these features are projected onto lower dimensional space for easier recognition.

3.7.4.1 Kernel Eigenfaces (Kernel PCA)

Basic PCA is explained in detail in section 3.7.2, as part of the explanation of two-dimensional PCA. The explanation here explains how the Kernel method attempts to make the algorithm more efficient. In simple terms, each vector is projected from the input space to a high dimensional feature space, in order to take advantage of higher order statistical data. This transformation is handled by a non-linear mapping:

$$\varphi: R^n \to R^f, f > n$$

Equation 3.7-2 - Eigenfaces Transformation equation

Where R^n is representative of the input space, R^f represents a high dimensional feature space. The feature space can be very large. For R^f, the eigenvalue problem is defined as:

$$\gamma \boldsymbol{w}^{\varphi} = C^{\varphi} \boldsymbol{w}^{\varphi}$$

Equation 3.7-3 - Eigenvalue problem equation

C^{φ} is a covariance matrix acting as a regularisation term in this case. All solutions $\boldsymbol{w}^{\varphi}$ lie in the span of $\varphi(x_1), \ldots, \varphi(x_m)$ where x is a vector projected from the original PCA algorithm, m is given here as the number of components. There exist coefficients (α_i) for these such that:

$$\boldsymbol{w}^{\boldsymbol{\varphi}} = \sum_{i=1}^{m} \alpha_i \varphi(x_i)$$

Equation 3.7-4 - Eigenfaces coefficient equation

The above equation denotes an m x m matrix K by:

$$k_{ij} = k(x_i x_j) = \varphi(x_i).\varphi(x_j)$$

Equation 3.7-5 - Matrix equation derived from Equation 3.7-4 above

The above equations form standard PCA results, the preceding equation becomes a Kernel problem:

$$m\gamma K\alpha = K^2\alpha \equiv m\gamma\alpha = K\alpha$$

Equation 3.7-6 - Kernel problem equation

Above, α denotes a column vector with entries 1 through m (form the matrix). All projected samples are centred in the high-dimensional feature space. The vectors in that high-dimensional feature space must be projected onto a lower space spanned by eigenvectors $\boldsymbol{w}^{\varphi}$ for matching schemes to attempt recognition. **X** is a test sample whose projection is $\varphi(x)$ in R^f:

$$\boldsymbol{w}^{\varphi}.\varphi(\boldsymbol{x}) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{\alpha_i}(\boldsymbol{\varphi}(\boldsymbol{x_i}).(\boldsymbol{\varphi}(\boldsymbol{x})) = \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{\alpha_i k}(\boldsymbol{x_i}, \boldsymbol{x})$$

Equation 3.7-7 - Eigenfaces matching scheme equation

3.7.4.2 Kernel Fisherfaces

The test vectors x are again centred in high-dimensional space R^f. Within-class scatter matrix S_w^{φ} and between-class scatter matrix S_B^{φ}, it is necessary to find eigenvalues γ and eignvectors w^{φ}:

$$\gamma S_w^{\varphi} w^{\varphi} = S_B^{\varphi} w^{\varphi}$$

Equation 3.7-8 - Eigenvalues and Eigenvectors derivation equation

These can be obtained by some matrix calculations, normalising the matrices and giving a vector as output.

The kernel function is then described as:

$$(k_{rs})_{tu} = k(x_{tr}, x_{us}) = \varphi(x_{tr}).\varphi(x_{us})$$

Equation 3.7-9 - Kernal function equation

Let K be an m x m matrix defined by the elements $(K_{tu})_{u=1,\dots,c}^{t=1,\dots,c}$ where K_{tu} is a matrix composed of dot products in the high-dimensional feature space.

Without covering the rest of the proof of the equation, the main equation to be used by this method can be shown below; more details are available in a recent paper comparing these methodologies (Yang M.-H. , 2002):

$$\begin{aligned} \boldsymbol{w}_{OPT}^{\varphi} &= \arg max \boldsymbol{w}^{\varphi} \frac{|(\boldsymbol{w}^{\varphi})^T S_B^{\varphi} w^{\varphi}|}{|(\boldsymbol{w}^{\varphi})^T S_w^{\varphi} w^{\varphi}|} \\ &= \arg max\, \boldsymbol{w}^{\varphi} \frac{|\alpha K Z K \alpha|}{|\alpha K K \alpha|} = [\boldsymbol{w_1^{\varphi}} \dots \boldsymbol{w_m^{\varphi}}] \end{aligned}$$

Equation 3.7-10 - Main kernel equation

Projecting this high-dimensional representation into lower dimensional space for comparison can be achieved in much the same way as in Kernel PCA in section 3.7.4.1. $\boldsymbol{w}_{OPT}^{\varphi}$ above is the set of optimal eigenvectors gathered from the process by means of evaluating the class scatter matrices S_w^{φ} and S_B^{φ} respectively. The extracted eigenvectors gathered from the equation directly above are Kernel Fisherfaces; utilising the standard Fisherfaces method described by many sources(Belhumeur, Hespanha, & Kriegman, 1997) and(Choi, Lee, Lee, & Yi, 2001), but extrapolating the results to high dimensional space for analysis before reducing them to lower-level orders for comparison and matching.

Unfortunately for this method, even during the training phase, each image was down sampled to 23 x 28 pixels because of the complex mathematics involved in multiple matrix calculations. According to results (Yang M.-H. , 2002) done on these methods (amongst others) the Kernel Fisherfaces results were the most successful. In addition, the kernel methods for Eigenfaces and Fisherfaces performed better than their original counterparts. The methods both perform better than their traditional counterparts when considering them in the context of pose and expression variations. Although, it is worth noting that the SVM (Support Vector Machine) method performed better than any of the methods under these conditions.

In short, under normalised conditions and with no expectations of processing efficiency, the kernel augmented methods perform better than traditional Eigenfaces/Fisherfaces. The error rates are markedly lower and the accuracy rates are high.

However, the methods are computationally inefficient and as mentioned, have serious problems dealing with facial expressions and pose (as do conventional PCA-based methods). Therefore, this approach may

be suitable in a situation where normalised images are used in a static environment and processing can be done offline. They are not particularly well suited to live (unnormalised) video sequances.

3.7.5 Support Vector Machines (SVMs)

Support Vector Machines is a pattern classification algorithm providing a means of training information used by polynomial classifiers and neural networks (Vapnik, The Nature of Statistical Learning Theory, 1995), (Vapnik & Cortes, Support-Vector Networks, 1995).

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane separates classes of objects (Statsoft, 2004). For example; if a decision was being made between dogs (red in Figure 3.7-5) and men (green in Figure 3.7-5), a Golden Retriever being added to the decision plane should be classified as a dog (and represented as red in Figure 3.7-5), Figure 3.7-5 illustrates:

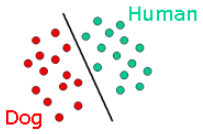

Figure 3.7-5 - Decision Plane example

The above problem separates linear classes (those that can be split by a line) most classifications are more complex than that:

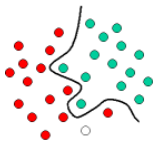

Figure 3.7-6 - Hyperplane example

Figure 3.7-6 depicts a more complex case of classification; the circular dot in white represents a test case requiring classification. It can therefore be classified as red. The process of SVM is the process of mapping the representation in Figure 3.7-6 to that shown in Figure 3.7-5:

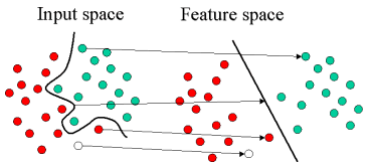

Figure 3.7-7 - Hyperplane mapped (using kernels) to Decision plane

Training these classifiers for facial detection requires an enormous amount of training data. The system as described above has been implemented previously for facial detection (Osuna, Freund, & Girosit, 1997). There is no reason why the same approach cannot be applied to the detection of individual features. In the

face pattern detection scenario, the system is trained using literally thousands of examples (50,000 in the example cited) so that the model can be as accurate as possible. The test image is scanned exhaustively at numerous scales and orientations to locate and match the facial shapes. Extending this to features is trivial; a separate model needs to be built for each of the features and these can be compared to the test patch as it is extracted as part of the search space.

There are obvious issues with this algorithm; firstly, the 'exhaustive search' suffers from serious computational efficiency problems, the addition of some weak classifiers would be beneficial but currently, this algorithm is far too inefficient to be used in real-time.

Secondly, the process of actually obtaining the training images is by no means simple. 50,000 noiseless images for each feature being detected is extremely difficult to find and utilise effectively (memory requirements alone would be extreme for this).

Finally, the algorithm itself is unsuitable in some situations; extreme variations in lighting may affect the greyscale images so much that feature matching may become unreliable anyway. In an unnormalised environment (such as the one in acquisition and processing of video sequences) lighting variations are real and difficult to remove effectively.

Conversely, in a normalised environment, SVMs may offer a very accurate means of detecting and recognising faces. However, the process is still likely to be far too slow for anything other than static image analysis.

3.8 Conclusion

A number of current approaches to the extraction of facial features have been proposed. The efficiency of these algorithms will vary. However, a decision had to be made based on the expected efficiencies. The Bayesian approach in section 3.7.1 is simple, supposedly efficient and extremely reliable. It has therefore been chosen as the main feature extraction algorithm to be used.

In addition, the face detector approach also had to be decided upon. Although, the SNoW-based approach discussed in section 3.6.2 offers certain accuracy advantages, the performance gains offered by the weak classifier approach in section 3.6.1 mean that it is the obvious choice.

Chapter 4 – SYSTEM DESIGN

4.1 Introduction

The purpose of this chapter is to illustrate the design process that took place during the development stages of this project. The chapter begins by looking at design specifications and some requirements capturing. It then moves on to covering design methodologies, relevant design diagrams and finally it looks briefly at the way the graphical user interface has been designed.

4.2 System Requirements & Specification

The system specifications are created by considering the user's and system requirements respectively.

4.2.1 System Requirements

Number	Requirement
1	The face recognition system should work directly from a live video feed (or buffered feed if the hardware is incapable of supporting real-time recognition)
2	Reference images are normalised before being uploaded to the database
3	System should utilise one or more of the methods mentioned in section 3.7 for feature extraction
4	System should utilise one or more of the methods mentioned in section 3.6 for face detection
5	Reference users should be easily removable from the database
6	GUI should show matched face over set interval (i.e. every 10 seconds the image is updated)
7	Information entered at the GUI should be validated properly
8	Error handling should be elegant and extensive

Table 4.2-1 - System requirements table

4.2.2 User Requirements

Number	Requirement
1	The Graphical User Interface (GUI) should be simplistic to use and intuitive
2	It should allow the user to upload reference images by specifying locations and subject's name
3	The system should respond in real-time when identifying a person (where hardware permits)
4	The GUI should display the live video feed and the closest person match in the same window
5	Errors should be clearly displayed when necessary

Table 4.2-2 - User Requirements Table

4.2.3 System Specification

Number	Specification
1	The Graphical user interface will be written in a high-level programming language that has well founded support for interface components. The Java Swing library or C# visual components would be suitable. Panels nested within frames will be used for the visual layout (JPanel or Panel respectively.)
2	All progress messages and errors will be printed to the console by default. Serious errors will be displayed in a dialog box where necessary
3	Separate java frames with relevant information boxes will be displayed allowing the addition or removal of reference faces. These frames will be accessible from a menu system on the main GUI window.
4	The images available for upload (reference images) can be in any standard format

Number	Specification
	(jpg, png, gif, bmp)
5	Text boxes will be made available for the entry of the subject's name
6	A list component will be used to display the users in the database for removal

Table 4.2-3 - System Specifications Table

4.3 Face Detector Specification

The choice of face detector is important; it needs to operate in real-time and must accurately locate a face in unnormalised conditions. The choice of face detectors is derived from the comparison in section 3.6.

It must exhibit the following specifications in order for it to be viable in this scenario:

Number	Specification
1	The face detector must run in real-time (hardware dependant)
2	It must accurately locate faces at a variety of scales, rotations and under various lighting conditions
3	The system must be compatible with the high-level programming language of choice (Java or C#)

Table 4.3-1 - Face Detection Specification

4.4 Feature Localisation Specification

The feature localisation algorithms are chosen from the comparisons made in section 3.7, the approach being investigated here for the purposes of specification gathering is detailed in section 3.7.1, a Bayesian approach to feature localisation (Everingham & Zisserman, 2006).

The equations being used here are derived entirely from a recent paper in this field (Everingham & Zisserman, 2006). The specification is split between training and testing.

4.4.1 Data Training Specification

It is essential to the successful operation of a (learning-based) feature detection system, that the learning phase be executed correctly. This is to ensure the appropriate acquisition of the mean feature, covariance matrix and to guarantee that this data is as noiseless as possible. For this particular method, a log-likelihood ratio will be collected. The largest value is then deemed to be the closest match.

Number	Specification
1	Acquire training images from pre-existing database (FERET or OWL)
2	Create mean image from entire collection of training data for each feature (left eye and right eye)
3	Extract covariance matrices and standard deviation from data obtained in step three
4	The Gaussian distribution can be formulated in the following way: $$p(x\|c) = \frac{1}{(2\pi)^{d/2}\|\sum\|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T\sum^{-1}(x-\mu)\right\}$$

Number	Specification
	(This will be modelled in a programming language for implementation, full details on how this algorithm is implemented are available in section 3.7.1)
5	The distribution in step 4 is compared to potential eye patch and a log-likelihood ratio is used to assign confidence as to whether the patch is an eye or not: $$llr(x) = \log p(x\|e) - \log p(x\|\bar{e})$$
6	The resulting values will be compared to those stored in a database in order to obtain a match. The method will also be used to obtain the features from reference images and those values will then be uploaded into a database for long-term storage.

Table 4.4-1 - Feature extraction training specification

4.4.2 Data Testing Specification

It is the responsibility of the system to compare the features obtained from the live video source and compare the features to those within the reference images[4].

Number	Specification
1	The test image is to be obtained from a live video source
2	The face detector is run on the image to ascertain the position of the face
3	He facial area is extracted and its size normalised depending on the size of the test image, for this it will be rescaled using language image processing tools
4	A 'sliding window' approach is applied to the test image so that every grouping of patches (size k, where k is the size of the patch of the image obtained from the training section)
5	The log-likelihood ratio mentioned above is used on every patch to ascertain which is most likely to contain an eye image

Table 4.4-2 - Data Testing Specification table

At this point it is expected that the proposed system will utilise face detector (Viola & Jones, 2004) and at least one of the feature recognising techniques, probably the Bayesian approach (Everingham & Zisserman, 2006). The graphical interface will be built separately from these other components and the above specifications give a good indication of how the system is expected to work.

At this point, java was selected as language to be used for the proposed system. This was due to the fact that it was easier to create a wrapper for the OpenCV version of the face detector system being used.

4.5 Description of Design Methodologies

The design work carried out during the development stages of this project consist of a few main areas, namely; the design of the face detection system, the ability to process live video footage and the implementation of at least one of the feature localisation methods discussed in section 3.7. The work is fully structured and is modular in conception; therefore a similar approach is adopted for the design

[4] It is assumed that the same features have already been extracted from the reference image when it is added to the system as mentioned in earlier chapters.

methodology. In effect, this means the system is being developed in a structured but modular fashion. Use of DFD's (Data Flow Diagrams), use case diagrams and other visual design methods are used to reduce complexity and illustrate decisions made.

There is an essential step between the illustration of information flows and the inception of classes to support these flows. Therefore, the industry standard methodology for object orientated design is used. This is the Object Orientated Approach (OOA) and it allows better modelling of class interaction.

The final part of this chapter briefly looks at the design of the visual components of the graphical interface so that they conform to the requirements of the users as specified in section 4.2.2.

The design of the system was originally going to be implemented in C# (pronounced c-sharp) because of its close relation to C, the speed at which it processes and the author's familiarity with that language. However, because of the lack of support available for this, relatively new, language and the fact that there have been virtually no attempts made in the computer vision community using C# an alternative was decided upon instead. The language to be used now is Java; it operates independent of platform, has huge online support and has been used extensively in the field of computer vision. It is generally considered to be slower than C# because it is essentially an interpretive language but performance is not expected to be a constraining factor in this implementation.

4.5.1 Description of Information Flow

The designs mentioned in this section can be found in section Appendix A. The data flows are depicted at various levels in order to appropriately model the information flows without providing too much information in any single diagram.

There exist two diagrams showing the overall information flow of the system. Figure 8.1-1 shows the overall system process for the face recognition system. It shows that a set o training images are compared to an image acquired from the face detector in an attempt to obtain a match for the face. Figure 8.1-2 forms a sublevel of the above high-level figure. It shows that the face detection system accepts a buffered image as input and returns a cropped picture as a result containing the subject's face only.

Figure 8.1-3 depicts a level 1 representation of the feature training process. It shows that the covariance, mean image and scaling information is derived from a database of training features and stored in a java data store for use in the main program. These values are then applied to the main program along with the live face detector in order to attempt a facial match. Sub processes exist for these steps in two separate forms; training and testing.

The training stage is shown in more detail on the level 1.1 diagram in Figure 8.1-4. This diagram shows that the greyscale images are derived from the chosen database (most likely FERET) and stored in .dat files for the java program to read. The exact process for the derivation of these values is shown in Figure 8.1-5. This diagram is a level 1.1.1 data flow and shows the exact process undertaken for the extraction of the mean, scaling and covariance values needed for the matching process. In short; the patches are taken one at a time and the feature (eye, mouth etc) is extracted. That feature is then normalised and reshaped to a vector (rather than a matrix). The covariance is determined and then inverted to create the inverse

covariance matrix, which in-turn, is saved to a .dat file. The scaling value is determined and saved and the mean feature is extracted. The mean feature is centred and reshaped as a vector before being saved to secondary storage.

As mentioned above, the second main process is the testing phase of the feature comparison. The level 1 diagram depicted in Figure 8.1-3 and mentioned above also describes the process 'feature detection testing'. This sub process is extrapolated in the level 1.2 diagram shown in Figure 8.1-6. This information flow diagram shows how the test face (as acquired from the face detector output) is systematically searched for the feature as trained during the training phase. The algorithm should take the size of the patch (size k, as determined during the training phase) as input and then systematically examine every set of pixels (size k) in that test image and compare the subset of pixels against the mean values obtained during the training stages to find a match. The actual matching process, once the measurements have been taken, is described in more depth in the diagram in Figure 8.1-7. This is a level 1.2.1 diagram and shows how the location of the features in the test image is compared to those in the reference images. The Euclidean calculation is shown in Figure 8.1-8, a level 1.2.1.1 diagram subtracting the test image feature location from each of the reference images. If the result of this calculation is zero the people match. The further from zero the result, the less likely that the two faces match.

4.5.2 Description of State Transition

The state transition can be found in section 8.1.9; Appendix A, it depicts the different states that the face recognition system can enter and aids in the design of the 'multiple frames' detection filter.

The system begins in an initial state, i.e. before the system has started. On initial execution, the system will begin to train the images (if it can) and begin the facial detection process. The training state on the diagram is distinct from the other states and is set off manually. It is usually run once and the data retained but can be retrained if necessary.

The actual face recognition process is executed in a loop. The system waits for the frames from the video source and extracts them one-at-a-time in a loop, prior to this, the system must have the data from the feature training in order to continue. The system must then await extracted faces and process them one-by-one in order to locate the features for matching. Failure to find a face or recognise a face accurately simply sends the system back to the beginning where it begins the process again. Final state here refers to the successful localisation of features prior to resetting the system.

4.5.3 Object Orientated Analysis (OOA)

A complete class diagram for the face detection system (written in java) is supplied in section 8.1.10, Appendix A. This diagram is intended to show the relationship between the various classes implemented within the system.

It was decided, using a combination of techniques (the design methodologies in section 8.1) to implement the system according to object orientated best practices. In order to do this it was necessary to create 14 classes excluding test classes and inner classes (those classes that are defined privately within other classes.)

Justification for the use of the aforementioned classes is in the next section:

4.5.4 Justification of class choices

FaceRecogGUI class: This class represents the main interface for the face recognition system. All messages are passed via this class and this class collects panels from the other classes and displays them in the correct order in a frame.

ImageDisplay class: This class provides the main 'back-end' class for the GUI mentioned above. It liaises with the face detection and feature detection classes and does the actual matching before passing the result to the GUI for display.

FaceLocations class: This class creates an arraylist object and stores the locations of faces. It takes the face from the face detector class and processes the result so that the x and y coordinates of the face is stored here.

FrameAccess class: Provides the face locations class with individual faces from individual frames.

FaceDetection class: The class provides the FrameAccess class (above) with the face locations on each frame (image) collected form that class.

FaecCrop class: Takes the individual frame from the FrameAccess class and combines it with the face location from the face detector class in order to crop the image leaving only the face to return.

FaceLocation class: Basic storage class containing accessors and mutators for the locations of the face; x, y and width.

Face class: stores individual information about the face. The subject's name, x and y locations if the eyes and cropped facial image are held here for the purposes of the reference face and the comparison are all stored here.

FeatureLocalisation class: This class attempts to localise the facial features using the methods investigated by this report.

EyePatch class: describes an individual eye patch so that the eye patches can be stored in an array and easily compared to find the relevant features.

Matrix class: A class providing useful Matrix functions that can be used by the feature finding process.

FaceDataAccessLayer class: This class provides all the program's database reading and writing functions in one place. All the relevant information is sent to this class for writing to the database.

FaceAddToDatabase class: This class is an interface; it provides a separate GUI to upload a reference picture containing a face and some attributes describing that image to a database for storage.

FaceDeleteFromDatabase class: This is similar to the above *FaceAddToDatabase* class, it provides a separate GUI and a list collected to the database for easy removal of references faces from the database.

4.5.5 Explanation of message passing (Figure 0-1)

(1) FaceDataAccessLayer → FaceAddToDatabase/FaceDeleteFromDatabase → FaceRecogGUI
This message flow represents the error message flow from the database back to the main GUI in the event of a failed call to the database

(2) FaceRecogGUI → FaceAddToDatabase/FaceDeleteFromDatabase
This message represents the invocation of the classes at the receiving end of the call. A second GUI is created in response to allow editions to the database to be made.

(3) FaceRecogGUI → ImageDisplay
This message is designed to invoke the imageDisplay class and begin the process of video frame retrieval and feature localisation.

(4) ImageDisplay → FeatureLocalisation
Once the imageDisplay class has been invoked as per step number 4, the featureLocalisation class is called and the face image is passed to it so that the features can be found and saved to secondary storage.

(5) ImageDisplay → FaceLocations → FrameAccess
This message call represents the request for individual frames from the video source. It then shows the the faces are stored in the FaceLocations class before being passed back to the ImageDisplay class for further processing

4.5.6 Graphical User Interface

The design of the user interface takes into account the user and system requirements mentioned in chapter 1. The GUI (Graphical user interface) must allow the easy matching of features and the easy addition and removal of reference faces from the program.

The main GUI window, when executed, begins the face recognition process automatically for whatever object is directly in front of the live video source (webcam). It can be seen in Figure 8.2-1 in **Error! Reference source not found.**.

From the main window mentioned in the above paragraph, there is a menu system (file, in the top left of the screen) that allows the addition and removal of reference faces from the system database. The windows to support these functions can be seen in Figure 8.2-2 and Figure 8.2-3 respectively. The add screen allows the upload of a reference face image and a name to match that image to the database, using standard windows 'browse' functionality. The 'delete reference face from database' window provides a list of names from the database and the option to select a name and click ok to confirm deletion.

As you can see, the GUI windows have been designed to allow maximum functionality with minimal components. The Main GUI window has a live video feed source on the left hand side of the screen and a space on the right for an image to be displayed with a name to denote the matched reference face.

Chapter 5 – SYSTEM IMPLEMENTATION

5.1 Introduction

This chapter describes how the face recognition was implemented. It explains specifically how the equations for feature detection were implemented, how the video component was added and how the database interface was made. The chapter details differences from the intended design in Chapter .

5.2 System Implementation

The implementation was originally planned to follow the design as laid out in Chapter . As expected, some problems were encountered during this work and were dealt with in a way that varied from the design slightly. Those areas are explained in subsequent sections, whilst this section is reserved for the explanation of how the complex problems have been implemented.

5.2.1 Feature Detection Training (Eye)

The basis of this face recognition system lies, in part, on the ability of the feature detection algorithm to accurately localise facial features. Although this part of the study could conceivably have been written in Java, the built in functions available in Matlab make the process more streamlined. Therefore, a decision was made early in the development of this system to utilise Matlab for the training portion of the prototype. The algorithm developed here is based on the training part of the Bayesian approach to feature localisation examined in section 3.7.1 (Everingham & Zisserman, 2006).

The Matlab file prepared for this is shown in full in section 8.3.1 of Appendix C and is shown in pseudo-code below in Figure 5.2-1:

```
Create array (rightEye[]) to hold eye patches
For each eye (i) in FERET database
      Extract right eye patch
      Normalise eye (size and greyscale)
      Reshape eye from Matrix to column Vector
      Add eye patch to rightEye[] array
Extract the mean (M) eye image from the rightEye[] array
Centre the data obtained about the mean
Compute the covariance matrix (S)
Create Gaussian distribution
Save scaled mean (M) to file
Save inverse covariance matrix (Sinv) to file
Save standard deviation value (s) to file
```

Figure 5.2-1- Training Data calculations pseudocode

This function has therefore output the mean, covariance matrix and standard deviation to .dat files for use in the java program when it is run.

5.2.2 Data Storage/Manipulation Classes

A number of classes have been written in order to store data in or do basic data manipulations. These are listed here with brief descriptions if appropriate, especially where there are differences from the original class diagram:

- Face.java class (Figure 8.3-2, Appendix C);
  - A class used to store information about the reference faces held within the database for matching purposes.
- FaceCrop.java class(Figure 8.3-3,Appendix C);
  - This class accepts a buffered image and the size of a rectangle as input and returns the sub-image of that image cropped according to the size and location of the rectangle. It can also accept a regular image object and crop that appropriately. This is designed for static-image testing of the face detection and feature localisation process.
- EyePatch.java class(Figure 8.3-4,Appendix C);
  - This class was not present in the original design. Its purpose is to store information about each of the patches investigated during the feature localising process. It stores the x and y position of the bottom-left corner of the patch, the size of the patch, an index and the likelihood that that patch contains the feature being searched for. This is for easy reference later.
- Matrix.java class(Figure 8.3-5,Appendix C);
  - This class is another class added since the initial design. It became apparent that a number of matrix calculations would need to be made by a number of classes (especially if this system was going to be extended). Therefore, this simple class provides a number of matrix multiplication, subtraction and column and vector calculations on behalf of other classes.
- OpenFileAction.java class(Figure 8.3-6,Appendix C);
  - The OpenFileAction class provides a GUI to open files from the file store and hence obtain the complete file path in a visual way. It offers this capability to other classes as they need it. Although this class is not shown in the original class diagram, its presence here simply reduces the time taken to write the code multiple times and does not reflect poor design technique.
- FaceLocation.java class(Figure 8.3-7,Appendix C);
  - Another class designed to store information about particular features. This class stores the location (x and y) and rectangle size of a face in relation to a particular image.
- FaceLocations.java class(Figure 8.3-8,Appendix C);
  - This class is a logical extension of the FaceLocation class mentioned above. It implements the face detector class and retrieves the faces (possibly multiple) from that class. It then creates a number of FaceLocation objects to represent those faces and stores them in an array for further processing.

There have been some deviations from the original design, as can be observed above. A new class diagram with all of the updated information will be made available at the end of this section.

5.2.3 Database Manipulation Classes

There exist a number of classes that are used to interface with the database. The purpose of these classes is to add and remove reference faces. The actual database integration is handled by a single data access layer class (in line with good programming practice (Aarsten, Brugali, & Menga, 1996).) This data access layer class, FaceDataAccessLayer.java handles all reading and writing to and from the database. Classes needing to use the database can simply call methods in the data access layer in order to facilitate that access. The class can be seen in Figure 8.3-9 of Appendix C.

The data access layer class above essentially offers database access to two of the other classes. Because the database is used in this context as persistent storage for the reference faces, the classes needing to access the database are those which add and remove reference faces to and from the database respectively. The classes in question here are the FaceAddToDatabase.java and FaceRemoveFromDatabase.java classes; both classes provide graphical interfaces for the addition and removal of reference faces and the database. The source code for these classes can be found in Figure 8.3-10 and Figure 8.3-11 of Appendix C respectively.

5.2.4 Video handling class

It became apparent during the design phase of this project that a class would be needed that would not only have the necessary codecs to be able to decode video files, but also have the ability to interact with live video sources and extract single frames of the videos and treat them as images. The full source code can be found in section 8.3.12 of Appendix C. In short, the class is responsible for decoding a video source so that it can be used and providing the individual frames as images for further processing.

Unfortunately, java offers no native video handling capabilities. The consequence of this is that a prerequisite condition is necessarily added to the entire program. The program must be made to use the JMF (Java Media Framework[5]), it is therefore necessary to install this prior to executing the program.

The class sets up a media player with individual frame access and uses the Windows native webcam as its source. The program utilises a number of codecs to successfully decode video sequences. The program uses JMF standard media control panels to support the video stream (pause, examine etc.) and uses a Media locator object to track the video source. The media stream is created, the frame accessor is assigned to listen to this stream, the controller components are added to the video display and that entire display is attached to a panel in order to be offered to the main GUI window.

5.2.5 Face Detection class

The face detection class has a simple job; it examines images and returns the locations of the faces (x and y position of the top-left hand corner of the face and the width of the square rectangle enclosing the face.) The class creates an object of class JNIOpenCV[6]; this class is stored in a .dll file and implements the OpenCV version of the face detector (Viola & Jones, 2004). It is, in effect, a java wrapper of a C program. The Face detection class acts as an interface for this wrapper and has been designed so that it

[5] The Java Media Framework is an open sourced development project aimed at enabling media support for java applications.
[6] Provided by Dr. Hu software organisation (Dr. Hu Software, 2002)

accepts an image and returns a list of face positions for processing. The next step is to pass these values to the face cropping algorithm to extract just the face. The full java source code can be found in Figure 8.3-13 of Appendix C.

5.2.6 Overview of main program functionality

The above classes provide storage, Graphical interfaces and low-level data manipulation tasks. The main program core is very much more complicated than that. From a high level perspective; the program accesses individual frames from a webcam and adds those frames to a queue. In the meantime, a separate thread of execution constantly processes the frames in the queue; detecting the faces and localising the features. The dual threads of execution are shown in pictorial form in Figure 0-2 of Appendix A. It is necessary that this functionality of the program be provided by a separate thread because accessing the work queue must be done on a constant basis for the duration of the overall program execution. In order to implement, there were two options:

- Create a timer for the program to periodically poll the queue for new frames to access
    - In theory this approach works quite well but in practice, without dynamically updating the time interval at which the queue is checked, the queue will quickly become overloaded. Especially given that the items in the queue are likely to be processed significantly slower than they are added. Thus, an unnecessary overhead is added to the queue.
- Create an infinite loop where the queue is constantly accessed and the frames are processed.
    - This invariably involves the creation of an infinite loop where the program continuously checks the queue for the presence of additional work to complete. In the event that the queue is empty, the loop will wait for an item to be added. As mentioned above, in order to prevent the program entering an infinite loop and then not completing the other processing (the extraction of individual frames), this loop would be implemented in a separate thread of execution and then both parts of the program (frame extraction and frame processing) would continue concurrently[7].

Once the frames of video have been extracted, the feature localiser works by extracting sub-images from the test face (patches). Then comparing these features to mean features already obtained during training. Every possible patch is examined by iterating over all possible x and y locations in the image. The most likely patch location is returned to the main program for comparison; see Figure 0-3, Appendix A. The actual sliding window approach is therefore critical to the successful operation of the entire system. In pseudocode it performs the following:

```
for y = height/2 to y = height - patch size //Look at top section of face only
        for x = 0 to x = width - patch size  // look at each row in turn
                Read sub-window as buffered image
                Convert sub-window to greyscale
                Convert sub window from matrix to vector
                Compare test vector to training mean feature image
```

[7] It is possible with modern processors to set processor affinity to specific threads so that the threads would be executed in a truly concurrent manner.

```
Store log-likelihood result for later comparison
```

Figure 5.2-2 - Pseudocode for sliding window approach

In the Bayesian approach discussed in depth in section 3.7.1, probabilistic models of the appearance of a patch are built independently for positive and negative classes.

The log-likelihood output of these probability models is then used to ascertain whether a patch is an eye or is not, according to the aforementioned probability models. Simply, the patch with the greatest log-likelihood ratio is deemed to contain the eye. Obviously, there should be two high-valued log-likelihood ratios for each of the eyes in every image. The two highest values of log-likelihood should both be obvious and accurate detect the position of the two eyes. In practice, it may well prove to be better to split the training images between left and right eyes in order to obtain slightly better classification during the test phase.

The log-likelihood value can be created using the following formulae:

$$llr(x) = \log p(x|e) - \log p(x|\bar{e})$$

Equation 5.2-1 - Log-likelihood comparison

That is to say that the likelihood that patch x is an eye; llr (log-likelihood ratio) is given by the logarithm of x given that it is an eye minus the logarithm of x given that it is not an eye. This is the logarithm of the differences.

The complete Gaussian distribution for each eye and non-eye can be shown as such (where each class c (eye and non-eye) is modelled as such):

$$p(x|c) = \frac{1}{(2\pi)^{d/2}|\sum\quad|^{1/2}} exp\left\{-\frac{1}{2}(x-\mu)^T \sum\quad^{-1}(x-\mu)\right\}$$

Equation 5.2-2 - Bayesian mean image building equation

d is the size of the patch (k x k), x is the test patch to be classified (eye or non-eye), μ is the mean image represented as a vector and ∑ is the covariance matrix. They are shown in the following way:

$$\tilde{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Equation 5.2-3 - Bayesian mean image

$$\tilde{\sum} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \tilde{\mu})(x_i - \tilde{\mu})^T$$

Equation 5.2-4 - Bayesian Covariance building equation

There is no particular reason why this formula for the mean is used but it is as suitable as any other. The estimate of the covariance must be regularised since d (d+1)/2 parameters must be estimated. This is done by adding an identity matrix to the covariance matrix:

$$\sum = \tilde{\sum} + \gamma I$$

Equation 5.2-5 - Bayesian addition of regularisation parameter equation

The above pseudo and mathematical equations have been implemented in java in the FeatureLocalisation.java class shown in section 8.3.14.

Unfortunately due to reasons discussed in the following chapter, this implementation failed in this particular instance. In short, because the complex matrix calculations take far too much time to complete, the queue being fed from the frame extractor class becomes overloaded very quickly. Therefore, although the training algorithm remained the same, Equation 5.2-1 was changed to be in the line with the pseudocode presented Figure 5.2-3. More details are given below.

As a work-around to this problem, the above implementation was changed. The mean feature was extracted in much the same way but is then compared directly to the test patch (extracted as part of the sliding window method) using a sum-of-squared difference approach. This has the unfortunate side-effect of being less accurate because the standard deviation and covariances are not being utilised. However, the process is fast enough to prevent system degradation within the first few iterations. The original class listing can be found in Figure 8.3-14. The change mentioned can be shown in java by replacing lines 370-402 with the following section of code:

```
double sumSquaredRight = Matrix.sumSquaredDiffVecs(meanEyeArrayRight, testEyePatchVector);
                              double sumSquaredLeft  =
Matrix.sumSquaredDiffVecs(meanEyeArrayLeft, testEyePatchVector);
                              EyePatch thisPatchLeft = new
EyePatch(patchIndexCounter,x,y,sumSquaredRight,patchSize);
                              EyePatch thisPatchRight= new
EyePatch(patchIndexCounter,x,y,sumSquaredLeft,patchSize);
```

Figure 5.2-3 - Sum-of-squares difference java source code

As you can see, the calculation is actually performed by the Matrix class as before. It also handles thi functionality.

5.2.7 Individual frame decoding

The frame decoding is handled by a specific class. This class serves two functions. It provides a 'wrapper' to the core functionality of the Java Media Framework[8], used here to process the live video feed. It attaches the video as well as some 'media player' controls to a panel and prepares to return that to the main GUI in the form of a panel. In addition to showing the video, it also accesses each individual frame as it is decoded and converts these frames to BufferedImages, before adding sequentially to the cross-threaded work queue discussed earlier and shown in Figure 8.3-16, the frame decoding class is shown in Figure 8.3-15.

The interface used here supports a relatively high frame rate, one which is much higher than the other component (the feature localisation component) can possibly hope to support. Therefore, during the frame access process, a throttling process is added to prevent too many frames being passed to the queue and the program running out of memory. This is provided by simple sending the thread to sleep for a quarter of a second at every iteration. This has the effect of reducing the frame rate from 15 frames per second to 4

[8] Code modified from that provided as an example interface to JMF by Sun Microsystems

frames per second. This unit may be changed during testing because it should, in effect, force the process of video frame extraction to take the same time as feature localisation and therefore ensures that the program will not run out of memory even if run for long periods of time.

5.2.8 Frame handling functionality

The frame handling class is different from that originally proposed during the system design phase. Its inception, as mentioned above was due to the fact that the part of the program it is responsible for (the feature localisation within decoded video frames) needed to be placed in its own thread so it didn't interfere with the other thread of execution. In short, therefore, its function is to access the work queue on a continuous basis, locate the face, and locate the features and return. The results of this 'matching scheme' are then placed on a frame within the main GUI window as results. The matching scheme referred to here is the process of comparing the located features on the test face with those features held in a database and obtaining the result. The faces in the database are loaded into an array at the beginning of the class as Face objects (section 5.2.2). The face object is updated statically by the class and is polled on a regular basis by the main GUI class to see if a different face has been matched and then update the GUI accordingly. The face coordinates, feature locations, subject's name and the reference image are held within the face object. This class, FrameHandler.java, replaces the functionality planned for use in ImageDisplay.java. ImageDisplay.java handled static images but was too awkward to change for use as a threaded process but still serves a useful purpose as a test class for the feature localisation process with regard to static images because debugging that process during the live video testing is extremely problematic. In order to fulfil the requirements of this project, inside the infinite loop mentioned above, where the frames are retrieved from the video source on a constant basis and processed for matching; a second loop exists inside that loop in order to take advantage of a number of successive frames for the matching scheme. In effect, a number of frames are processed and the features are located, the features are then loaded into arrays as part of this loop and at the end of the loop (after 5 or 10 frames have been processed) the values in the arrays are averaged (to get the average feature distance) in an attempt to remove 'noisy' data. The matching process then takes place and the main interface's matching scheme is updated to show the matched person.

The final class diagram for the implemented system can be found in Figure 8.2-1 of Appendix A. The final database schema for face storage is also shown in Figure 0-2, and the entire code listing can be found in Appendix C.

Chapter 6 – SYSTEM EVALUATION

6.1 Introduction

The chapter looks at the different types of testing that have been performed on the programme in order to ascertain its reliability. The purpose of the testing phase is to establish that the code is reliable and that all the functionality has been tested. This first part of the chapter is designed to give a brief overview of the testing undertaken. The chapter then looks at how well the solution was implemented and areas that require further work.

6.2 System Testing

The main method of testing throughout the design of the project was to establish that each individual class was performing as expected. In order to facilitate this, each class implements its own main method and can be executed independently of the rest of the system. This is to ensure that each component works before tying them together. In addition, a number of print statements have been prepared at different points throughout the program to help locate failures when the program had been integrated and the complex threads of execution were difficult to follow. Lines 61-66 of Figure 8.3-13 show how a main method can be used to ensure that the program works as expected, the print statements give the relevant information about whether the execution was successful or not. Some print statements showing useful information about program execution can be seen in lines 279-332 of Figure 8.3-12. This class also has a test main method which isn't utilised during the execution of the main program, shown on lines 226-257. The purpose of this test is to ensure that the video sequence is working correctly and that the frames are being accessed (shown by print statements on the console.)

In addition to main methods being created, a number of test classes were created to ensure correct operation prior to the final compilation of the code. ImageDisplay.java was originally planned for use in the main program; however, the problem of threads mentioned earlier meant that this class couldn't be easily modified for use with the rest of the system because its functionality was required for testing. Its function is to take a static image, run the face detector, outline the face with a rectangle and then apply the same process to feature localisation. It was therefore used to debug the face detection and feature localisation classes. The source code can be found in Figure 8.3-18 of section 8.3.18. Other similar classes were used for the testing of the database and updating the image display, shown in Figure 8.3-19 and Figure 8.3-20 respectively. These classes were created to ensure the database imaging was being created correctly by saving to the database and then retrieving it again and to display the image on a panel in a dynamic way before using it in the main entry class.

6.2.1 System Black Box/ Functionality Testing

In order to ensure the proper function of the main interface once running, black box testing was performed on the application. A number of test conditions were created to check abnormal behaviour. The test steps can be found in Table 8.4-1 of Appendix D. The screen shots associated with this testing can also be found in Appendix D and Appendix B.

The testing included; verification of error messages displayed to end users when reference information held in the filestore was missing and ensuring that the system met the requirements.

6.2.2 Training system testing

The testing of the training system was a fairly informal process. The solution was developed in Matlab because of its advanced computer graphics programming abilities. To ensure that the eyes were the features being extracted, a number of them were displayed at random as part of the loop. Because the solution was using images from the FERET database, this was simply to ensure that the coordinates within the eyes arrays represented the centre of those eyes. It was accepted that the data was noisy (the coordinates didn't necessarily represent the exact centre of the eye) but this was unlikely to cause

problems with the overall feature representation process because there were over three thousand features per feature.

The information was displayed on the console during each phase in the training process (see Figure 8.3-1 for code) to ensure that the matrices were the appropriate sizes and that the information contained within those matrices seemed appropriate.

The overall accuracy of this system, especially when the above data was used to localise and display eye positions in Figure 8.3-18 was used to check the integrity of the training algorithm.

6.2.3 System Accuracy Testing

In order to test the system, 10 reference faces were loaded into the system. This was deemed enough to test the accuracy without hindering performance. Serious degradation in performance is likely to be seen when comparing upwards of 100 faces because the time to iterate over them in an arraylist (java object array storage) is constant and therefore increases with increased load.

The system, as per the requirements, utilises a number of frames in order to remove erroneous feature detection from the matching scheme before that matching takes place. It is expected that the number of frames will have a positive effect on the matching performance of the system.

The results are in fact quite poor. They are shown in Figure 8.4-7 of Appendix D. As you can see from the graph, the correct identification of a face occurs in between 10 and 25 percent of recognition attempts. Although, this may seem poor, the results do show that accuracy is increased, on average, as more frames are used to identify the face. This, in part at least, proves that using multiple frames of video to identify a face is beneficial even though the localisation by the eyes alone is apparently inadequate.

6.2.4 System Requirements Testing

The system has successfully implemented all of the requirements gathered in sections 4.2, 4.3 and 4.4. The results of these tests can be seen in brief in section 8.4.2 of Appendix D.

6.2.5 Test Summary

The testing process in this section (6.2) is split in to three main parts:

The first part tests the user interface and checks that all of the expected functionality is in place and that all of the potential problems with the interface are dealt with elegantly.

Then the ‘black box’ testing checked the system to ensure that unexpected behaviour was dealt with effectively and that suitable responses from the system are issued to users so that they can attempt to fix the problems.

The final part of the testing centres on system accuracy testing. This highlights the poor results obtained from using the eyes alone for facial identification. However, the results also showed the added benefits obtained from using a fusion of video frames to obtain an average feature match.

6.3 System Evaluation

The report was written alongside the development of the system for the most part. It has been split into stages to mimic that of a real software development project; a background study of the approaches commonly employed, a formal system design and a system implementation of the approaches mentioned.

The design phase was based on work carried out by software development projects and took a number of weeks to complete fully.

The implementation phase was expected to be extremely difficult; a number of problems that could not be solved during the design phase were necessarily encountered here. These problems were likely to be centred on the performance of the Java virtual machine when dealing with facial detection and the application of the chosen feature detection algorithm (Everingham & Zisserman, 2006). However, other significant problems presented themselves; the necessity of deploying a multi-threaded application and Java's lack of support for any kind of video were the most significant. One of the aforementioned possible problems also presented itself when implemented. The feature detection algorithm proposed was extremely effective in its work, as determined during static-image accuracy testing. However, when this was combined with the video reading capability, it became apparent that the complex matrix calculations required to get the algorithm working would make the system impossibly slow to use in real-time. This was realised through the fact that the work queue quickly became completely overwhelmed. Instead, a much simpler approach was developed; the mean data extracted from the training process was retained but the covariance information and standard deviation had to be excluded. The test vectors where created using the sliding window approach as per the design but were then compared directly to the mean feature vectors using a sub of squared difference approach. This had the effect of making the system viable in real-time but at the expense of the accuracy. The margin of error already exhibited by the original algorithm was then exasperated and had an effect on the overall matching scheme.

It was clear from the design that it would not be possible to iterate over the database table for each and every feature match; in order to address this, all reference faces are retrieved from the database and stored in an 'in-memory' array for quick and easy access. However, iterating over this array itself was only possible in constant time and therefore it would not be possible to overload the database with reference faces without rethinking this approach. A weak classifier used in the face detection part of the program (Figure 3.6-2 on page 17), could be used here in an attempt to alleviate this issue.

An unforeseen problem was also present with the face detection .dll file used to detect facial patterns in images; it was assumed as part of the design that this library could take an image path as input for the purposes of testing the system with static images but then take a memory-buffered image directly when the approach was integrated with video. This, unfortunately, was not possible and a work-around had to be formulated; this involved temporarily storing the image on disk after being retrieved from the frame access class. The path would then be passed to the face detector, face detection performed and the image destroyed. This could potentially have slowed the system massively, but the actual effect is unknown.

The testing phase was completed in conjunction with the implementation phase. This involved a modular testing approach where each class was tested alone, then tested with the classes that use it directly. The

frame extraction class was built independently of the feature extraction classes so that those classes could be appropriately tested with static images before the video component was added to the main interface class. The testing, as mentioned in section 6.2, involved main methods for most of the classes and print statements to show the flow of execution.

After this was completed, black-box testing was executed on the interfaces to ensure appropriate (and elegant) operation in the event of failure. These results can be found in Table 8.4-1 of Appendix D and discussed in section 6.2.1.

6.3.1 Adopted approach

The approach adopted is based on the investigation work adopted in sections 3.6 and 3.7. The face detector employed is based on the current industry standard (Viola & Jones, 2004). Tests show this worked as expected, even though it appears to curtail some of the images more than others when detecting them and potentially caused matching problems when the image was rescaled; the rescaling involves changing the facial dimensions to ensure that they are the same size are the training images (100 x 100 pixels in this case.) The feature extraction process is based on a currently technologically advanced Bayesian method (Everingham & Zisserman, 2006), but heavily modified for performance purposes.

The language chosen for the implementation was Java. This was against preference of the author but because the support for images in Java is so extensive on the Internet and many other Vision systems have been developed in this language to prove its viability, it seemed like a good choice. Its performance limitations were noted but thought to be non-critical.

6.3.2 Outcome

The approach was adopted partially based on a recent paper citing eye localisation as both important, and the only feature necessary for fast, reliable face recognition (Hjelmås & Wroldsen, 1999). Given that the system needed to accurately recognise faces extremely quickly (real-time) and also that the system was designed as a prototype, this seemed an appropriate approach to follow.

The results shown in section 6.2.3 do not support this claim, although they do clearly show that using a 'fusion' of videos frames is extremely helpful to accuracy. This means that one of the main hypotheses of the report was accurate.

With regard to the initial report objectives, the following have been met:

- The initial objective was to gain a solid understanding of some of the more recent advancements in facial recognition techniques and employ some of those in the prototype; it is believed that this has been successfully met by the work carried out in Chapter 3 and the Feature localisation class implemented in Figure 8.3-14.
    - It is conceded that the feature localisation class, as mentioned above, was insufficient to meet the requirement of a reliable face recognition program. Another approach from section 3.7 could have been applied to another of the features; section 3.7.3 details a simple approach that uses a combination of two-dimensional shapes, in varying sizes and

line curvatures to find features (eyes, mouths, eyebrows, noses etc.) This approach is recommended because of its simplicity and speed of processing.

  - In addition, the Bayesian approach used for eye localisation could be easily extended to locate other features without modifying the training process at all. In fact, the testing process (in the sliding window method) would only need to reference the new arrays created from the new features. Therefore, the addition of these features would require almost no extra processing in order implement. And crucially, no extra mathematics or complex programming techniques would need to be developed.

- The second objective was to gain a better understanding of the chosen high-level language used to implement this system. Although there is no previous basis from which to compare the author's programming knowledge, it is believed that this has been demonstrated by the size and complexity of some of the techniques employed here; low-level graphical processing, library interfacing, database connectivity, video handling and multiple thread processing are some of the more complicated techniques not previously known.
  - The designs and implementation were undertaken according to best practices and therefore it is believed that any complexity encountered was simply overcome rather than avoided. The system was also designed according to object orientated design methodologies to produce the most appropriate object orientated solution to this problem. However, several improvements could be proposed; the use of processor affinity set for specific threads would be beneficial to performance and would effectively give the separate threads a truly concurrent programming environment. In addition, it would be worthwhile spending some time auditing the source code with regard to performance. Some of the methods could be overly complex and inefficient and these should be addressed.
- The final object was to develop a complex system for real-time facial recognition. The system implements the face detection, feature localisation and recognition from a 'fusion' of video frames. It does this in real-time or approximately 4 frames per second to be exact.
  - The Bayesian approach should be extended to cover more features as mentioned above. Also, some time should have been devoted to a more accurate face detection system because scaling the shape obtained is unpredictable if though the detector does usually locate most of the face.

Finally, the database component was not part of the original design but was thought to be a suitable medium to store the face features if they were subsequently accessed in an efficient manner. It is worth noting that the database schema is not in 'normalised' form and this should be addressed in a future version. The eyes, along with other features, should be split from the main table so multiple features can be stored without over complicating the main table.

Displaying the live video in the main interface window alongside the matched face and the menu structure offering access to database management windows was also not part of the original requirements.

Finally, the fact that the Bayesian eye localisation algorithm failed to perform well enough to be used in a real-time environment meant that it would have to be replaced by a more efficient approach. The simplest,

and therefore quickest, solution to this problem appeared to be a sum of squared difference calculation. Although, this meant a decrease in accuracy, the increase in performance countered this and made it possible to meet the real-time processing objective.

When comparing the project outcome to the user and system requirements, the following can be seen:

-

Overall, the system performs relatively well given the complex programming needed to get the system working. The feature localisation should be extrapolated to take account of more features and this should make the system more reliable. The face detection algorithm also requires some tuning as mentioned above. Finally, the system becomes unstable after a couple of minutes of running. This is because the frame access class constantly adds frames to the work queue, apparently, slightly faster than the frame handler can deal with them. Therefore, the java virtual machine slowly runs out of memory. Efficiency should be sought in the algorithms developed to make the frame handler faster or the frame rate should be throttled even more than it has been already to prevent this queuing problem. In the event that both of these fail, the system should be pursued in a faster language such as C or C++.

In order to apply a more formal evaluation approach (Boloix & Robillard, 1995), a software system evaluation framework has been applied. This is used to compare the objectives and expected use of the system with the outcome. The evaluation can be seen in Figure 8.4-8 of Appendix D (page 118). The evaluation highlights the following:

These areas of application importance can be categorised as ‘advanced’:

- Tools, Products, Technology and contribution;
    - The Tools, Products and Technology refer to the frameworks that the system runs on and how they are implemented within the application. The Java Media Framework and threaded execution can be considered advanced and fulfil the requirements of the project appropriately.
    - The contribution refers to how well the overall system reflects the objectives with regard to the end users. This is considered advanced because all of the components are available in a single place and offer complex functionality.

These areas of application importance can be categorised as ‘intermediate’:

- Process followed and system usability;
    - Although the process followed is based on solid experience and examples, the problems with real-time efficiency and lack of features used overall results in the system being relatively ineffective. It is categorised as intermediate because of the excellent basis for further work that is provided here.
    - System usability is simple; but no heuristic studies have taken place to confirm this and some components are missing (ability to change the file stores, change the .dat files used and general system clean-up activities etc.) The result is a system which is functional but no really ‘usable’.

These areas of application importance can be categorised as ‘basic’:

- The performance and the system compliance
    - The performance is sub-par; partially because of the ineffectiveness of Java and partially because of the inefficient algorithms being utilised. This needs some work in order to correct.
    - The compliance is considered basic because of the lack of accessibility features available through the interface and the complex frameworks that the system requires to even run. This coupled with the complexity of changing the video source and the requirement of an SQL Server 2005 database server results in poor compliance.

Chapter 7 – STUDY CONCLUSION AND AREAS OF FURTHER INVESTIGATION

7.1 Introduction

The purpose of this chapter is to briefly conclude the learning outcomes of the project and then address any relevant areas of further investigation that are recommended for future focus of study direction.

7.2 Study Conclusions

It has been the focus of this report to study current state-of-the-art facial recognition techniques and apply those techniques to a formal design, implementation and testing process to gain a better understanding of their relevance in the field.

The project was successful to a degree (discussed in section 6.3.2 above). It successfully implemented the entire user and system requirements deemed necessary and implemented a number of complex programming methods in order to support the objectives. In order to fully satisfy the requirement that the system should provide; that is robust, reliable face recognition in real-time, it is conceded that it would be necessary to extend the programming done to include other features for more accurate matching. It may also be beneficial to provide another feature localisation algorithm to provide faster localisation than the algorithm discounted as part of the system implementation (Everingham & Zisserman, 2006).

The Bayesian feature localisation solution; training and testing can be considered extremely successful. This algorithm is easy to train and relatively easy to apply. It also offers excellent results. If it weren't for the poor performance that this algorithm appears to offer (at least in Java), it would have made a valuable contribution to this project. However, it is worth pointing out that this approach has a margin for error of approximately ± 2 pixels for each localised eye position, which means a potential 8 pixels of error between both eyes. This, in turn, equates to approximately 8 pixels of total error with regards to the Euclidean distance between the eyes overall. Given this, it was naive to assume that the eyes could be used alone to recognise a face. Having said that, the eyes are still considered to be the most important feature when recognising a face (Zhao, Chellappa, Rosenfeld, & Phillips, 2007) and should be located with a suitably accurate algorithm.

The extension of the author's Java skills is quite extreme as this project was extremely difficult to build and debug.

The major restriction on the project was the time available to be spent on it. It would have been very good to detect more features; firstly using the approach used already, then using another approach and comparing them. This was not possible however.

Finally, more research should be placed on efficient video frame processing and the order of importance of different facial features for the purposes of facial recognition. In addition, a face detector should be developed which is more suitable for this task (input and normalised face area output) or another one used which offers better consistency and performance.

7.3 Further areas of investigation

Areas of further investigation are covered mainly in the above section (7.2). They are summarised here for ease of reading:

- The key area of further work in preparing this application for commercial use lies in the extension of the feature extraction techniques already deployed. This means either:
    - Extending the Bayesian approach so that more features are used in order to match the face. As soon as these features are available, the Euclidean distances can be measured between them and a connected component analysis can result offering, potentially, a very accurate facial recognition system (Bing-Bing, Vass, & Zhuang, 1999).
    - Or alternatively, another feature detector could be used in conjunction with, or instead of, the Bayesian approach to better localise the features more quickly. This report suggests the technique covered in section 3.7.3 (Moon, Chellappa, & Rosenfeld, 2002) because of its ease of use and possible computational efficiency.
- Further to this, careful consideration should be given to the efficiency of the algorithms presented within the project. Ideally, some careful streamlining of these algorithms would make the application more effective and may allow the application to process a few more frames of video per second.
- Some investigation should be done into the newer queue systems available as part of the new Java 6.0 frameworks. It is likely that this will offer the functionality and efficiency required without the problem of synchronising the threads.
- The face detector software should be removed because of its incompatibility. It should either be replaced with another face detector available or one should be developed using the same techniques so that it performs better.
- If the methods used by the Bayesian approach could be made more efficient, the use of the Bayesian approach in its suggested state (rather than the modified version employed as part of this application) may become viable.
- Finally, a cost-benefit analysis should be conducted into translating this project into one created in C or an equally fast language to ascertain whether the extra number of frames processed could benefit greatly from doing this.

Chapter 8 – APPENDICES

8.1 Appendix A – Design Documentation

8.1.1 High Level System Data Flow

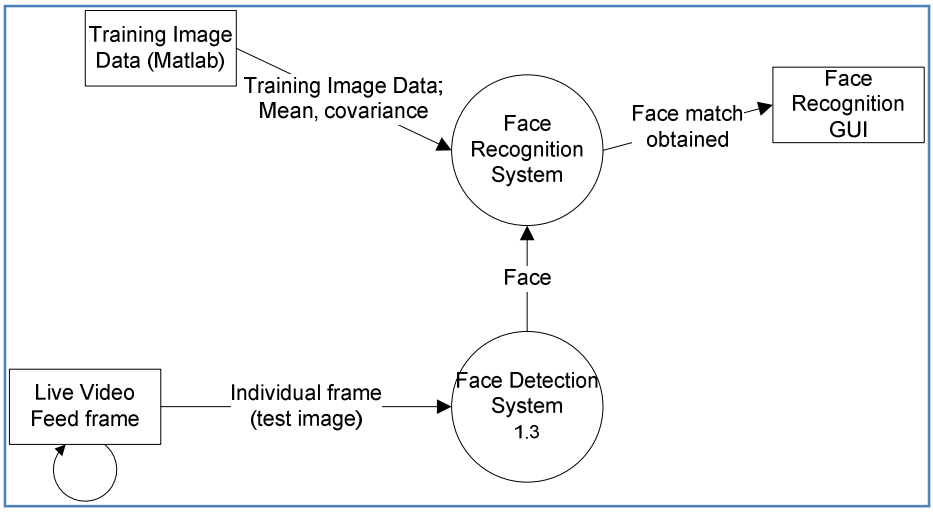

Figure 8.1-1 - High Level Data Flow Diagram (Whole System)

8.1.2 Level 1.3 High Level Face Detection flow

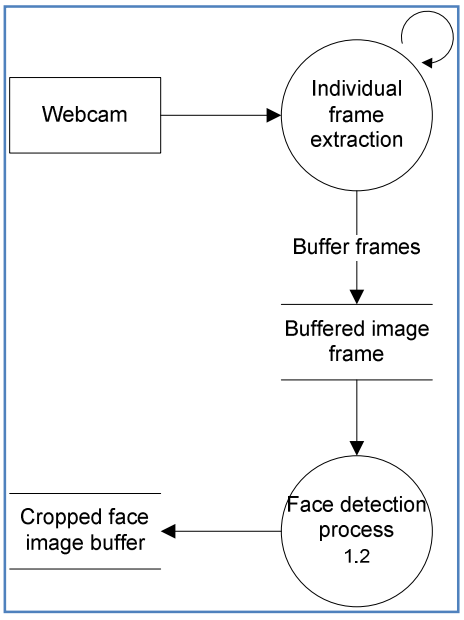

Figure 8.1-2 - High Level Face Recognition System

8.1.3 Level 1 feature training flow

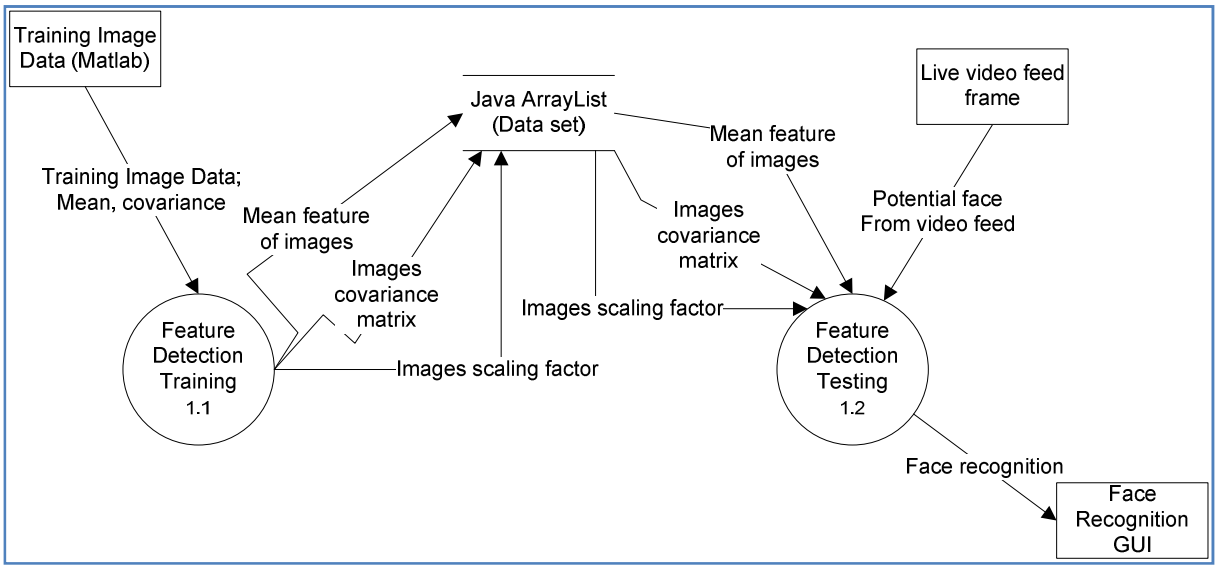

Figure 8.1-3 - Level 1 System Data Flow Diagram

8.1.4 Level 1.1 feature training flow

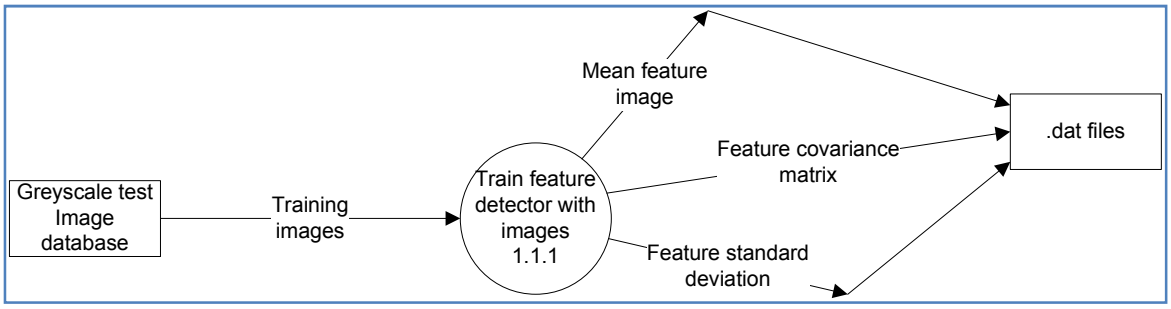

Figure 8.1-4 - Level 1.1 Feature Training Flow

8.1.5 Level 1.1.1 detailed feature training flow

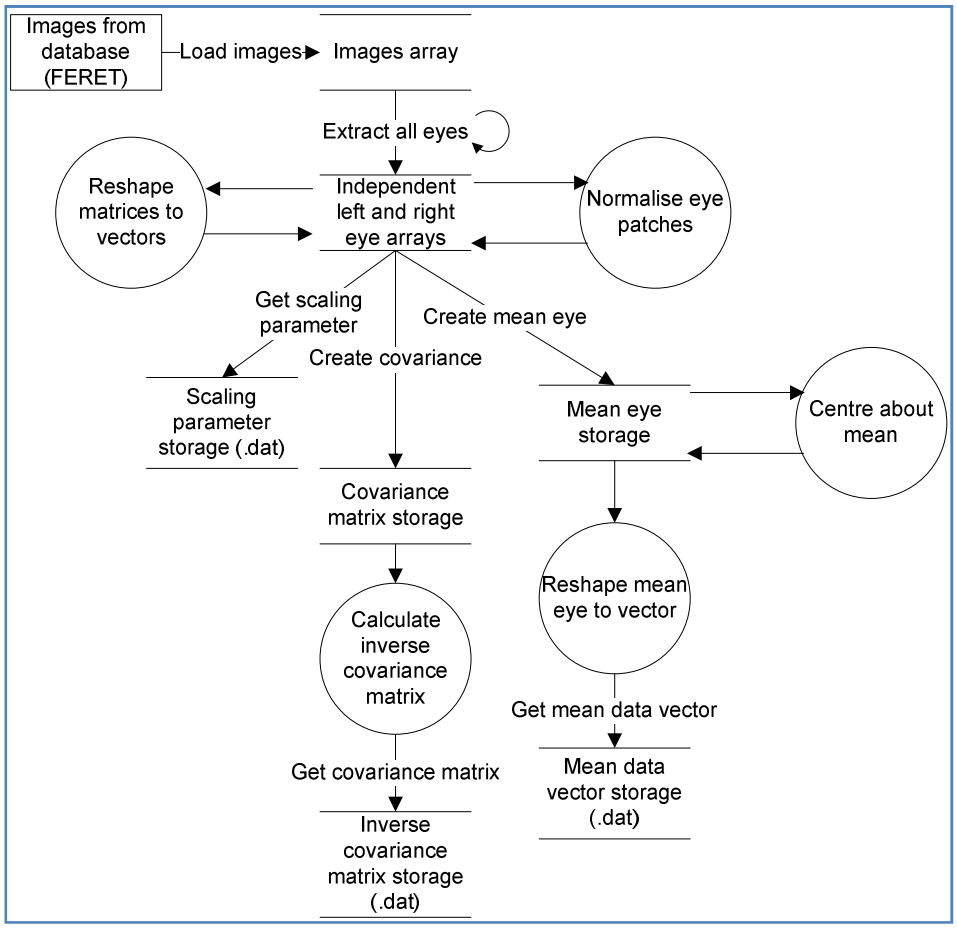

Figure 8.1-5 - Level 1.1.1 Low Level Feature Training

8.1.6 *Level 1.2 feature extraction testing (comparison) flow*

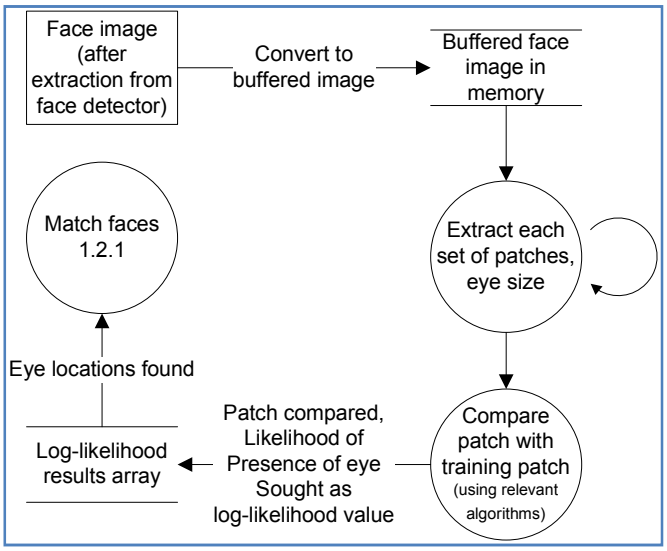

Figure 8.1-6 - Level 1.2 Feature Testing Data Flow

8.1.7 *Level 1.2.1 feature matching scheme flow*

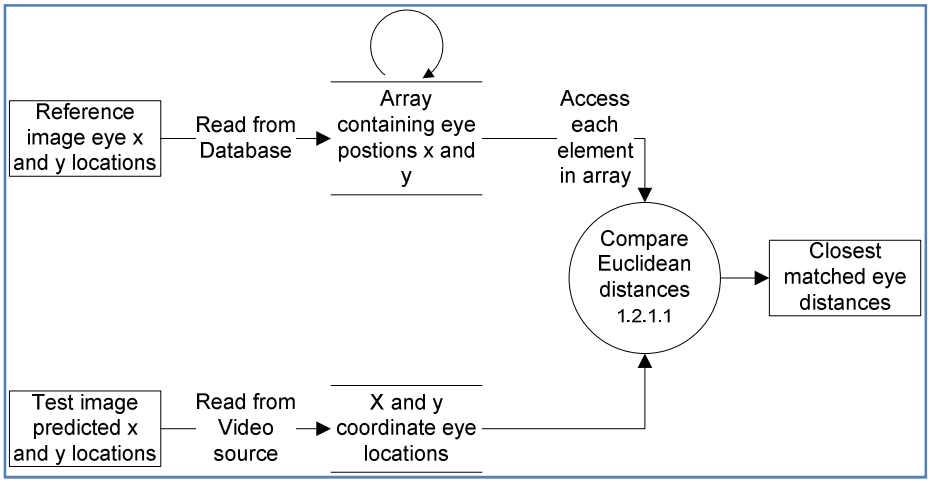

Figure 8.1-7 - Level 1.2.1 Feature Matching Scheme Data Flow

8.1.8 *Level 1.2.1.1 low level Euclidean distance comparison flow*

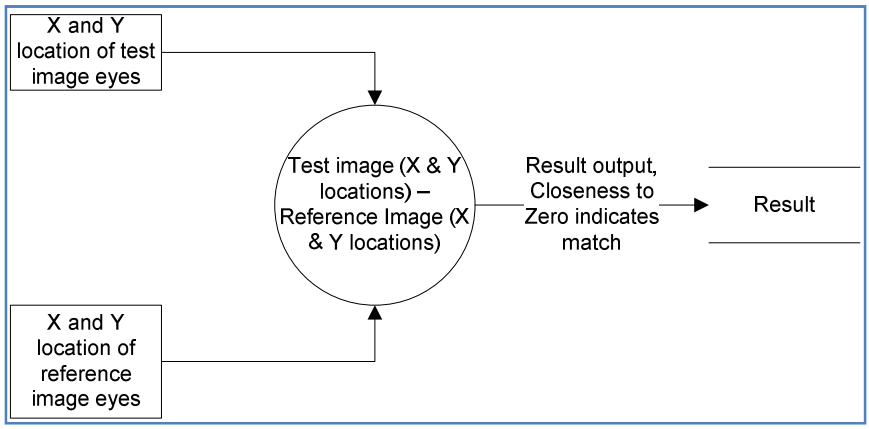

Figure 8.1-8 - Level 1.2.1.1 Low Level Matching Data Flow

8.1.9 *State Transition Diagram*

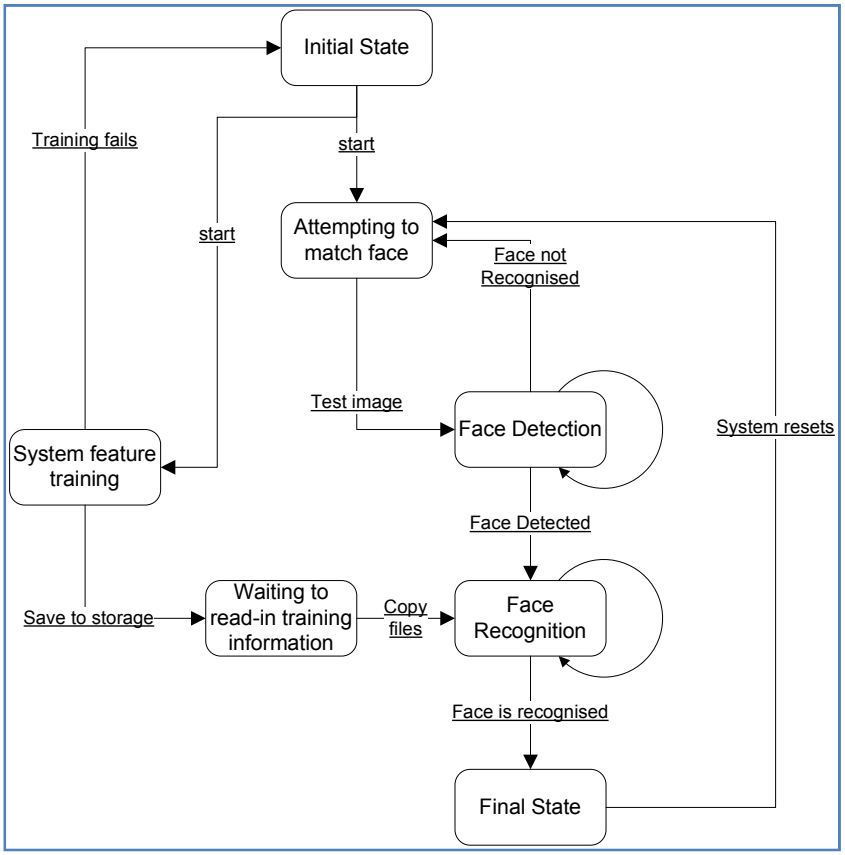

Figure 8.1-9 - System State Transition Diagram

8.1.10 Object Orientated Analysis (Class) Diagram

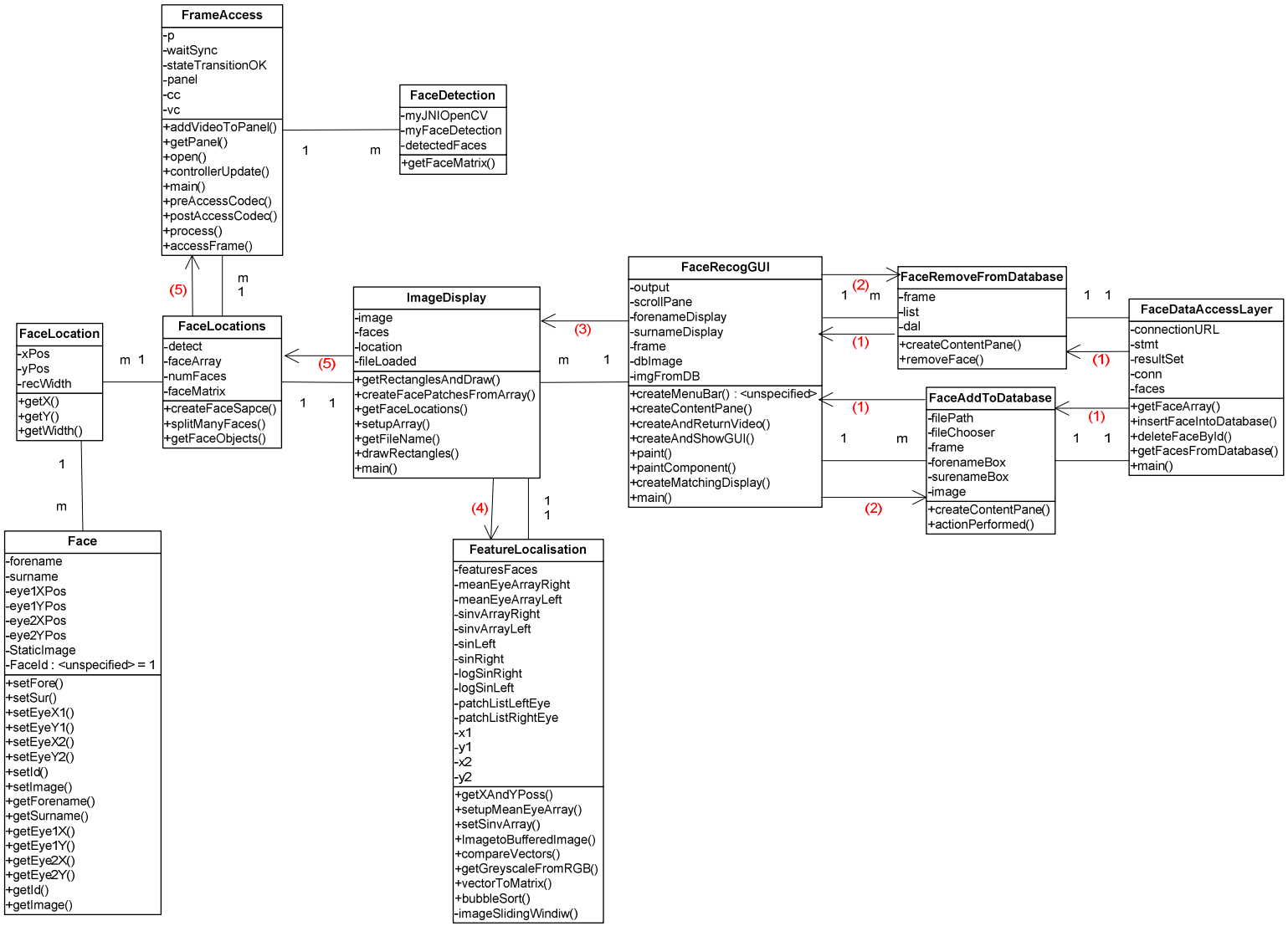

Figure 0-1- Class Diagram

8.1.11 Threads of execution Diagram

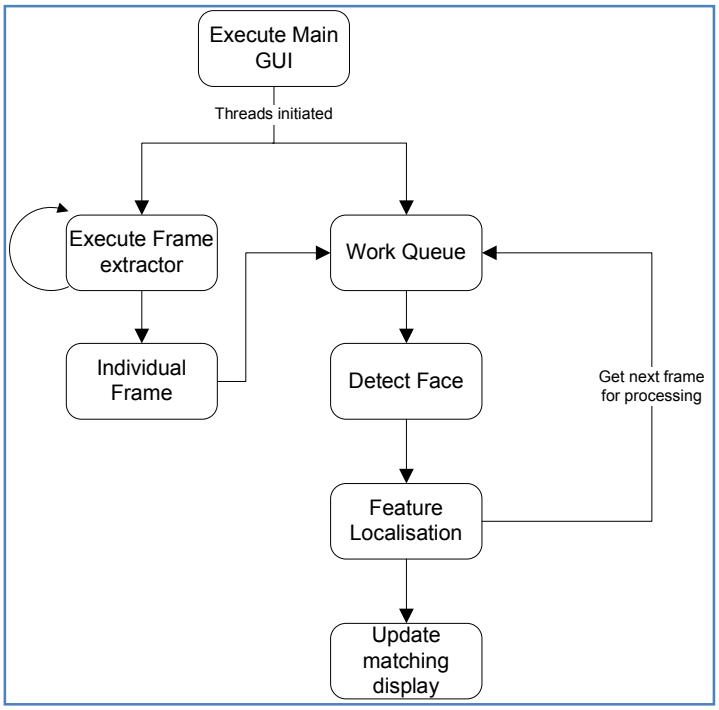

Figure 0-2 - Thread of execution (main program)

8.1.12 High Level Implementation Diagram

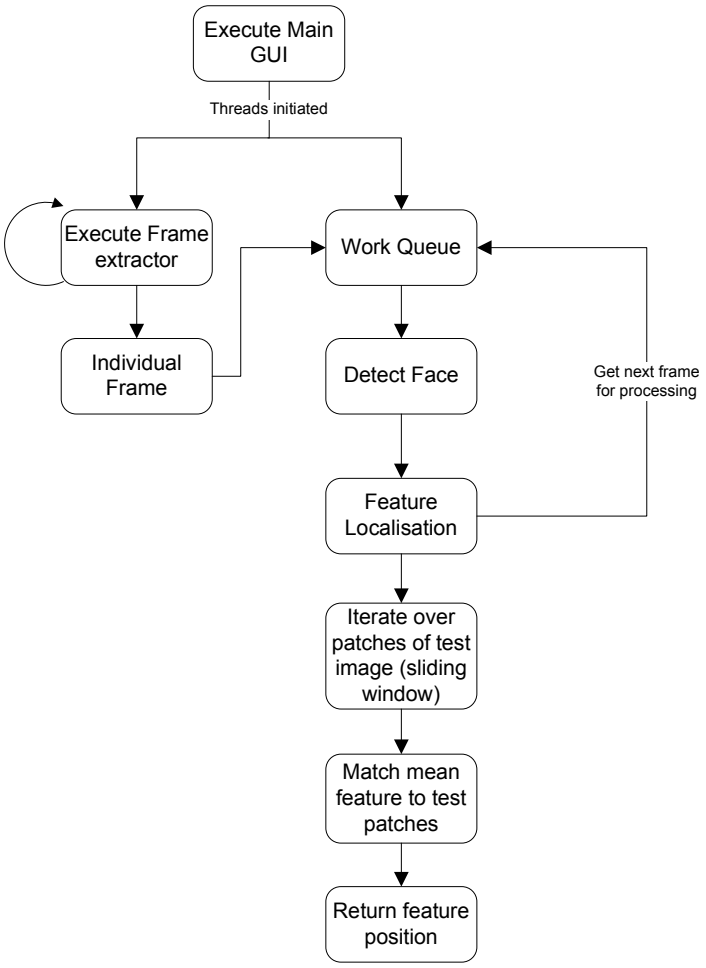

Figure 0-3 - HLD Diagram

8.1.13 Final class diagram

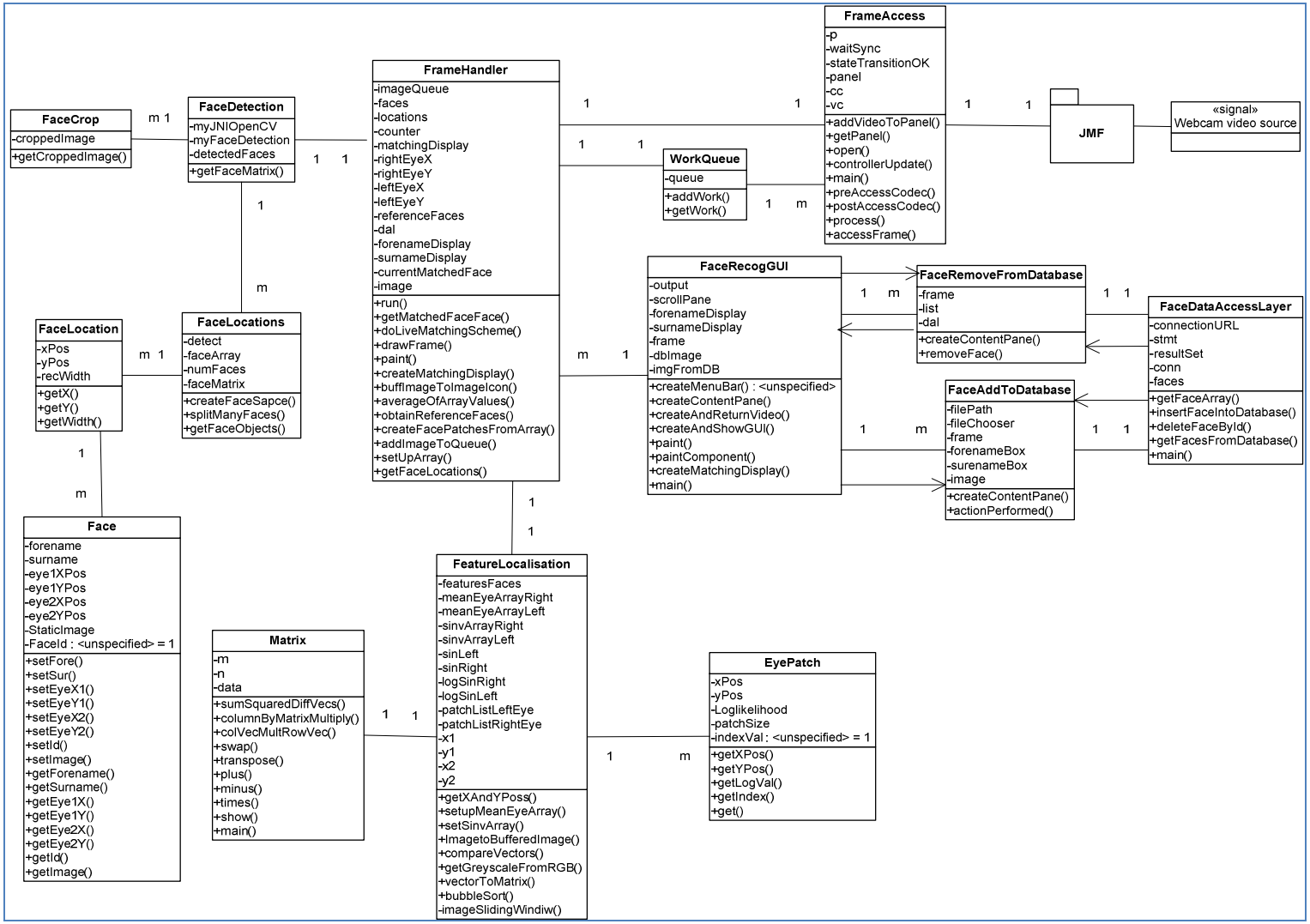

Figure 0-1 - Final Class Diagra

8.1.14 Simple Single-table Database Schema

Face	
PK	**FaceId**
	Forename Surname EyeOneX EyeOneY EyeTwoX EyeTwoY eucDistBetweenEyes filePath

Figure 0-2 - System Database Schema

8.2 Appendix B – Interface Windows

8.2.1 Main Interface Window

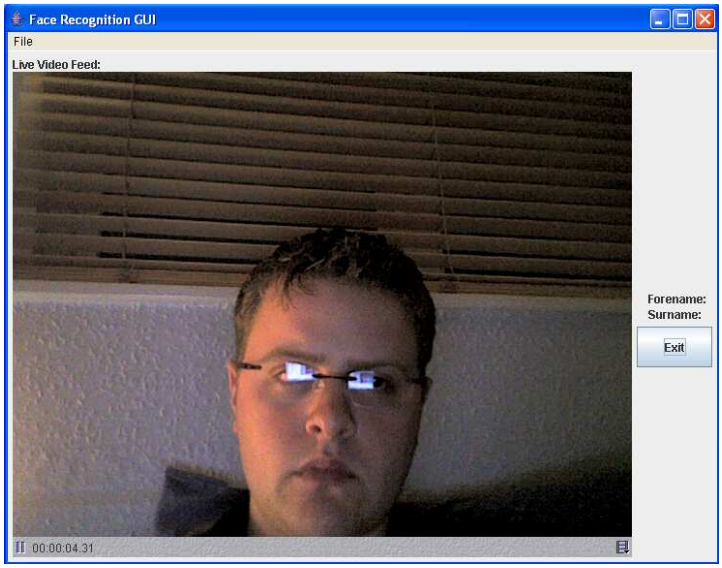

Figure 8.2-1 - Main Interface GUI display

8.2.2 Add Reference Image to database window

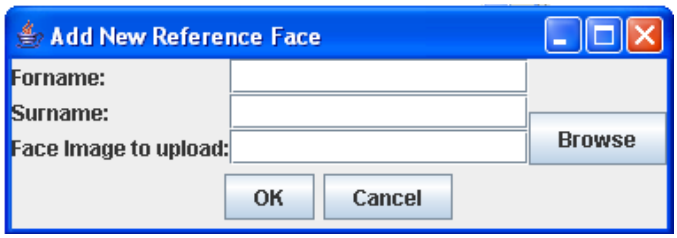

Figure 8.2-2 - Add new reference image to database GUI

8.2.3 Remove Reference Image from database window

Figure 8.2-3 - Remove existing image from reference database GUI

8.3 Appendix C – Code Listing

8.3.1 Training Eye Feature Detector (MATLAB)

The below program illustrates the use of Matlab to train the eye features for use in the feature localisation program in Java. Note; the below program shows the detection and model building of the 'right' eye, an identical program exists for the extraction of the left eye.

```
addpath(genpath('\\smb.csunix.leeds.ac.uk\vislib\dch_matlab\lib'));
addpath(genpath('Z:\work\Third Year\PD32'));
load leedsfyp_feret.mat;
n = length(FIMGS);
eyePatchesRight = [];
% K equals regularisation parameter
k =1;
patchSize = 16;
dim3 = 20;
for i=1:n
    Image = FIMGS(:,:,i);
    eye1 = P(:,1,i);
    eye2 = P(:,2,i);
    eye1(1,:) = eye1(2,:);% - dim3/2;
    eye1(2,:) = eye1(1,:) - dim3/2;
    eye2(1,:) = eye2(2,:);% - dim3/2;
    eye2(2,:) = eye2(1,:) - dim3/2;
    eye1(3,:) = dim3;
    eye2(3,:) = dim3;
    eyePatch1 = newextractpatches(Image,eye1,patchSize);
    eyePatch2 = newextractpatches(Image,eye2,patchSize);
    X = normalise(eyePatch1);
    Y = reshape(X, patchSize*patchSize,1);
    W = normalise(eyePatch2);
    Z = reshape(W, patchSize*patchSize,1);
    eyePatchesRight = [eyePatchesRight, Y];
end
numEyesRight    = length(eyePatchesRight);
```

```
M               = mean(eyePatchesRight,2);
M
CXRight = eyePatchesRight - repmat(M,1,numEyesRight);
SRight  = CXRight * CXRight' / numEyesRight;
regularisationParamRight = k*eye(size(SRight));
SRight = SRight + regularisationParamRight;
ARight = mkmn(M,SRight);
ARight
M  = ARight.M;
SinvRight = ARight.icovariance;
sRight  = ARight.scaling;
save rightEyeMean.dat M -ascii;
save SinvRight.dat SinvRight -ascii;
save scaleFactor.dat sRight -ascii;
```

Figure 8.3-1- getMeanEyeRight(Left).m Training Module

8.3.2 Face.java class source code

```
import java.awt.*;

public class Face {
	private String forename;
	private String surname;
	private double eyeXPos1;
	private double eyeYPos1;
	private double eyeXPos2;
	private double eyeYPos2;
	private Image staticImage;
	private String faceId;

	public Face()
	{

	}
	public Face(String fore, String sur, String eye1X, String eye1Y, String eye2X, String
eye2Y, String Id, Image image)
	{
		forename = fore;
		surname  = sur;
		eyeXPos1 = Double.valueOf(eye1X);
		eyeYPos1 = Double.valueOf(eye1Y);
		eyeXPos2 = Double.valueOf(eye2X);
		eyeYPos2 = Double.valueOf(eye2Y);
		staticImage = image;
		faceId   = Id;
	}
	public void setFore(String fore)
	{
		forename = fore;
	}
	public void setSur(String sur)
	{
		surname = sur;
	}
	public void setEyeX1(String eye)
	{
		eyeXPos1 = Double.valueOf(eye);
	}
	public void setEyeY1(String eye)
	{
		eyeYPos1 = Double.valueOf(eye);
	}
	public void setEyeX2(String eye)
	{
		eyeXPos2 = Double.valueOf(eye);
	}
	public void setEyeY2(String eye)
	{
		eyeYPos2 = Double.valueOf(eye);
	}
	public void setId(String Id)
	{
		faceId = Id;
	}
	public void setImage(Image image)
	{
		staticImage = image;
	}
	public String getForename()
	{
		return forename;
	}
	public String getSurname()
	{
		return surname;
	}
```

```
69          public Double getEye1X()
70          {
71                  return eyeXPos1;
72          }
73          public Double getEye1Y()
74          {
75                  return eyeYPos1;
76          }
77          public Double getEye2X()
78          {
79                  return eyeXPos2;
80          }
81          public Double getEye2Y()
82          {
83                  return eyeYPos2;
84          }
85          public String getId()
86          {
87                  return faceId;
88          }
89          public Image getFaceImage()
90          {
91                  return staticImage;
92          }
93  }
```

Figure 8.3-2 - Face.java class

8.3.3 FaceCrop.java class source code

```
1  import java.awt.Image;
2  import java.awt.image.CropImageFilter;
3  import java.awt.image.FilteredImageSource;
4  import java.awt.*;
5  import java.awt.image.BufferedImage;
6
7  public class FaceCrop
8  {
9          Image croppedImage;
10         BufferedImage croppedBugImg;
11
12         public FaceCrop(Image image, int x, int y, int wh)
13         {
14                 Toolkit tk = Toolkit.getDefaultToolkit();
15                 Image orginalImage = tk.createImage(new FilteredImageSource(image.getSource(), new
16 CropImageFilter(x, y, wh, wh)));
17                 croppedImage = orginalImage;
18         }
19         public FaceCrop(BufferedImage buf, int x, int y, int wh)
20         {
21                 croppedBugImg = buf.getSubimage(x, y, wh, wh);
22         }
23         public Image getCroppedImage()
24         {
25                 return croppedImage;
26         }
27         public BufferedImage getCroppedBufImg()
28         {
29                 return croppedBugImg;
30         }
}
```

Figure 8.3-3 - FaceCrop.java class

8.3.4 EyePatch.java class source code

```
1  public class EyePatch {
2          private int xPos;
3          private int yPos;
4          private double logLiklihood;
5          private int patchSize;
6          private int indexVal;
7
8          public EyePatch(int index,int x, int y, double logLike, int patch)
9          {
10                 xPos = x;
11                 yPos = y;
12                 logLiklihood = logLike;
13                 patchSize = patch;
14                 indexVal = index;
15         }
16         public int getXPos()
17         {
18                 return xPos;
19         }
20         public int getYPos()
21         {
22                 return yPos;
23         }
24         public double getLogLikehoodVal()
25         {
26                 return logLiklihood;
27         }
28         public int getIndex()
29         {
30                 return indexVal;
31         }
32 }
```

Figure 8.3-4 - EyePatch.java class

8.3.5 Matrix.java class source code

```
package FaceRecogPackage;
/*************************************************************************
 *  Compilation:  javac Matrix.java
 *  Execution:    java Matrix
 *  Author:             Robert Sedgewick and Kevin Wayne,
http://www.cs.princeton.edu/introcs/95linear/Matrix.java.html
 *  Modified by:  Martyn Booth (scs3mwb)
 *
 *  A class providing basic matrix calculations sed by some of the other classes
 *
 *************************************************************************/

final public class Matrix {
    private final int M;             // number of rows
    private final int N;             // number of columns
    private final double[][] data;   // M-by-N array

    // create M-by-N matrix of 0's
    public Matrix(int M, int N) {
        this.M = M;
        this.N = N;
        data = new double[M][N];
    }

    // create matrix based on 2d array
    public Matrix(double[][] data) {
        M = data.length;
        N = data[0].length;
        this.data = new double[M][N];
        for (int i = 0; i < M; i++)
            for (int j = 0; j < N; j++)
                    this.data[i][j] = data[i][j];
    }

    // copy constructor
    private Matrix(Matrix A) { this(A.data); }

    // create and return a random M-by-N matrix with values between 0 and 1
    public static Matrix random(int M, int N) {
        Matrix A = new Matrix(M, N);
        for (int i = 0; i < M; i++)
            for (int j = 0; j < N; j++)
                A.data[i][j] = Math.random();
        return A;
    }
    public static double sumSquaredDiffVecs(double[] vector1, double[] vector2)
    {
        double differenceValue = 0.0;
        for(int i=0;i<vector1.length;i++)
        {
                double tempVal = vector2[i] - vector1[i];
                differenceValue += tempVal * tempVal;
        }
        return differenceValue;
    }
    public static double[] columnByMatrixMultiply(Matrix m, double[] columnVector)
    {
        double[] newColumn = new double[columnVector.length];
        for(int i =0; i < m.data[0].length;i++)
        {
                double tempValue = 0.0;
                for(int j=0;j<m.data[0].length;j++)
                {
                        tempValue += m.data[i][j] * columnVector[i];
                        newColumn[i] = tempValue;
                }
```

```
        }
        return newColumn;
    }
    public static double colVecMultRowVec(double[] rowVector, double[] columnVector)
    {
        double result = 0.0;
        for(int i=0;i<rowVector.length;i++)
        {
                result += rowVector[i] * columnVector[i];
        }
        return result;
    }

    // create and return the transpose of the invoking matrix
    public Matrix transpose() {
        Matrix A = new Matrix(N, M);
        for (int i = 0; i < M; i++)
            for (int j = 0; j < N; j++)
                A.data[j][i] = this.data[i][j];
        return A;
    }

    // return C = A * B
    public Matrix times(Matrix B) {
        Matrix A = this;
        if (A.N != B.M) throw new RuntimeException("Illegal matrix dimensions.");
        Matrix C = new Matrix(A.M, B.N);
        for (int i = 0; i < C.M; i++)
            for (int j = 0; j < C.N; j++)
                for (int k = 0; k < A.N; k++)
                    C.data[i][j] += (A.data[i][k] * B.data[k][j]);
        return C;
    }

    // print matrix to standard output
    public void show() {
        for (int i = 0; i < M; i++) {
            for (int j = 0; j < N; j++)
                System.out.printf("%9.4f ", data[i][j]);
            System.out.println();
        }
    }



    // test client
    public static void main(String[] args) {
        double[][] d = { { 1, 2, 3 }, { 4, 5, 6 }, { 9, 1, 3} };
        Matrix D = new Matrix(d);
        D.show();
        System.out.println();

        Matrix A = Matrix.random(5, 5);
        A.show();
        System.out.println();

        Matrix B = A.transpose();
        B.show();
        System.out.println();

        B.times(A).show();
        System.out.println();

        Matrix b = Matrix.random(5, 1);
        b.show();
        System.out.println();

    }}
```

Figure 8.3-5 - Matrix.java class

8.3.6 OpenFileAction.java class source code

```
import javax.swing.*;
import java.io.*;
import java.awt.event.*;

// This action creates and shows a modal open-file dialog.
    public class OpenFileAction extends AbstractAction {
        JFrame frame;
        JFileChooser chooser;

        OpenFileAction(JFrame frame, JFileChooser chooser) {
            super("Open...");
            this.chooser = chooser;
            this.frame = frame;
        }

        public void actionPerformed(ActionEvent evt) {
            // Show dialog; this method does not return until dialog is closed
            chooser.showOpenDialog(frame);

            // Get the selected file
            File file = chooser.getSelectedFile();
        }
    };
```

Figure 8.3-6 - OpenFileAction.java class

8.3.7 FaceLocation.java class source code

```
public class FaceLocation {

        private int xPosition;
        private int yPosition;
        private int recWidth;

        public FaceLocation(int x, int y, int width)
        {
                xPosition = x;
                yPosition = y;
                recWidth  = width;
        }
        public int getX()
        {
                return xPosition;
        }
        public int getY()
        {
                return yPosition;
        }
        public int getWidth()
        {
                return recWidth; }
}
```

Figure 8.3-7 - FaceLocation.java class

8.3.8 FaceLocations.java class source code

```
import java.util.*;

public class FaceLocations {

	private FaceDetection detect;
	private int[] faceArray;
	private int numFaces = 0;
	private ArrayList faceMatrix;

	public FaceLocations()
	{

	}
	public FaceLocations(String file, int cascade, int height, int width)
	{
		faceMatrix = new ArrayList();
		String fileNameSlash = file;
		String fileName;
		if(fileNameSlash.indexOf("\\") < 0)
		{
			fileName = "trainPhotos/" + file;
		}
		else
		{
			fileName = file;
		}

		createFaceSpace(fileName, cascade, height, width);
	}
	public void createFaceSpace(String file, int cascade, int height, int width)
	{
		detect = new FaceDetection(file, 1, 40, 40);
		faceArray = detect.getFaceMatrix();
		numFaces = faceArray.length / 4;
		splitManySpaces(faceArray);
	}
	public void splitManySpaces(int[] faces)
	{
		for (int i = 0; i < numFaces; i++)
		{
			int x = 0;
			int y = 0;
			int w = 0;
		x = faceArray[4 * i + 0];
		y = faceArray[4 * i + 1];
		w = faceArray[4 * i + 2];
		FaceLocation face = new FaceLocation(x,y,w);
		faceMatrix.add(face);
		// System.out.println("1: "+ x + "  2: " + y + "  3: "+ w);
	}
	}
	public ArrayList getFaceObjects()
	{
		return faceMatrix;
	}
	public static void main(String args[])
	{
		FaceLocations myFaceDetection = new FaceLocations();
	}
}
```

Figure 8.3-8 - FaceLocations.java class

8.3.9 *FaceDataAccessLayer.java class source code*

```
import java.awt.Toolkit;
import java.sql.*;
import java.util.*;
import java.io.*;
import java.awt.*;

public class FaceDataAccessLayer {
        String connectionUrl = "jdbc:sqlserver://localhost:1433;" +
        "databaseName=FaceRecognition;user=FaceUser;password=Password1";
        Statement stmt = null;
        ResultSet rs = null;
        Connection conn;
        ArrayList<Face> faces;

        public FaceDataAccessLayer()
        {
                try
                {
                        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
                        conn = DriverManager.getConnection(connectionUrl);
                }
                catch(Exception e)
                {
                        System.out.println(e.getMessage());
                        System.out.println("Connection error");
                }

        }
        public ArrayList<Face> getFaceArray()
        {
                return faces;
        }
        public void insertFaceIntoDatabase(String name, String sur, int eye1X, int eye1Y, int
eye2X, int eye2Y, String faceIm)
        {
                 int len;
          String query;
          PreparedStatement pstmt;

          try
          {
                  File file = new File(faceIm);
                  FileInputStream fis = new FileInputStream(file);
                  len = (int)file.length();

                  query = ("insert into Face (Forename,Surname,eyeOneX,eyeOneY,eyeTwoX,eyeTwoY,
FaceImage, FileName, FileSize) VALUES(?,?,?,?,?,?,?,?,?)");
                  pstmt = conn.prepareStatement(query);
                  pstmt.setString(1,name);
                  pstmt.setString(2,sur);
                  pstmt.setString(3,String.valueOf(eye1X));
                  pstmt.setString(4,String.valueOf(eye1Y));
                  pstmt.setString(5,String.valueOf(eye2X));
                  pstmt.setString(6,String.valueOf(eye2Y));
                  pstmt.setString(8,file.getName());
                  pstmt.setInt(9, len);

                  // Method used to insert a stream of bytes
                  pstmt.setBinaryStream(7, fis, len);
                  pstmt.executeUpdate();

          }
          catch (Exception e)
          {
                  e.printStackTrace();
```

```
          }
        }
        public void deleteFaceById(String Id)
        {
          String query = "DELETE FROM Face WHERE FaceId = ";

          try
          {
                 stmt = conn.createStatement();
                 query += Id;
                 System.out.println(query);
              stmt.executeQuery(query);
              conn.close();
          }
          catch (Exception e)
          {
                  e.printStackTrace();
          }

        }
        public void getFacesFromDatabase()
        {
                String SQL = "SELECT * FROM Face";
                faces = new ArrayList();
                try
                {
                        stmt = conn.createStatement();
                        rs   = stmt.executeQuery(SQL);
                        while (rs.next())
                        {

        //System.out.println(rs.getString(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.ge
tString(4)+"\t"+rs.getString(5)+"\t"+rs.getString(6)+"\t"+rs.getString(7));
                                Face newFace = new Face();
                                newFace.setId(rs.getString(1));
                                newFace.setFore(rs.getString(2));
                                newFace.setSur(rs.getString(3));
                                newFace.setEyeX1(rs.getString(4));
                                newFace.setEyeY1(rs.getString(5));
                                newFace.setEyeX2(rs.getString(6));
                                newFace.setEyeY2(rs.getString(7));
                                //Turn the image capture back on once the image insert is ready

                                Toolkit toolkit = Toolkit.getDefaultToolkit();
                                byte[] imageBinData;
                                imageBinData = rs.getBytes(8);
                                Image tempIm = toolkit.createImage(imageBinData);
                                newFace.setImage(tempIm);
                                faces.add(newFace);
                        }
                        //conn.close();
                }
                catch(Exception e)
                {
                        System.out.println(e.getMessage());
                        System.out.println("Get Faces statement failed");
                }


        }
        public static void main(String[] args)
        {
                FaceDataAccessLayer dal = new FaceDataAccessLayer();
        }
}
```

Figure 8.3-9 - FaceDataAccessLayer.java class

8.3.10 FaceAddToDatabase.java class source code

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class FaceAddToDatabase extends Frame
{
        private JTextField   filePath  = new JTextField(15);
    private JFileChooser _fileChooser = new JFileChooser();
    private JFrame frame;
        private JTextField forenameBox;
        private JTextField surnameBox;
        private Image image;

        public FaceAddToDatabase()
        {
                frame = new JFrame("Add New Reference Face");
         frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
                //frame.setSize(width, height);
                frame.setContentPane(this.createContentPane());
                frame.setVisible(true);
                frame.pack();
                frame.setLocationRelativeTo(null);


        }
        public Container createContentPane() {
                JLabel forename = new JLabel();
                forename.setSize(100, 25);
                forename.setText("Forname:");
                JLabel surname = new JLabel();
                surname.setSize(100, 25);
                surname.setText("Surname:");
                JLabel image = new JLabel();
                image.setSize(100,25);
                image.setText("Face Image to upload:");

                forenameBox = new JTextField();
                forename.setLabelFor(forenameBox);
                //forenameBox.setSize(100,25);
                forenameBox.setColumns(15);
                surnameBox = new JTextField();
                surname.setLabelFor(surnameBox);
                //surnameBox.setSize(100,25);
                surnameBox.setColumns(15);
         //Create the content-pane-to-be.
                //JTextField filePath = new JTextField();
                image.setLabelFor(filePath);
                //filePath.setColumns(15);
                JButton imageBrowse = new JButton();
                imageBrowse.addActionListener(new OpenAction());
                imageBrowse.setText("Browse");
         JPanel contentPane = new JPanel(new BorderLayout());
         contentPane.setOpaque(true);
         JPanel labelPanel = new JPanel(new GridLayout(0,1));
         labelPanel.add(forename);
         labelPanel.add(surname);
         labelPanel.add(image);
         JPanel textPanel  = new JPanel(new GridLayout(0,1));
         textPanel.add(forenameBox);
         textPanel.add(surnameBox);
         textPanel.add(filePath);
         JPanel right      = new JPanel(new GridLayout(0,1));
         JLabel nullLabel = new JLabel();
         nullLabel.setSize(100,25);
         nullLabel.setText(" ");
         JLabel nullLabel2 = new JLabel();
         nullLabel2.setSize(100,25);
         nullLabel2.setText(" ");
```

```
72          right.add(nullLabel);
73          //right.add(nullLabel2);
74          right.add(imageBrowse);
75          JPanel options = new JPanel(new FlowLayout());
76          JButton submit = new JButton("OK");
77          submit.addActionListener(new SubmitFaceToDB());
78          JButton cancel = new JButton("Cancel");
79          cancel.addActionListener(new HideAddFace());
80          options.add(submit);
81          options.add(cancel);
82          contentPane.add(labelPanel,BorderLayout.WEST);
83          contentPane.add(textPanel,BorderLayout.CENTER);
84          contentPane.add(right,BorderLayout.EAST);
85          contentPane.add(options,BorderLayout.SOUTH);
86
87          return contentPane;
88      }
89          ////////////////////////////////////////////////// OpenAction
90          class OpenAction implements ActionListener {
91              public void actionPerformed(ActionEvent ae) {
92                  //... Open a file dialog.
93                  int retval = _fileChooser.showOpenDialog(FaceAddToDatabase.this);
94                  if (retval == JFileChooser.APPROVE_OPTION) {
95                      //... The user selected a file, get it, use it.
96                      File file = _fileChooser.getSelectedFile();
97
98                      //... Update user interface.
99                      filePath.setText(file.getPath());
100                 }
101             }
102         }
103         private class HideAddFace extends AbstractAction
104         {
105                 public HideAddFace()
106                 {
107                 super("Disable frame");
108                 }
109
110                 public void actionPerformed(ActionEvent e)
111                 {
112                 frame.setVisible(false);
113                 }
114         }
115         private class SubmitFaceToDB extends AbstractAction
116         {
117                 public SubmitFaceToDB()
118                 {
119                 }
120                 public void actionPerformed(ActionEvent e)
121                 {
122                         String forenameVal = forenameBox.getText();
123                         String surnameVal  = surnameBox.getText();
124                         //System.out.println(forenameVal);
125                         //System.out.println(surnameVal);
126                         String filePathVal = filePath.getText();
127                         Toolkit toolkit = Toolkit.getDefaultToolkit();
128                         String file = filePathVal;
129                         Image faceImage = toolkit.getImage(file);
130                         FaceLocations locations = new FaceLocations(file,1,40,40);
131                         ArrayList<FaceLocation> faces = new ArrayList<FaceLocation>();
132                         faces = locations.getFaceObjects();
133                         FaceLocation face = (FaceLocation)faces.get(0);
134                         int x = face.getX();
135                         int y = face.getY();
136                         int w = face.getWidth();
137                         FaceCrop crop = new FaceCrop(image,x,y,w);
138                         FeatureLocalisation feature = new
139 FeatureLocalisation(crop.getCroppedImage(),file,16);
140                         int eye1X = feature.getXPos1();
141                         int eye1Y = feature.getYPos1();
142                         int eye2X = feature.getXPos2();
```

```
143                         int eye2Y = feature.getYPos2();
144                         /*BufferedImage buffFace = FeatureLocalisation.toBufferedImage(faceImage);
145                         buffFace = buffFace.getSubimage(x, y, w, w);*/
146                         //filePathVal = filePathVal.replace("\\", "/");
147                         //System.out.print(filePath.getText());
148                         //Image img = Toolkit.getDefaultToolkit().getImage(filePathVal);
149                         //String file = filePath.getText();
150                         //System.out.println(filePathVal);
151                         //image =
152 toolkit.getImage("http://www.particle.kth.se/~lindsey/JavaCourse/Book/Part1/Java/Chapter06/javalo
153 go52x88.gif");//filePath.getText());
154                         FaceDataAccessLayer dal = new FaceDataAccessLayer();
155                         /*System.out.println(img.getHeight(null));
156                         System.out.println(img.getWidth(null));*/
157                         dal.insertFaceIntoDatabase(forenameVal, surnameVal, eye1X, eye1Y, eye2X,
158 eye2Y, filePathVal);
159                         forenameBox.setText("");
160                         surnameBox.setText("");
161                         filePath.setText("");
162                 }
163         }
164 }
```

Figure 8.3-10 - FaceAddToDatabase.java class

8.3.11 FaceRemoveFromDatabase.java class source code

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

class FaceRemoveFromDatabase extends Frame
{
    private JFrame frame;
    private JList list;
    private FaceDataAccessLayer dal;

        public FaceRemoveFromDatabase()
        {
                dal = new FaceDataAccessLayer();
                frame = new JFrame("Remove a Reference Face");
         frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
                //frame.setSize(width, height);
                frame.setContentPane(this.createContentPane());
                frame.setVisible(true);
                frame.pack();
                frame.setLocationRelativeTo(null);
        }
        public Container createContentPane()
        {
         JPanel contentPane = new JPanel(new GridLayout(0,1));
         JPanel jListPane = new JPanel (new FlowLayout());
         contentPane.setOpaque(true);
         dal.getFacesFromDatabase();
         ArrayList<Face> faces = dal.getFaceArray();
         String[] faceDetails = new String[faces.size()];
         for(int i = 0;i<faces.size();i++)
         {
                String tempString;
                Face face = faces.get(i);
                String forename = face.getForename();
                String surname  = face.getSurname();
                //String id           = face.getId();
                tempString          = forename+"\t"+surname;
                faceDetails[i] = tempString;
         }
         list = new JList(faceDetails);
         jListPane.add(list);
         JPanel buttonPane = new JPanel(new FlowLayout());
         JButton submit = new JButton("OK");
         submit.addActionListener(new RemoveFace());
         JButton cancel = new JButton("Cancel");
         buttonPane.add(submit);
         buttonPane.add(cancel);
         cancel.addActionListener(new HideRemFace());
         contentPane.add(jListPane);
         contentPane.add(buttonPane);

         return contentPane;
    }
        private class RemoveFace extends AbstractAction
        {
                public RemoveFace()
                {

                }
                public void actionPerformed(ActionEvent e)
                {
                        //FaceDataAccessLayer dataAL = new FaceDataAccessLayer();
                        int currentItem = list.getSelectedIndex();
                        dal.getFacesFromDatabase();
                 ArrayList<Face> facesArr = dal.getFaceArray();
```

```
67                  String currentId = facesArr.get(currentItem).getId();
68                  //System.out.println(currentId);
69                  dal.deleteFaceById(currentId);
70                  frame.hide();
71                  }
72          }
73          private class HideRemFace extends AbstractAction
74          {
75                  public HideRemFace()
76                  {
77                          super("Disable frame");
78                  }
79
80                  public void actionPerformed(ActionEvent e)
81                  {
82                          frame.setVisible(false);
83                  }
84          }
85  }
```

Figure 8.3-11 - FaceRemoveFromDatabase.java class

8.3.12 FrameAccess.java class source code

```
import java.awt.*;
import javax.media.*;
import javax.media.control.TrackControl;
import javax.media.Format;
import javax.media.format.*;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.media.util.*;
import java.io.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

public class FrameAccess extends JFrame implements ControllerListener {

		Processor p;
   Object waitSync = new Object();
   boolean stateTransitionOK = true;
   private JPanel panel;
   private Component cc;
		private Component vc;
		private int counter=0;
		private int frameThrottlingCounter=0;

   public FrameAccess()
   {

   }
   public void addVideoToPanel(String urlIn)
   {
		panel = new JPanel(new BorderLayout());
		panel.setOpaque(true);
		String url;
		if(urlIn.equalsIgnoreCase(""))
		{
			url = "vfw:Microsoft WDM Image Capture (Win32):0";
		}
		else
		{
			url = urlIn;
		}

		MediaLocator ml;

		if ((ml = new MediaLocator(url)) == null)
		{
		   System.err.println("Cannot build media locator from: " + url);
		   System.exit(0);
		}
		open(ml);
		if ((vc = p.getVisualComponent()) != null)
		{
		   panel.add(vc,BorderLayout.CENTER);
		}

		if ((cc = p.getControlPanelComponent()) != null)
		{
		   panel.add(cc,BorderLayout.SOUTH);
		}
		p.start();

		panel.setOpaque(true);
		panel.setVisible(true);

     //panel.add(scrollPane, BorderLayout.CENTER);
```

```
65      }
66      public JPanel getPanel()
67      {
68              return panel;
69      }
70      /**
71       * Given a media locator, create a processor and use that processor
72       * as a player to playback the media.
73       *
74       * During the processor's Configured state, two "pass-thru" codecs,
75       * PreAccessCodec and PostAccessCodec, are set on the video track.
76       * These codecs are used to get access to individual video frames
77       * of the media.
78       *
79       * Much of the code is just standard code to present media in JMF.
80       */
81      public boolean open(MediaLocator ml) {
82
83              try
84              {
85                      p = Manager.createProcessor(ml);
86              }
87              catch (Exception e)
88              {
89                      System.err.println("Failed to create a processor from the given url: " + e);
90                      return false;
91              }
92
93              p.addControllerListener(this);
94
95              // Put the Processor into configured state.
96              p.configure();
97              if (!waitForState(p.Configured))
98              {
99                      System.err.println("Failed to configure the processor.");
100                     return false;
101             }
102
103             // So I can use it as a player.
104             p.setContentDescriptor(null);
105
106             // Obtain the track controls.
107             TrackControl tc[] = p.getTrackControls();
108
109             if (tc == null)
110             {
111                     System.err.println("Failed to obtain track controls from the processor.");
112                     return false;
113             }
114
115             // Search for the track control for the video track.
116             TrackControl videoTrack = null;
117
118             for (int i = 0; i < tc.length; i++)
119             {
120                     if (tc[i].getFormat() instanceof VideoFormat)
121                     {
122                             videoTrack = tc[i];
123                             break;
124                     }
125             }
126
127             if (videoTrack == null)
128             {
129                     System.err.println("The input media does not contain a video track.");
130                     return false;
131             }
132
133             System.err.println("Video format: " + videoTrack.getFormat());
134
```

```
135             // Instantiate and set the frame access codec to the data flow path.
136             try
137             {
138                     Codec codec[] =
139                     {
140                             new PreAccessCodec(), new PostAccessCodec()};
141                             videoTrack.setCodecChain(codec);
142                     }
143             catch (UnsupportedPlugInException e)
144             {
145                     System.err.println("The process does not support effects.");
146             }
147
148             // Realize the processor.
149             p.prefetch();
150             if (!waitForState(p.Prefetched))
151             {
152                     System.err.println("Failed to realize the processor.");
153                     return false;
154             }
155
156             // Display the visual & control component if there's one.
157
158             /*setLayout(new BorderLayout());
159
160
161             if ((vc = p.getVisualComponent()) != null) {
162                add("Center", vc);
163                /*panel*///add(vc,BorderLayout.CENTER);
164             /*}
165
166             if ((cc = p.getControlPanelComponent()) != null) {
167                add("South", cc);
168                     /*panel.*///add(cc,BorderLayout.SOUTH);
169             //}
170
171             // Start the processor.
172             //p.start();
173
174             //panel.setOpaque(true);
175             //setVisible(true);
176
177
178             return true;
179     }
180
181     public void addNotify() {
182             super.addNotify();
183             pack();
184       }
185
186       /**
187        * Block until the processor has transitioned to the given state.
188        * Return false if the transition failed.
189        */
190       boolean waitForState(int state) {
191             synchronized (waitSync) {
192                try {
193                     while (p.getState() != state && stateTransitionOK)
194                        waitSync.wait();
195                } catch (Exception e) {}
196             }
197             return stateTransitionOK;
198       }
199
200
201       /**
202        * Controller Listener.
203        */
204       public void controllerUpdate(ControllerEvent evt) {
```

```
205
206             if (evt instanceof ConfigureCompleteEvent ||
207                 evt instanceof RealizeCompleteEvent ||
208                 evt instanceof PrefetchCompleteEvent) {
209                 synchronized (waitSync) {
210                         stateTransitionOK = true;
211                         waitSync.notifyAll();
212                 }
213             } else if (evt instanceof ResourceUnavailableEvent) {
214                 synchronized (waitSync) {
215                         stateTransitionOK = false;
216                         waitSync.notifyAll();
217                 }
218             } else if (evt instanceof EndOfMediaEvent) {
219                 p.close();
220                 System.exit(0);
221             }
222         }
223
224
225         /**
226          * Main program
227          */
228         public static void main(String [] args)
229         {
230
231             if (args.length < 0)
232             {
233                         prUsage();
234                         System.exit(0);
235             }
236
237             String url = "vfw:Microsoft WDM Image Capture (Win32):0";//args[0];
238
239             if (url.indexOf(":") < 0)
240             {
241                         prUsage();
242                         System.exit(0);
243             }
244
245             MediaLocator ml;
246
247             if ((ml = new MediaLocator(url)) == null)
248             {
249                         System.err.println("Cannot build media locator from: " + url);
250                         System.exit(0);
251             }
252
253             FrameAccess fa = new FrameAccess();
254
255             if (!fa.open(ml))
256                         System.exit(0);
257         }
258
259         static void prUsage()
260         {
261             System.err.println("Usage: java FrameAccess <url>");
262         }
263
264
265
266
267
268         /**************************************************************
269          * Inner class.
270          *
271          * A pass-through codec to access to individual frames.
272          *************************************************************/
273
274             public class PreAccessCodec implements Codec {
```

```
    /**
* Callback to access individual video frames.
*/
    void accessFrame(Buffer frame)
    {

                // For demo, we'll just print out the frame #, time &
                // data length.

                long t = (long)(frame.getTimeStamp()/10000000f);

                System.err.println("Pre: frame #: " + frame.getSequenceNumber() +
                        ", time: " + ((float)t)/100f +
                        ", len: " + frame.getLength());
                if(frameThrottlingCounter % 10 == 0)
                {
                        BufferToImage stopBuffer = new BufferToImage((VideoFormat)frame.getFormat());
                        Image stopImage = stopBuffer.createImage(frame);
                        //File fOut = new File("VideoFrameBuffer/"+counter+".jpg");
                        counter++;
                        try {
                                            Thread.sleep(400);
                                            //1000 millisecs
                                        }
                                        catch(InterruptedException ex) {
                                        }
                        try
                                {
                                //Possibly no video being obtained here
                                System.out.println("Create buffered image");
                                BufferedImage outImage = new BufferedImage(640,480,BufferedImage.TYPE_INT_RGB);
                                //FrameHandler.addImageToQueue(outImage);
                                //if(outImage!=null)
                                //System.out.println("hi4");
                                System.out.println("Get graphics");
                                Graphics og = outImage.getGraphics();
                                //FrameHandler.addImageToQueue(outImage);
                                System.out.println("write image to Image");
                                og.drawImage(stopImage, 0, 0, 640, 480, null);
                                //System.out.println("Width: "+outImage.getWidth()+"Height: "+outImage.getHeight());
                                System.out.println("Add image to stack");
                                FrameHandler.addImageToQueue(outImage);
                                System.out.println("image added to queue");
                                FrameHandler.getStackSize();
                                //ImageIO.write(outImage,"jpg",fOut);
                                }
                        catch (Exception ie)
                        {
                                System.out.println("get video frame failed");
                                System.out.println("Error :" + ie);
                                System.out.println(ie.getMessage());
                                System.out.println(ie.getStackTrace());
                        }
                }


                }


                /**
                 * The code for a pass through codec.
                 */

                // We'll advertize as supporting all video formats.
                protected Format supportedIns[] = new Format []
                {
                                new VideoFormat(null)
                };
                public boolean clearFrameBuffer()
```

```
345                     {
346                             boolean buffClear = false;
347                             return buffClear;
348                     }
349
350                     // We'll advertize as supporting all video formats.
351                     protected Format supportedOuts[] = new Format []
352                     {
353                                     new VideoFormat(null)
354                     };
355
356                     Format input = null, output = null;
357
358                     public String getName()
359                     {
360                             return "Pre-Access Codec";
361                     }
362
363                     // No op.
364             public void open()
365             {
366             }
367
368                     // No op.
369                     public void close()
370                     {
371                     }
372
373                     // No op.
374                     public void reset()
375                     {
376                     }
377
378                     public Format [] getSupportedInputFormats()
379                     {
380                             return supportedIns;
381                     }
382
383                     public Format [] getSupportedOutputFormats(Format in)
384                     {
385                             if (in == null)
386                                     return supportedOuts;
387                             else
388                             {
389                             // If an input format is given, we use that input format
390                             // as the output since we are not modifying the bit stream
391                             // at all.
392                             Format outs[] = new Format[1];
393                                     outs[0] = in;
394                                     return outs;
395                     }
396                     }
397
398                     public Format setInputFormat(Format format)
399                     {
400                             input = format;
401                             return input;
402                     }
403
404                     public Format setOutputFormat(Format format)
405                     {
406                             output = format;
407                             return output;
408                     }
409
410                     public int process(Buffer in, Buffer out)
411                     {
412
413                     // This is the "Callback" to access individual frames.
414                     accessFrame(in);
```

```
                        // Swap the data between the input & output.
                        Object data = in.getData();
                        in.setData(out.getData());
                        out.setData(data);

                        // Copy the input attributes to the output
                        out.setFormat(in.getFormat());
                        out.setLength(in.getLength());
                        out.setOffset(in.getOffset());

                        return BUFFER_PROCESSED_OK;
                        }

                        public Object[] getControls()
                        {
                                return new Object[0];
                        }

                        public Object getControl(String type)
                        {
                        return null;
                        }
        }

        public class PostAccessCodec extends PreAccessCodec
        {
                // We'll advertize as supporting all video formats.
                public PostAccessCodec()
                {
                        supportedIns = new Format []
                        {
                                new RGBFormat()
                        };
                }

                 /**
               * Callback to access individual video frames.
               */
                void accessFrame(Buffer frame)
                {

                        // For demo, we'll just print out the frame #, time &
                        // data length.

                        long t = (long)(frame.getTimeStamp()/10000000f);

                        System.err.println("Post: frame #: " + frame.getSequenceNumber() +
                                ", time: " + ((float)t)/100f +
                                ", len: " + frame.getLength());
                }

                public String getName() {
                        return "Post-Access Codec";
                }
        }
}
```

Figure 8.3-12 - FrameAccess.java class

8.3.13 FaceDetection.java class source code

```
class JNIOpenCV {
    static {
        System.loadLibrary("JNI2OpenCV");
    }
    public native int[] detectFace(int minFaceWidth, int minFaceHeight, String cascade, String
filename);
}

public class FaceDetection {
        private JNIOpenCV myJNIOpenCV;
        private FaceDetection myFaceDetection;
        private int[] detectedFaces;

        public FaceDetection(String file, int cascadeXML, int faceHeight, int faceWidth) {
                myJNIOpenCV = new JNIOpenCV();
                String filename = file;
                String cascade = "";
                switch (cascadeXML)
                {
                        case 1:
                                cascade = "Cascades/haarcascade_frontalface_alt.xml";
                                break;
                        case 2:
                                cascade = "Cascades/haarcascade_frontalface_alt1.xml";
                                break;
                        case 3:
                                cascade = "Cascades/haarcascade_frontalface_alt2.xml";
                                break;
                        case 4:
                                cascade = "Cascades/haarcascade_frontalface_alt_tree.xml";
                                break;
                        case 5:
                                cascade = "Cascades/haarcascade_frontalface_default.xml";
                                break;
                        case 6:
                                cascade = "Cascades/haarcascade_fullbody.xml";
                                break;
                        case 7:
                                cascade = "Cascades/haarcascade_lowerbody.xml";
                                break;
                        case 8:
                                cascade = "Cascades/haarcascade_profileface.xml";
                                break;
                        case 9:
                                cascade = "Cascades/haarcascade_upperbody.xml";
                                break;
                        default:
                                cascade = "Cascades/haarcascade_frontalface_alt.xml";
                                break;  }
                detectedFaces = myJNIOpenCV.detectFace(faceHeight, faceWidth, cascade, filename);
                int numFaces = detectedFaces.length / 4;

        System.out.println("numFaces = " + numFaces);
        for (int i = 0; i < numFaces; i++) {
                System.out.println("Face " + i + ": " + detectedFaces[4 * i + 0] + " " +
detectedFaces[4 * i + 1] + " " + detectedFaces[4 * i + 2] + " " + detectedFaces[4 * i + 3]);
        } }
        public int[] getFaceMatrix()
        {
                return detectedFaces; }
    public static void main(String args[]) {
         String fileIs = "C:/Documents and
Settings/Martyn/MyDocuments/Eclipse/FaceRecognitionProj/trainPhotos/lena.jpg";
        FaceDetection myFaceDetection = new FaceDetection(fileIs,40,40,1);

    }
```

```
67  }
```

Figure 8.3-13 - FaceDetection.java class

8.3.14 FeatureLocalisation.java original class source code

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.lang.Math;
import javax.swing.JFrame;

public class FeatureLocalisation extends JFrame
{
	private Image featuresFace;
	private BufferedImage featuresFaceBuff;
	private double[] meanEyeArrayRight;
	private double[] meanEyeArrayLeft;
	private double[][] sinvArrayRight;
	private double[][] sinvArrayLeft;
	private static double sinRight = 5.6276e-103;
	private static double sinLeft  = 5.2953e-103;
	private static double logSinRight = Math.log(sinRight);
	private static double logSinLeft  = Math.log(sinLeft);
	private ArrayList<EyePatch> patchListLeftEye;
	private ArrayList<EyePatch> patchListRightEye;
	private int x1;
	private int y1;
	private int x2;
	private int y2;

	/**
	 * setMeanEyeArray
	 *
	 * File from Matlab containing mean eye vector is read into Java and stored in global
variable patchEyeList Left or Right
	 *
	 * @param filename .dat file containing mean eye values (one per line)
	 * @param array    array to read the values into
	 */
	public int getXPos1()
	{
		return x1;
	}
	public int getYPos1()
	{
		return y1;
	}
	public int getXPos2()
	{
		return x2;
	}
	public int getYPos2()
	{
		return y2;
	}
	public void setMeanEyeArray(String filename, double[] array)
	{
		//array = new double[256];
		try
		{
		BufferedReader in = new BufferedReader(new FileReader("trainData\\"+filename));
		String str;
		int counter = 0;
		while ((str = in.readLine()) != null)
		{
			double d = Double.valueOf(str.trim()).doubleValue();
			array[counter] = d;
			counter++;
		}
		in.close();
```

```
69              }
70                  catch (IOException e)
71                  {
72                          System.out.println(e.getMessage());
73              }
74
75          }
76          public void setSinvArray(String filename, int patchSize, double[][] sinvArray)
77          {
78                  sinvArray = new double[patchSize*patchSize][patchSize*patchSize];
79                  try
80                  {
81                          //int resultLength = 0;
82                  BufferedReader in = new BufferedReader(new FileReader("trainData\\"+filename));
83                  String str;
84                  int counter = 0;
85                  while ((str = in.readLine()) != null)
86                  {
87                          String[] result = str.split(" ");
88                          for(int token=0;token<result.length;token++)
89                          {
90                                  int lineCounter = 0;
91                                  if(!result[token].equals(""))
92                                  {
93                                          //System.out.println(result[token]);
94                                          double d =
95  Double.valueOf(result[token].trim()).doubleValue();
96                                          sinvArray[counter][lineCounter] = d;
97                                          lineCounter++;
98                                  }
99                          }
100                         counter++;
101                 }
102                 //System.out.println(resultLength);
103                 in.close();
104             }
105                 catch (IOException e)
106                 {
107                         System.out.println(e.getMessage());
108             }
109         }
110         public static BufferedImage toBufferedImage(Image image) {
111          if (image instanceof BufferedImage)
112              return (BufferedImage) image;
113          ColorModel cm = getColorModel(image);
114          int width = image.getWidth(null);
115          int height = image.getHeight(null);
116          return copy(createBufferedImage(cm, width, height), image);
117     }
118         public static BufferedImage createBufferedImage(ColorModel cm, int w, int h) {
119                 WritableRaster raster = cm.createCompatibleWritableRaster(w, h);
120                 boolean isRasterPremultiplied = cm.isAlphaPremultiplied();
121                 return new BufferedImage(cm, raster, isRasterPremultiplied, null);
122         }
123         public static BufferedImage copy(BufferedImage target, Image source) {
124                 Graphics2D g = target.createGraphics();
125                 g.drawImage(source, 0, 0, null);
126                 g.dispose();
127                 return target;
128         }
129         public static ColorModel getColorModel(Image image) {
130          try {
131              PixelGrabber pg = new PixelGrabber(image, 0,0,1,1, false);
132              pg.grabPixels();
133              return pg.getColorModel();
134          } catch (InterruptedException e) {
135              throw new RuntimeException("Unexpected interruption", e);
136          }
137     }
138         public double[] compareVectors(double[] meanEyeV, double[] testPatchVector)
139         {
```

```
140                 double[] compareVector = new double[meanEyeV.length];
141                 for(int i=0;i<meanEyeV.length;i++)
142                 {
143                         compareVector[i] = testPatchVector[i] - meanEyeV[i];
144                 }
145                 return compareVector;
146         }
147
148
149         public FeatureLocalisation(Image image, String fileName, int patchSize)
150         {
151                 patchListLeftEye = new ArrayList<EyePatch>();
152                 patchListRightEye= new ArrayList<EyePatch>();
153                 meanEyeArrayLeft = new double[patchSize*patchSize];
154                 meanEyeArrayRight = new double[patchSize*patchSize];
155                 sinvArrayRight = new double[patchSize*patchSize][patchSize*patchSize];
156                 sinvArrayLeft  = new double[patchSize*patchSize][patchSize*patchSize];
157                 setMeanEyeArray("leftEyeMean.dat",meanEyeArrayLeft);
158                 setMeanEyeArray("RightEyeMean.dat",meanEyeArrayRight);
159                 setSinvArray("SinvRight.dat", patchSize, sinvArrayRight);
160                 setSinvArray("SinvLeft.dat", patchSize, sinvArrayLeft);
161                 featuresFace = image;
162                 featuresFace = image.getScaledInstance(100,100,5);
163
164                 Toolkit toolkit = Toolkit.getDefaultToolkit();
165                 String file = fileName;
166                 if (file.indexOf("\\") < 0)
167                 {
168                         file = "trainPhotos/" + fileName;
169                 }
170                 else
171                 {
172                         file = fileName;
173                 }
174                 image = toolkit.getImage(file);
175                 MediaTracker mediaTracker = new MediaTracker(this);
176                 mediaTracker.addImage(featuresFace, 0);
177                 try
178                 {
179                         mediaTracker.waitForID(0);
180                 }
181                 catch (InterruptedException ie)
182                 {
183                         System.err.println(ie);
184                         System.exit(1);
185                 }
186                 addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent e)
187                         {
188                         //System.exit(0);
189                         hide();
190                 }});
191                 setSize(featuresFace.getWidth(null), featuresFace.getHeight(null));
192                 setTitle("Face");
193                 show();
194                 BufferedImage imgToBuf = toBufferedImage(featuresFace);
195                 imageSlidingWindow(imgToBuf,patchSize);
196         }
197         public FeatureLocalisation(BufferedImage image, int patchSize)
198         {
199                 System.out.println("Running feature localisation");
200                 patchListLeftEye = new ArrayList<EyePatch>();
201                 System.out.println("1");
202                 patchListRightEye= new ArrayList<EyePatch>();
203                 System.out.println("2");
204                 meanEyeArrayLeft = new double[patchSize*patchSize];
205                 System.out.println("3");
206                 meanEyeArrayRight = new double[patchSize*patchSize];
207                 System.out.println("4");
208                 sinvArrayRight = new double[patchSize*patchSize][patchSize*patchSize];
209                 System.out.println("5");
210                 sinvArrayLeft  = new double[patchSize*patchSize][patchSize*patchSize];
```

```
                System.out.println("6");
                setMeanEyeArray("leftEyeMean.dat",meanEyeArrayLeft);
                System.out.println("7");
                setMeanEyeArray("RightEyeMean.dat",meanEyeArrayRight);
                System.out.println("8");
                setSinvArray("SinvRight.dat", patchSize, sinvArrayRight);
                System.out.println("9");
                setSinvArray("SinvLeft.dat", patchSize, sinvArrayLeft);
                System.out.println("Arrays setup and running");
                featuresFaceBuff = image;
                //feturesFaceBuff = image.getScaledInstance(100,100,5);
                GraphicsConfiguration gc = getDefaultConfiguration();
                System.out.println("Running sliding window");

                imageSlidingWindow(getScaledInstance(featuresFaceBuff,100,100,null),patchSize);
        }
        public static BufferedImage getScaledInstance(BufferedImage image, int width, int height,
GraphicsConfiguration gc) {
            if (gc == null)
                gc = getDefaultConfiguration();
            int transparency = image.getColorModel().getTransparency();
            return copy(image, gc.createCompatibleImage(width, height, transparency));
        }
        public static GraphicsConfiguration getDefaultConfiguration() {
         GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
         GraphicsDevice gd = ge.getDefaultScreenDevice();
         return gd.getDefaultConfiguration();
    }
        public void paint(Graphics graphics) {
                graphics.drawImage(featuresFace, 0, 0, null);
                getRectanglesAndDraw(graphics);

        }
        public double getGreyScaleFromRGBScaled(int red, int green, int blue)
        {
                double grayValue = 0.0;
                double redValue = 0.3;
                double greenValue = 0.59;
                double blueValue = 0.11;
                grayValue = (redValue*red) + (greenValue*green) + (blueValue*blue);
                return (grayValue/256);
        }
        public double[][] vectorToMatrix(double[] vector)
        {
                int matrixDimension = (int)Math.sqrt(vector.length);
                double[][] reconstructedMatrix = new double[matrixDimension][matrixDimension];
                int columnCounter = 0;
                int rowCounter = 0;
                for(int i=0;i<vector.length;i++)
                {
                        reconstructedMatrix[rowCounter][columnCounter] = vector[i];
                        if(rowCounter == matrixDimension-1)
                        {
                                rowCounter = 0;
                                columnCounter++;
                        }
                        else
                        {
                                rowCounter++;
                        }
                }
                return reconstructedMatrix;
        }
        public void drawRectangles(Graphics graphics)
        {
                getRectanglesAndDraw(graphics);
        }
        public void drawRectangle(Graphics graphic, int x, int y, int z)
        {
                graphic.drawRect(x, y, z, z);
        }
```

```
282         public void getRectanglesAndDraw(Graphics graphicPass)
283         {
284                 drawRectangle(graphicPass, y1, x1, 16);
285         }
286
287
288         // bubbleSort - sorts, in place, the items in the ArrayList 'entries'
289         public void bubbleSort(ArrayList<EyePatch> faceArrayUnsorted)
290         {
291                 boolean swapped=true;
292                 for (int i = faceArrayUnsorted.size()-1; i >= 0 && swapped; i--)
293                 {
294                         swapped = false;
295                         for (int j = 0; j < i; j++)
296                         {
297                          if (faceArrayUnsorted.get(j+1).getLogLikehoodVal() <
298 (faceArrayUnsorted.get(j).getLogLikehoodVal()))
299                          {
300                            EyePatch tmp = faceArrayUnsorted.get(j+1);
301                            faceArrayUnsorted.set(j+1, faceArrayUnsorted.get(j));
302                            faceArrayUnsorted.set(j, tmp);
303                            swapped = true;
304                          }
305                     }
306                 }
307         }
308         public void imageSlidingWindow(BufferedImage bufferedFace, int patchSize)
309         {
310
311                 int height = bufferedFace.getHeight(null);
312                 int width  = bufferedFace.getWidth(null);
313                 int lastValueX = width - patchSize;
314                 System.out.println("Maxiumu X Value of image:\t"+lastValueX);
315                 System.out.println("Running sliding window");
316                 //Value here is divided by two to remove the bottom half of the image from the
317 search space
318                 int startValueY = height/2;//84;//height;
319                 //The last value is in effect close to the top because of the x,y coordinates
320                 int lastValueY  = height-patchSize-1;//(height/2) - patchSize;
321                 //Load face into buffer from disk
322                 int patchIndexCounter = 0;
323                 for(int y=startValueY;y<lastValueY;y++)
324                 {
325                         for(int x=0;x<lastValueX;x++)
326                         {
327                                 //Array of pixels to represent image patch in colour
328                                 int[][] pixels = new int[patchSize][patchSize];
329                                 double[][] grayPixels = new double[patchSize][patchSize];
330                                 for(int row=0;row<patchSize;row++)
331                                 {
332                                 //Extract rows from the image patch and load them into the RGB
333 array above
334                                         bufferedFace.getRGB(x, /*row+(y-patchSize)*/y+row,
335 patchSize, 1, pixels[row], 0, patchSize);
336                                 }
337                                 for(int i=0;i<patchSize;i++)
338                                 {
339                                         //System.out.println();
340                                         for(int j=0;j<patchSize;j++)
341                                         {
342                                                 //int alpha = 0;
343                                                 int red   = 0;
344                                                 int green = 0;
345                                                 int blue  = 0;
346                                                 //alpha = (pixels[i][j]>>24)&255;
347                                                 red   = (pixels[i][j]>>16)&255;
348                                                 green = (pixels[i][j]>>8) &255;
349                                                 blue  =  pixels[i][j]     &255;
350                                                 double grayScaleValue =
351 getGreyScaleFromRGBScaled(red,green,blue);
352                                                 grayPixels[i][j] = grayScaleValue;
```

```
                                        //System.out.println("Grayscale value:
"+grayScaleValue);

                                }
                        }
                        double[] testEyePatchVector = new double[patchSize*patchSize];
                        //Counter to create the vector
                        int vectorCounter = 0;
                        for(int colV =0;colV<patchSize;colV++)
                        {
                                for(int rowV=patchSize-1;rowV>-1;rowV--)
                                {
                                        testEyePatchVector[vectorCounter] =
grayPixels[rowV][colV];
                                        vectorCounter++;
                                }
                        }
                        double[] eyePatchCompVectorRight =
compareVectors(meanEyeArrayRight,testEyePatchVector);
                        double[] eyePatchCompVectorLeft  = compareVectors(meanEyeArrayLeft,
testEyePatchVector);
                        Matrix SinvMatrixRight = new Matrix(sinvArrayRight);
                        Matrix SinvMatrixLeft  = new Matrix(sinvArrayLeft);
                        double[] newColumnRight =
Matrix.columnByMatrixMultiply(SinvMatrixRight,eyePatchCompVectorRight);
                        double[] newColumnLeft  =
Matrix.columnByMatrixMultiply(SinvMatrixLeft, eyePatchCompVectorLeft);
                        double colRowVecRight  =
Matrix.colVecMultRowVec(eyePatchCompVectorRight, newColumnRight);
                        double colRowVecLeft   =
Matrix.colVecMultRowVec(eyePatchCompVectorLeft, newColumnLeft);

                        EyePatch thisPatchLeft = new
EyePatch(patchIndexCounter,x,y,colRowVecRight,patchSize);
                        EyePatch thisPatchRight= new
EyePatch(patchIndexCounter,x,y,colRowVecLeft,patchSize);
                        patchIndexCounter++;
                        patchListLeftEye.add(thisPatchLeft);
                        patchListRightEye.add(thisPatchRight);
                        double[][] compMatrixRight =
vectorToMatrix(eyePatchCompVectorRight);
                        double[][] compMatrixLeft  =
vectorToMatrix(eyePatchCompVectorLeft);
                        Matrix meanCompRight = new Matrix(compMatrixRight);
                        Matrix meanCompLeft  = new Matrix(compMatrixLeft);
                        Matrix compMatrixTransposeRight = meanCompRight.transpose();
                        Matrix compMatrixTransposeLeft  = meanCompLeft.transpose();
                        Matrix meanCompMultSinvRight =
SinvMatrixRight.times(meanCompRight);
                        Matrix meanCompMultSinvLeft  = SinvMatrixLeft.times(meanCompLeft);
                }
        }

        int highIndex = 0;
        int highIndex2= 0;
        double logVal = 100.0;

        int xPos  = 0;
        int yPos  = 0;
        int xPos2 = 0;
        int yPos2 = 0;

        System.out.println("Right eye position");
        logVal = 100.0;
        for(int i=0;i<patchListRightEye.size();i++)
        {
                EyePatch testPatch2 = patchListRightEye.get(i);
                //System.out.println("XPosition of patch:\t"+Math.abs(testPatch2.getXPos()
- x1));
                if(testPatch2.getLogLikehoodVal() < logVal/*
&&(Math.abs(testPatch2.getXPos() - x1) > (patchSize))*/)
```

```
424                         {
425                                 //if(Math.abs(testPatch2.getXPos() - x1) > patchSize)
426                                         {
427                                                 logVal = testPatch2.getLogLikehoodVal();
428                                                 highIndex2 = testPatch2.getIndex();
429                                                 xPos = testPatch2.getXPos();
430                                                 yPos = testPatch2.getYPos();
431                                                 System.out.print(testPatch2.getXPos());
432                                         }
433                         }
434                 }
435                 x1 = xPos;
436                 y1 = yPos;
437                 for(int i=0;i<patchListLeftEye.size();i++)
438                 {
439                         EyePatch testPatch = patchListLeftEye.get(i);
440                         if(testPatch.getLogLikehoodVal() < logVal)
441                         {
442                                 if(Math.abs(testPatch.getXPos() - x1) > patchSize)
443                                 {
444                                         logVal = testPatch.getLogLikehoodVal();
445                                         highIndex = testPatch.getIndex();
446                                         xPos2 = testPatch.getXPos();
447                                         yPos2 = testPatch.getYPos();
448                                         System.out.print(xPos+"\t");
449                                 }
450                         }
451                 }
452                 x2 = xPos2;
453                 y2 = yPos2;
454                 System.out.println("X1: "+x1+"\tY1: "+y1+"\tX2: "+x2+"\tY2: "+y2);
455                 System.out.println("Sliding window applied!");
456         }
457 }
```

Figure 8.3-14 - FeatureLocalisation.java Source code (original)

8.3.15 FrameHandler.java class source code

```
1  import java.awt.FlowLayout;
2  import java.awt.Graphics;
3  import java.awt.GridLayout;
4  import java.awt.MediaTracker;
5  import java.awt.Toolkit;
6  import java.awt.image.BufferedImage;
7  import java.util.ArrayList;
8  import java.io.*;
9  import javax.imageio.ImageIO;
10 import javax.swing.JButton;
11 import javax.swing.JLabel;
12 import javax.swing.JPanel;
13 import javax.swing.ImageIcon;
14
15 import java.awt.event.WindowAdapter;
16 import java.awt.event.WindowEvent;
17 import java.awt.geom.*;
18 import java.util.TimerTask;
19
20
21 /**
22  * FrameHandler.java
23  * @author Martyn Booth
24  * Implements a new thread for that the infinite loop used here does not
25  */
26
27 public class FrameHandler extends Thread//implements Runnable
28 {
29         private static WorkQueue imageQueue;
30         private ArrayList faces;
31         private FaceLocations locations;
32         private int counter = 0;
33         private JPanel matchingDisplay;
34         private int[] rightEyeX;
35         private int[] rightEyeY;
36         private int[] leftEyeX;
37         private int[] leftEyeY;
38         private ArrayList<Face> referenceFaces;
39         private FaceDataAccessLayer dal;
40         private JLabel forenameDisplay;
41    private JLabel surnameDisplay;
42    private JLabel imageIconHost;
43    private JPanel dbImage;
44    private BufferedImage imgFromDB;
45    private static Face currentMatchedFace;
46    private java.awt.Image image;
47
48         public FrameHandler()
49         {
50                 imageIconHost = new JLabel();
51                 imageQueue = new WorkQueue();
52         }
53         public void run()
54         {
```

```
55                      doLiveMatchingScheme();
56              }
57              public static Face getMatchedFaceFace()
58              {
59                      return currentMatchedFace;
60              }
61              public void doLiveMatchingScheme()
62              {
63                      for(;;)
64                      {
65                      dal = new FaceDataAccessLayer();
66                      obtainReferenceFaces();
67                              int numberOfFrames = 5;
68                              rightEyeX = new int[numberOfFrames];
69                              rightEyeY = new int[numberOfFrames];
70                              leftEyeX  = new int[numberOfFrames];
71                              leftEyeY  = new int[numberOfFrames];
72                              FeatureLocalisation feature;
73                              int counter2 = 0;
74                              for(int i = 0;i<numberOfFrames;i++)
75                              {
76                                      String path ="";
77                                      BufferedImage vidFrame = new BufferedImage(20,20,BufferedImage.TYPE_INT_RGB);
78                                      File fOut = new File("Dummy");
79                                      try
80                                      {
81
82                                              vidFrame = (BufferedImage)imageQueue.getWork();
83                                              System.out.println("Got work from queue");
84                                              Graphics og = vidFrame.getGraphics();
85                                              path = "VideoFrameBuffer/"+counter+".jpg";
86
87                                              fOut = new File(path);
88                                              ImageIO.write(vidFrame, "jpg", fOut);
89                                              String newPath = "C:\\Documents and Settings\\Martyn\\My
90  Documents\\Eclipse\\FaceRecognitionProj\\VideoFrameBuffer\\"+counter+".jpg";
91                                              System.out.println("Get Face locations");
92                                              getFaceLocations(newPath);
93                                              System.out.println("intialise array");
94                                              setUpArray();
95                                              System.out.println("Get Features");
96                                              feature = createFacePatchesFromArray(vidFrame,16);
97                                              System.out.println("Get face patches complete");
98                                              rightEyeX[counter2] = feature.getXPos1();
99                                              rightEyeY[counter2] = feature.getYPos1();
100                                             leftEyeX[counter2]  = feature.getXPos2();
101                                             leftEyeY[counter2]  = feature.getYPos2();
102                                             boolean success = fOut.delete();
103                                             if(!success)
104                                             {
105                                                     System.out.println("Temporary file not deleted correctly");
106                                             }
107
108
109                                     }
110                                     catch(Exception e)
111                                     {
112                                             System.out.println("Remove video frame from work queue failed: "+e.getMessage());
```

```
                                        }
                                        System.out.println("Continuing image access");
                                        counter++;

                                }System.out.println("iteration complete");
                                int rightEyeXInt = averageOfArrayValues(rightEyeX);
                                int rightEyeYInt = averageOfArrayValues(rightEyeY);
                                int leftEyeXInt = averageOfArrayValues(leftEyeX);
                                int leftEyeYInt = averageOfArrayValues(leftEyeX);
                                System.out.println("arrays averaged");
                                Point2D testRightEye = new Point2D.Double(rightEyeXInt, rightEyeYInt);
                                Point2D testLeftEye  = new Point2D.Double(leftEyeXInt,  leftEyeYInt);
                                double testImgEucDist = testRightEye.distance(testLeftEye);
                                int matchedFace = 0;
                                for(int i = 0;i<faces.size();i++)
        {
                Face face = (Face)faces.get(i);
                double faceEucDistTest = face.getEuclideanDistance();
                if(Math.abs(testImgEucDist - faceEucDistTest) < Math.abs(testImgEucDist - face.getEuclideanDistance()))
                {
                                matchedFace = i;
                }
        }
                                currentMatchedFace = (Face)faces.get(matchedFace);
                                System.out.println(currentMatchedFace.getForename());
                                Face face = FrameHandler.getMatchedFaceFace();
                FaceRecogGUI.forenameDisplay.setText(face.getForename());
                FaceRecogGUI.surnameDisplay.setText(face.getSurname());
                BufferedImage buffImg = FeatureLocalisation.toBufferedImage(face.getFaceImage());
                FaceRecogGUI.imageLabel = buffImageToImageIcon(buffImg);
                FaceRecogGUI.matchDetails.repaint();

                }
        }
        /**
         * Test method to check on the presence of images from the frame splitter
         * @param fileName name of the image being tested
         */
        public void drawFrame(String fileName)
        {
                javax.swing.JFrame frame = new javax.swing.JFrame("Frame image");
                Toolkit toolkit = Toolkit.getDefaultToolkit();
                String file = /*"trainPhotos/" +*/ fileName;
                file = "C:\\Documents and Settings\\Martyn\\My Documents\\Eclipse\\FaceRecognitionProj\\VideoFrameBuffer\\0.jpg";
                image = toolkit.getImage(file);
                MediaTracker mediaTracker = new MediaTracker(frame);
                mediaTracker.addImage(image, 0);

                try
                {
                                mediaTracker.waitForID(0);
                }
                catch (InterruptedException ie)
                {
                                System.err.println(ie);
                                System.exit(1);
                }
                frame.addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent e)
```

```
                                {
                        System.exit(0);
                        }
                        });

                        frame.setSize(image.getWidth(null), image.getHeight(null));
                        frame.setTitle(fileName);
                        frame.setVisible(true);

        }
        public void paint(Graphics graphics)
        {
                        graphics.drawImage(image, 0, 0, null);
        }
        /**
         * Create the panel for displaying the matching face details and return to main GUI
         * @return JPanel containing relevant matching details
         */
        public JPanel createMatchingDisplay()
{
        //Create the display for showing the face that has been matched to the video
        JPanel matchContainer = new JPanel(new GridLayout(0,1));
        dbImage                                         = new JPanel(new FlowLayout());
        JPanel matchFieldsContainer = new JPanel(new FlowLayout());
        JPanel labelPanel       = new JPanel(new GridLayout(0,1));
        JPanel answerPanel      = new JPanel(new GridLayout(0,1));
        JLabel forenameLabel  = new JLabel("Forename: ");
        JLabel surnameLabel   = new JLabel("Surname: ");
        //dbImage panel for showing image from database
        String name = "";
        String surname = "";
        BufferedImage buffImg = new BufferedImage(20,20,BufferedImage.TYPE_INT_RGB);
        if(faces!=null)
        {
                Face face = currentMatchedFace;
                name = face.getForename();
                surname = face.getSurname();
                buffImg = FeatureLocalisation.toBufferedImage(face.getFaceImage());
        }
        //dbImage.add(imageIconHost);
        labelPanel.add(forenameLabel);
        labelPanel.add(surnameLabel);
        forenameDisplay = new JLabel(name);
        surnameDisplay  = new JLabel(surname);
        answerPanel.add(forenameDisplay);
        answerPanel.add(surnameDisplay);
        matchFieldsContainer.add(answerPanel);
        matchFieldsContainer.add(labelPanel);
        JLabel imageLabel = buffImageToImageIcon(buffImg);
        dbImage.add(imageLabel);
        matchContainer.add(dbImage);
        matchContainer.add(matchFieldsContainer);
        return matchContainer;
}
        public JLabel buffImageToImageIcon(BufferedImage buffImg)
        {
                JLabel imageHost = new JLabel(new ImageIcon(buffImg));

```

```
229                     return imageHost;
230             }
231             private int averageOfArrayValues(int[] array)
232             {
233                     int average = 0;
234                     int temp = 0;
235                     for(int i = 0; i<array.length;i++)
236                     {
237                             temp =+ array[i];
238                     }
239                     double averageNum = (double)temp/array.length;
240                     average = (int)averageNum;
241                     return average;
242             }
243             public boolean obtainReferenceFaces()
244             {
245                     boolean isSuccess = false;
246                     referenceFaces = dal.getFaceArray();
247                     isSuccess = true;
248                     return isSuccess;
249
250             }
251             public FeatureLocalisation createFacePatchesFromArray(BufferedImage image, int patchSize)
252             {
253                     FeatureLocalisation feature;
254                     //for(int i=0; i < 1;i++)//faces.size();i++)
255                     //{
256                             System.out.println("Get face location");
257                             FaceLocation face = (FaceLocation)faces.get(0);//i);
258                             System.out.println("Got face locaiton 1");
259                             System.out.println(face.toString());
260                             int x = face.getX();
261                             System.out.println(x);
262
263                             int y = face.getY();
264                             System.out.println(y);
265                             int w = face.getWidth();
266                             System.out.println(w);
267                             System.out.println("crop image");
268                             FaceCrop crop = new FaceCrop(image,x,y,w);
269                             System.out.println("get features");
270                             feature = new FeatureLocalisation(crop.getCroppedBufImg(),patchSize);
271                             System.out.println("return features");
272                     //}
273                     return feature;
274             }
275             public static synchronized void addImageToQueue(BufferedImage img)
276     {
277             imageQueue.addWork(img);
278     }
279             public static synchronized void getStackSize()
280             {
281                     System.out.println(imageQueue.getSize());
282             }
283             public void setUpArray()
284             {
285                     faces = locations.getFaceObjects();
286             }
```

```
287             public void getFaceLocations(String fileName)
288             {
289                     System.out.println("Getting face locations");
290                     locations = new FaceLocations(fileName,1,40,40);
291                     System.out.println(locations.getFaceObjects().get(0));
292             }
293     }
```

Figure 8.3-15 - FrameHandler.java source

8.3.16 WorkQueue.java class

```
1  import java.util.LinkedList;
2  import java.awt.image.BufferedImage;
3
4  public class WorkQueue {
5     LinkedList<BufferedImage> queue = new LinkedList<BufferedImage>();
6
7     // Add work to the work queue
8     public synchronized void addWork(BufferedImage img) {
9        queue.addLast(img);
10       notifyAll();
11    }
12
13    // Retrieve work from the work queue; block if the queue is empty
14    public synchronized Object getWork() throws InterruptedException {
15       while (queue.isEmpty()) {
16             System.out.println("Getting work from queue");
17          wait();
18       }
19       return queue.removeFirst();
20    }
21    public int getSize()
22    {
23             return queue.size();
24    }
25 }
```

Figure 8.3-16 - WorkQueue.java source

8.3.17 FaceRecogGUI.java class (Main program entry point)

```
import java.awt.*;
import java.awt.event.*;

import javax.swing.AbstractAction;

import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import java.awt.image.*;
import java.io.*;
import java.util.Timer;
import javax.security.auth.Refreshable;

/**
 *  FaceRecogGUI.java
 *  Author: Martyn Booth
 *  */

/*
 * Main class, provides GUI functionality for all of the other classes and acts as a glue
 */
public class FaceRecogGUI implements ActionListener, ItemListener
{
    JTextArea output;
    JScrollPane scrollPane;
    private static JFrame frame;
    private static JLabel forenameDisplay;
    private static JLabel surnameDisplay;
    private static JLabel imageIconHost;
    private static JLabel imageLabel;
    private static ImageIcon imgIcon;
    private JPanel dbImage;
    private BufferedImage imgFromDB;
    private static JPanel matchDetails;

    public FaceRecogGUI ()
    {
        imageLabel = new JLabel();
    }
    public MenuBar createMenuBar() {

        MenuBar menuBar;
        Menu menu;
        MenuItem menuItem;
        menuBar = new MenuBar();
        menu = new Menu("File");
        menuBar.add(menu);
        menuItem = new MenuItem("Add new Reference Face");
        menu.add(menuItem);
        //      ...for each MenuItem instance:
        menuItem.addActionListener(new ActionListener()
                {
                  public void actionPerformed(ActionEvent event)
                  {
                      FaceAddToDatabase add = new FaceAddToDatabase();
                  }
                }
                );
```

```
        menuItem = new MenuItem("Remove Reference Face");
        menu.add(menuItem);
        //      ...for each MenuItem instance:
        menuItem.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {

               FaceRemoveFromDatabase add = new FaceRemoveFromDatabase();
            }
          }
          );


        return menuBar;
    }

    public Container createContentPane() {
        //Create the content-pane-to-be.
        JPanel contentPane = new JPanel(new FlowLayout());
        contentPane.setOpaque(true);

        //Create a selction of panes and layouts to support the visual components
        JPanel vidPane = new JPanel(new BorderLayout());
        JPanel vidPaneContainer = new JPanel(new BorderLayout());
        vidPane = createAndReturnVideo();
        vidPane.setSize(450, 400);
        JPanel vidPaneName = new JPanel(new BorderLayout());
        vidPaneName.setSize(450, 200);
        JLabel vidLabel = new JLabel();
        vidLabel.setText("Live Video Feed:");
        vidPaneName.add(vidLabel,BorderLayout.CENTER);
        vidPaneContainer.add(vidPaneName,BorderLayout.NORTH);
        vidPaneContainer.add(vidPane,BorderLayout.SOUTH);

        matchDetails = new JPanel(new FlowLayout(),true);
        matchDetails = createMatchingDisplay();


        //Add the panels to the content pane.
        contentPane.add(vidPaneContainer);

        contentPane.add(matchDetails);

        return contentPane;
    }


    public JPanel createAndReturnVideo()
    {
        //Get the video source from the webcam and display on screen
        JPanel videoPanel = new JPanel(new FlowLayout());
        videoPanel.setSize(450, 400);
        FrameAccess access = new FrameAccess();
        access.addVideoToPanel("");
         videoPanel = access.getPanel();
         videoPanel.setOpaque(true);
         videoPanel.setVisible(true);

        return videoPanel;
    }
    public void paint(Graphics graphics)
    {
        }
    public void actionPerformed(ActionEvent e){}
    public void itemStateChanged(ItemEvent e){}

    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
```

```
137         */
138        /*public JPanel createMatchingDisplay()
139        {
140            FrameHandler handler = new FrameHandler();
141            handler.start();
142            matchDetails = handler.createMatchingDisplay();
143            JButton exitButton = new JButton("Exit");
144             exitButton.addActionListener(new ExitApplicationClick());
145             matchDetails.add(exitButton);
146            return matchDetails;
147        }*/
148        private static void createAndShowGUI() {
149            //Create and set up the window.
150            frame = new JFrame("Face Recognition GUI");
151            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
152
153            //Create and set up the content pane.
154            FaceRecogGUI demo = new FaceRecogGUI();
155
156            frame.setContentPane(demo.createContentPane());
157            frame.setMenuBar(demo.createMenuBar());
158
159            //Display the window.
160            frame.pack();
161            frame.setVisible(true);
162        }
163        private class ExitApplicationClick extends AbstractAction
164            {
165                    public ExitApplicationClick()
166                    {
167                    super("Disable frame");
168                    }
169
170                    public void actionPerformed(ActionEvent e)
171                    {
172                    frame.setVisible(false);
173                    System.exit(0);
174                    }
175            }
176        public JPanel createMatchingDisplay()
177        {
178            FrameHandler handler = new FrameHandler();
179            handler.start();
180            //Create the display for showing the face that has been matched to the video
181            JPanel matchContainer = new JPanel(new GridLayout(0,1));
182            dbImage                         = new JPanel(new FlowLayout());
183            JPanel matchFieldsContainer = new JPanel(new FlowLayout());
184            JPanel labelPanel        = new JPanel(new GridLayout(0,1));
185            JPanel answerPanel       = new JPanel(new GridLayout(0,1));
186            JLabel forenameLabel  = new JLabel("Forename: ");
187            JLabel surnameLabel   = new JLabel("Surname: ");
188            //dbImage panel for showing image from database
189            String name = "";
190            String surname = "";
191            //BufferedImage buffImg = new BufferedImage(20,20,BufferedImage.TYPE_INT_RGB);
192            //dbImage.add(imageIconHost);
193            labelPanel.add(forenameLabel);
194            labelPanel.add(surnameLabel);
195            forenameDisplay = new JLabel(name);
196            surnameDisplay  = new JLabel(surname);
197            imgIcon = new ImageIcon();
198            imageLabel = new JLabel(imgIcon);
199            dbImage.add(imageLabel);
200            answerPanel.add(forenameDisplay);
201            answerPanel.add(surnameDisplay);
202            matchFieldsContainer.add(labelPanel);
203            matchFieldsContainer.add(answerPanel);
204            JButton exitButton = new JButton("Exit");
205            exitButton.setSize(50, 25);
206             exitButton.addActionListener(new java.awt.event.ActionListener() {
207                 public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
                    System.exit(0);
                }
            });

            //imageLabel = buffImageToImageIcon(buffImg);
            dbImage.add(imageLabel);
            matchContainer.add(dbImage);
            matchContainer.add(matchFieldsContainer);
            matchContainer.add(exitButton);
            return matchContainer;
        }

        public static void main(String[] args) {
            //Schedule a job for the event-dispatching thread:
            //creating and showing this application's GUI.

            javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    createAndShowGUI();
                }
            });
        }
        public static void reinitaliseMatchDisplay(String forename, String surename, BufferedImage
buffImg)
        {
            forenameDisplay.setText(forename);
            surnameDisplay.setText(surename);
            Image image = buffImg.getScaledInstance(200, 150, 1);
            imgIcon.setImage(image);// = new JLabel(new
ImageIcon(buffImg));//buffImageToImageIcon(buffImg);
            matchDetails.repaint();
            frame.pack();
        }
        public static JLabel buffImageToImageIcon(BufferedImage buffImg)
            {
                    JLabel imageHost = new JLabel(new ImageIcon(buffImg));

                    return imageHost;
            }



}
```

Figure 8.3-17 - FaceRecogGUI.java source code, main program entry point

8.3.18 Java test classes:

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.util.*;
4  import javax.swing.JFrame;
5
6  public class ImageDisplay extends JFrame {
7          private Image image;
8          private ArrayList faces;
9          FaceLocations locations;
10         private String fileLoaded;
11
12         public ImageDisplay(String fileName, int patchSize) {
13
14                 Toolkit toolkit = Toolkit.getDefaultToolkit();
15                 String file = /*"trainPhotos/" +*/ fileName;
16                 fileLoaded = fileName;
17                 image = toolkit.getImage(file);
18                 MediaTracker mediaTracker = new MediaTracker(this);
19                 mediaTracker.addImage(image, 0);
20
21                 try
22                 {
23                         mediaTracker.waitForID(0);
24                 }
25                 catch (InterruptedException ie)
26                 {
27                         System.err.println(ie);
28                         System.exit(1);
29                 }
30                 addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent e)
31                         {
32                 System.exit(0);
33                 }
34                 });
35
36                 setSize(image.getWidth(null), image.getHeight(null));
37                 setTitle(fileName);
38                 getFaceLocations(fileName);
39                 setUpArray();
40                 createFacePatchesFromArray(image,fileName,patchSize);
41                 show();
42
43         }
44         public void getRectanglesAndDraw(Graphics graphicPass)
45         {
46                 getFaceLocations();
47                 setUpArray();
48                 for(int i=0; i < faces.size();i++)
49                 {
50                         FaceLocation face = (FaceLocation)faces.get(i);
51                         int x = face.getX();
52                         int y = face.getY();
53                         int w = face.getWidth();
54                         drawRectangle(graphicPass, x, y, w);
55                 }
56         }
```

```
	public void createFacePatchesFromArray(Image image, String fileName,int patchSize)
	{
		for(int i=0; i < faces.size();i++)
		{
			FaceLocation face = (FaceLocation)faces.get(i);
			int x = face.getX();
			int y = face.getY();
			int w = face.getWidth();
			FaceCrop crop = new FaceCrop(image,x,y,w);
			FeatureLocalisation feature = new FeatureLocalisation(crop.getCroppedImage(),fileName,patchSize);
		}
	}
	public void getFaceLocations()
	{
		System.out.println(getFileName());
		locations = new FaceLocations(getFileName(),1,40,40);
	}
	public void getFaceLocations(String fileName2)
	{
		locations = new FaceLocations(fileName2,1,40,40);
	}
	public void setUpArray()
	{
		faces = locations.getFaceObjects();
	}
	public void setFileName(String name)
	{
		fileLoaded = name;
	}
	public String getFileName()
	{
		return fileLoaded;
	}
	public void paint(Graphics graphics) {
		graphics.drawImage(image, 0, 0, null);
		//  Rectangle2D.Float rc = new Rectangle2D.Float(215F,203F,174F,174F);
		//graphics.drawRect(215, 203, 174, 174);
		getRectanglesAndDraw(graphics);
		/*Date now2 = new Date();
		now2.setTime(System.currentTimeMillis());
		System.out.println(now2);*/

	}
	public void drawRectangles(Graphics graphics)
	{
		getRectanglesAndDraw(graphics);
	}
	public void drawRectangle(Graphics graphic, int x, int y, int z)
	{
		graphic.drawRect(x, y, z, z);
	}

	public static void main(String[] args)
	{
		/*Date now = new Date();
		now.setTime(System.currentTimeMillis());
		System.out.println(now);*/
```

```
114                     String fileTemp = "C:\\Documents and Settings\\Martyn\\My
115 Documents\\Eclipse\\FaceRecognitionProj\\VideoFrameBuffer\\0.jpg";
116                     ImageDisplay display = new ImageDisplay(fileTemp,16);
117                     display.setFileName(fileTemp);
118
119                     //FaceDetection detect = new FaceDetection();
120             }
121 }
```

Figure 8.3-18 - ImageDisplay.java test class source code

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import java.awt.geom.*;
4  import java.awt.image.*;
5  import java.io.*;
6  import java.net.*;
7  import java.util.ArrayList;
8
9  import javax.imageio.*;
10 import javax.swing.*;
11
12 public class ImageDisTest {
13             private FaceDataAccessLayer dal;
14             private Image image;
15
16             public ImageDisTest()
17             {
18                         dal = new FaceDataAccessLayer();
19                         ArrayList<Face> faces = new ArrayList<Face>();
20                         dal.getFacesFromDatabase();
21
22                         faces = dal.getFaceArray();
23                         Face face = (Face)faces.get(0);
24                         image = face.getFaceImage();
25             }
26    public static void main(String[] args) throws IOException {
27       ImageDisTest test = new ImageDisTest();
28       GraphicsConfiguration gc = getDefaultConfiguration();
29       Image thisImage = test.image;
30       BufferedImage buff = FeatureLocalisation.toBufferedImage(thisImage);
31       BufferedImage image = toCompatibleImage(buff, gc);
32       final double SCALE = 1.75;
33       int w = (int) (SCALE * image.getWidth());
34       int h = (int) (SCALE * image.getHeight());
35       final BufferedImage resize = getScaledInstance(image, w, h, gc);
36
37       JPanel p = new JPanel(new GridLayout(1,2));
38       p.add(new JLabel(new ImageIcon(image)));
39       p.add(new JLabel(new ImageIcon(resize)));
40
41       JPanel south = new JPanel();
42       JButton save = new JButton("save");
43       save.addActionListener(new ActionListener(){
44          public void actionPerformed(ActionEvent evt) {
45             try {
46                ImageIO.write(resize, "jpeg", new File("test.jpeg"));
47             } catch (IOException e) {
48                e.printStackTrace();
49             }
50          }
51       });
52       south.add(save);
53
54       JFrame f = new JFrame("Example");
55       f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
56       Container cp = f.getContentPane();
```

```
57          cp.add(p, BorderLayout.CENTER);
58          cp.add(south, BorderLayout.SOUTH);
59          f.pack();
60          f.setLocationRelativeTo(null);
61          f.setVisible(true);
62      }
63
64      public static GraphicsConfiguration getDefaultConfiguration() {
65          GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
66          GraphicsDevice gd = ge.getDefaultScreenDevice();
67          return gd.getDefaultConfiguration();
68      }
69
70      public static BufferedImage toCompatibleImage(BufferedImage image, GraphicsConfiguration gc) {
71          if (gc == null)
72              gc = getDefaultConfiguration();
73          int w = image.getWidth();
74          int h = image.getHeight();
75          int transparency = image.getColorModel().getTransparency();
76          BufferedImage result = gc.createCompatibleImage(w, h, transparency);
77          Graphics2D g2 = result.createGraphics();
78          g2.drawRenderedImage(image, null);
79          g2.dispose();
80          return result;
81      }
82
83      //returns target
84      public static BufferedImage copy(BufferedImage source, BufferedImage target) {
85          Graphics2D g2 = target.createGraphics();
86          g2.setRenderingHint(RenderingHints.KEY_INTERPOLATION, RenderingHints.VALUE_INTERPOLATION_BICUBIC);
87          double scalex = (double) target.getWidth()/ source.getWidth();
88          double scaley = (double) target.getHeight()/ source.getHeight();
89          AffineTransform xform = AffineTransform.getScaleInstance(scalex, scaley);
90          g2.drawRenderedImage(source, xform);
91          g2.dispose();
92          return target;
93      }
94
95      public static BufferedImage getScaledInstance(BufferedImage image, int width, int height, GraphicsConfiguration gc) {
96          if (gc == null)
97              gc = getDefaultConfiguration();
98          int transparency = image.getColorModel().getTransparency();
99          return copy(image, gc.createCompatibleImage(width, height, transparency));
100     }
101 }
```

Figure 8.3-19 - ImageDisTest.java source code

```
1  import java.awt.FlowLayout;
2  import java.awt.Graphics;
3  import java.awt.GridLayout;
4  import java.awt.MediaTracker;
5  import java.awt.Toolkit;
6  import java.awt.image.BufferedImage;
7  import java.util.ArrayList;
8  import java.io.*;
9
10 import javax.imageio.ImageIO;
11 import javax.swing.JButton;
12 import javax.swing.JLabel;
13 import javax.swing.JPanel;
14 import javax.swing.JFrame;
15 import javax.swing.ImageIcon;
16 import java.awt.Image;
17
18 import java.awt.event.WindowAdapter;
19 import java.awt.event.WindowEvent;
20 import java.awt.geom.*;
21 import java.util.TimerTask;
22 import java.awt.*;
23
24 public class DBTestClass extends JFrame
25 {
26
27
28         private FaceDataAccessLayer dal;
29         private Image image;
30
31         public DBTestClass()
32         {
33                 dal = new FaceDataAccessLayer();
34                 ArrayList<Face> faces = new ArrayList<Face>();
35                 dal.getFacesFromDatabase();
36
37                 faces = dal.getFaceArray();
38                 Face face = (Face)faces.get(0);
39                 image = face.getFaceImage();
40                 Toolkit toolkit = Toolkit.getDefaultToolkit();
41                 //toolkit.cre
42                 File fOut = new File("file");
43                 BufferedImage buff = FeatureLocalisation.toBufferedImage(image);
44                 try
45                 {
46                 ImageIO.write(buff, "jpg", fOut);
47                 }
48                 catch(Exception e){}
49
50                 //image = toolkit.getImage(file);
51                 MediaTracker mediaTracker = new MediaTracker(this);
52                 mediaTracker.addImage(image, 0);
53
54                 try
55                 {
56                         mediaTracker.waitForID(0);
57                 }
58                 catch (InterruptedException ie)
```

```
59                      {
60                              System.err.println(ie);
61                              System.exit(1);
62                      }
63                      addWindowListener(new WindowAdapter() { public void windowClosing(WindowEvent e)
64                              {
65                      System.exit(0);
66                      }
67                      });
68
69                      setSize(image.getWidth(null), image.getHeight(null));
70                      setTitle("Test");
71                      show();
72
73              }
74              public void paint(Graphics graphics) {
75                      graphics.drawImage(image, 0, 0, null);}
76              public static void main(String[] args)
77              {
78                      System.out.println("Going");
79                      DBTestClass test = new DBTestClass();
80              }
81              public static GraphicsConfiguration getDefaultConfiguration() {
82                 GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
83                 GraphicsDevice gd = ge.getDefaultScreenDevice();
84                 return gd.getDefaultConfiguration();
85              }
86
87              public static BufferedImage toCompatibleImage(BufferedImage image, GraphicsConfiguration gc) {
88                 if (gc == null)
89                    gc = getDefaultConfiguration();
90                 int w = image.getWidth();
91                 int h = image.getHeight();
92                 int transparency = image.getColorModel().getTransparency();
93                 BufferedImage result = gc.createCompatibleImage(w, h, transparency);
94                 Graphics2D g2 = result.createGraphics();
95                 g2.drawRenderedImage(image, null);
96                 g2.dispose();
97                 return result;
98              }
99
100 }
```

Figure 8.3-20 - DBTestClass.java test class source code

8.4 Appendix D – System Testing

8.4.1 System Testing Results

Test No.	*Description*	*Successful?*	*Reference*	*Additional*
	Interfaces correctly displayed			
1	Main interface is correctly displayed	Yes	Figure 8.2-1	
2	Add reference face interface is correctly displayed	Yes	Figure 8.2-2	
3	Remove reference face interface is correctly displayed	Yes	Figure 8.2-3	
	Testing Interface			
4	Menu items launch relevant sub-interfaces	Yes	Figure 8.2-2	
5	Main interface begins with no matching face displayed	Yes	Figure 8.2-3	
6	Test video is displayed constantly	Yes	Figure 8.2-1	
7	Connection to database failed message shown if connection fails	Yes	Figure 8.4-1	
8	Console shows feature information when adding reference face then clears screen	Yes	N/A	
9	Message is displayed if reference image is missing	Yes	Figure 8.4-2	The reference image is stored in a file store and the path is in the database. This checks for manual tampering
10	Message is shown if .dat file (Covariance) from Matlab are missing	Yes	Figure 8.4-4	

Test No.	*Description*	*Successful?*	*Reference*	*Additional*
11	Message is shown if .dat file (Mean) from Matlab are missing	Yes	Figure 8.4-3	
12	Message s shown if Video capture device is unavailable	Yes	Figure 8.4-5	
13	Reference face is added to filestore	Yes	N/A	Added on disk
14	Matched person details appear on screen and are updated periodically	Yes	Figure 8.4-6	

Table 8.4-1 - System tests

Figure 8.4-1 - Database unavailable window

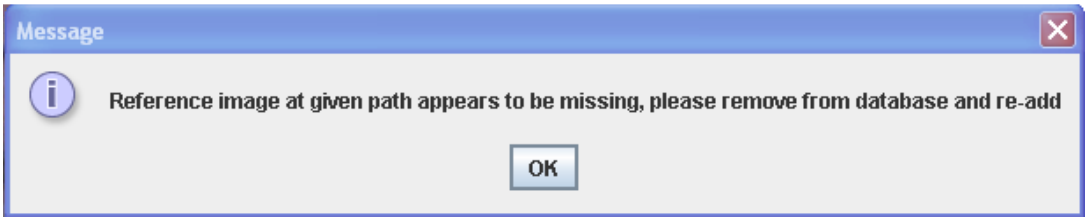

Figure 8.4-2 - Reference Image missing warning

Figure 8.4-3 - Mean .dat file missing error message

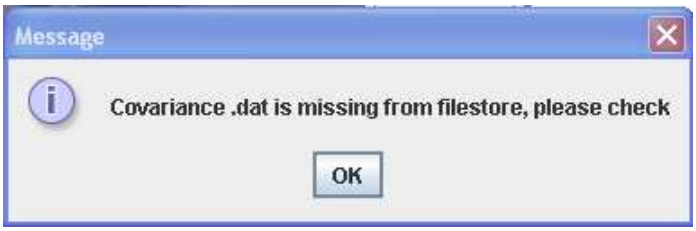

Figure 8.4-4 - Covariance .dat file missing error message

Figure 8.4-5 - Video source unavailable error message

Figure 8.4-6 - Final running main interface

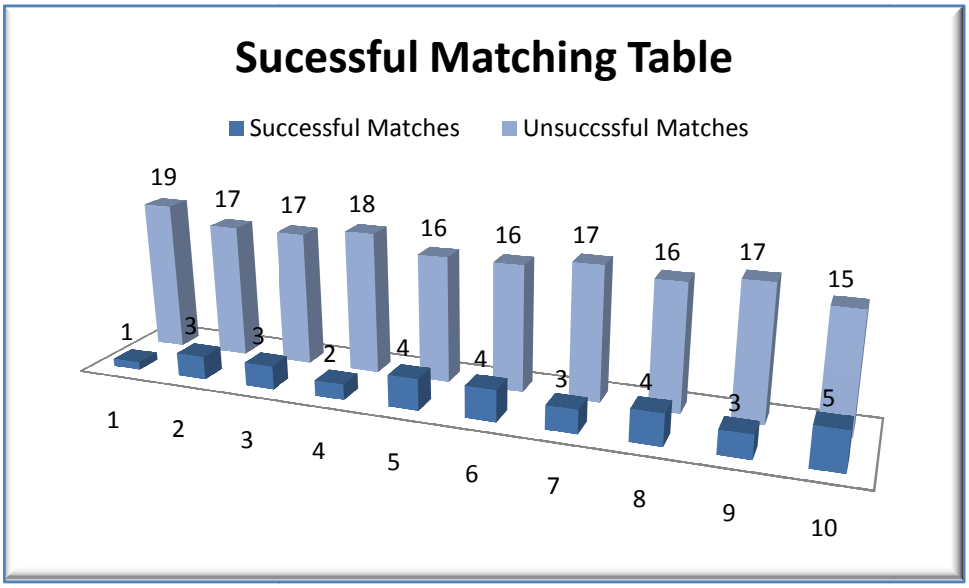

Figure 8.4-7 - Results Graph

8.4.2 *System Requirements Testing Results*

Number	Successful?	Comments
1	Yes	The video feed works correctly
2	Yes	Reference image are cropped about the face but not normalised anymore than that (they are, however, normalised as part of the feature extraction process.)
3	Yes	Bayesian feature localisation utilised
4	Yes	Weak Classifier face detector used
5	Yes	Provided by specialist interface
6	Yes	Image is updated periodically with closest match
7	No	No validation is required because the data is string-like (for the name) and the face is validated automatically by the face detector
8	Yes	Error messages are used extensively and are informative

Table 8.4-2 - System requirements results ref. (Table 4.2-1)

Number	Successful?	Comments
1	Yes	Figure 8.2-1, although this is subjective to user preferences
2	Yes	Figure 8.2-2
3	Yes	Although, real-time here is 4 frames per

Number	Successful?	Comments
		second
4	Yes	Figure 8.2-1
5	Yes	Figure 8.4-1 through Figure 8.4-5

Table 8.4-3 - User requirements results ref. (Table 4.2-2)

Number	Successful?	Comments
1	Yes	Java, Figure 8.3-17
2	Yes	Figure 8.4-1 through Figure 8.4-5
3	Yes	Figure 8.2-2 and Figure 8.2-3
4	Yes (theoretically)	Although, all reference images were in .jpeg format
5	Yes	Figure 8.2-2
6	Yes	Figure 8.2-3

Table 8.4-4 - System specification results ref. (Table 4.2-3)

Number	Successful?	Comments
1	Yes	Although, real-time here is 4 frames per second
2	Yes	Figure 8.4-7
3	Yes	N/A

Table 8.4-5 - Face Detector Specification results ref. (Table 4.3-1)

8.4.3 Evaluation Framework

The following evaluation framework diagram (Figure 8.4-8) is a useful diagram used to highlight certain issues or successes with an application framework. The right hand column classifies the results as; Advanced (A), Intermediate (I) or Basic (B). These refer to the system implementation as compared to the original design specification. If no 'tick' has been placed in the 'category' section, it is not relevant in this scenario.

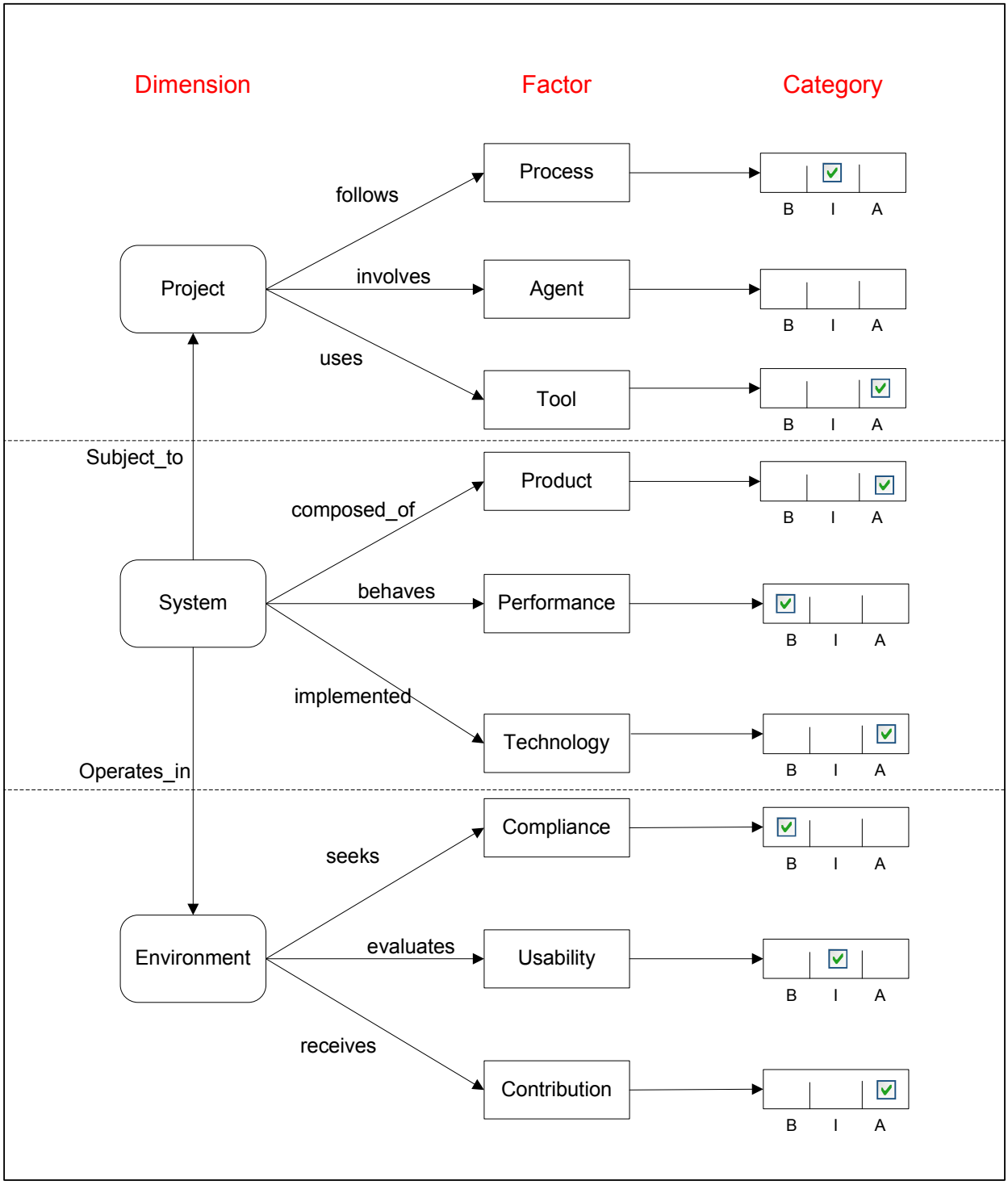

Figure 8.4-8 - Evaluation Framework Diagram

8.5 Appendix E – Personal Reflection

When I began this project, I wanted to pick something that satisfied a number of criteria;

- For the project to be suitably complex in order to challenge me to a suitable extent
- For the project to teach me about complex programming paradigms for use in industry

- For the project to be loosely linked to the area of computer security and biometrics because this is an area of particular interest to me.

I believe the project realistically started in the worst possible way. I had an uneven module split; weighted toward the first semester and therefore left the project mainly to be done in the second semester. This was a mistake because travelling to and from London in order to conduct job interviews took more time than anticipated. Finally, coming into Easter, I was significantly behind schedule and the project was starting to look in trouble.

However, having worked all hours in the day for a couple of months, I managed to produce something that I was reasonably happy with. The program was becoming extremely complex but was producing excellent results and was nearing completion. The final push of Easter (and the post-Easter coursework lull) gave me time to write up the project properly and finalise the code by 'gluing' the individual components together.

I believe that it would have been immensely beneficial for me to conduct much more background research earlier. This means not only looking aimlessly at the mounds of research available, but spending time with the library staff in order to gain a better understanding of how to find the information that I needed. I must have wasted days aimlessly searching for relevant information.

I then found Google scholar and from there, located the IEEE website (to which access credentials are provided by the University library). This website contains hundreds of papers from around the world in the areas of computer vision and image processing. This is truly a 'golden' resource.

I would also strongly suggest that people delay job interviews until after the dissertations are in. This really was a significant strain on my time and I would have been much more comfortable in completing the project and maybe even adding additional feature detectors to the software itself.

However, I believe that the programming went very well and am extremely pleased with the results of that. Given more time, as mentioned above, I would have liked to extend the project to use more feature detectors and therefore make it a more reliable systems. Time constraints, which should really have been managed better, made this impossible.

If I were to redo this project, I would begin by looking through a large number of previous projects on this topic because that would have given me a better idea of exactly what was expected of me. This would save me time later on, during the report writing phase, on time spent editing and adding missed sections.

To summarise, planning and time keeping is paramount and I would have liked to do this better. Relying on a supervisor's knowledge and using the time in meetings to maximum effect was extremely beneficial to my project and I wouldn't have been able to complete it without that help. Finally, spend some time with library staff to understand how to use the search system. It's more complex than it appears to be.

As a side note; plan in advance to have somebody read over the report well before submission. Unfortunately, I failed to do this and couldn't find anybody that would do it at short notice. Therefore, I had to rely on my own, extremely tired, eyes.

8.6 Bibliography

1) Aarsten, A., Brugali, D., & Menga, G. (1996). *Patterns for Three-Tier Client/Server Applications*. Torino, Italy: Dept. of Automatica e Informatica.

2) Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence. 19*, pp. 711-720. IEEE Computer Society.

3) Bing-Bing, C., Vass, J., & Zhuang, X. (1999). Significance-Linked Connected Component Analysis for Wavelet Image Coding. *IEEE TRANSACTIONS ON IMAGE PROCESSING. 8*, pp. 774-785. Princeton: IEEE Computer Society.

4) Boloix, G., & Robillard, P. N. (1995). *A Software System Evaluation Framework.* Montreal, Canada: IEEE Computer Society.

5) Bornet, O. (2005, May 19). *Learning Based Computer Vision with Intel's Open Source Computer Vision Library*. Retrieved April 2007, 2007, from Intel.com Web site: http://www.intel.com/technology/itj/2005/volume09issue02/art03_learning_vision/p04_face_detection.htm

6) Brunelli, R., & Poggio, T. (1993). Face Recognition: Features versus templates. *IEEE Transaction on Pattern Analysis and Machine Intelligence , 15* (10), 1042-1052.

7) Choi, J., Lee, S., Lee, C., & Yi, J. (2001). A Real-time Face Recognition System using Multiple Mean Faces and Dual Mode Fisherfaces. *IEEE Transactions on Pattem Analysis and machine Intelligence* (pp. 1686-1689). Suwon, Korea: IEEE Computer Society.

8) Dr. Hu Software. (2002). *JNI Face Detection (OpenCV wrapper in Java)*. Retrieved February 21, 2007, from Drhu.org: http://www.drhu.org/freeDownload.asp

9) Dyer, C. R. (2004). *Chuck Dyer on Facial Recognition*. Retrieved April 6, 2007, from The University of Wisconsin: http://www.cs.wisc.edu/~gdguo/faceExpression.htm

10) Efford, N. (2006). *SY32 Lecture 14*. Retrieved April 07, 2007, from www.comp.leeds.ac.uk/sy32: http://www.comp.leeds.ac.uk/sy32/lectures/lect14b.pdf

11) Everingham, M., & Zisserman, A. (2006). Regression and Classification Approaches to Eye Localization in Face Images. *7th International Conference on Automatic Face and Gesture Recognition* , 441-448.

12) Heisele, B. (2006). A Component-based Framework for Face Detection and Identification. *International Journal of Computer Vision* .

13) Heusch, G., Cardinaux, F., & Marcel. (2003). Lighting Normalization Algorithms for Face Verification. *IDIAP Research Institute* .

14) Hjelmås, E., & Wroldsen, J. (1999). Recognizing Faces from Eyes Only. *In Proceedings of the 11th Scandinavian Conference on Image Analysis.* Oslo.

15) Horn, C., Gatune, J., Thomas, A., & Woodward, J. (. (2003). *Biometrics; A Look at Face Recognition.* Virginia State Crime Commission: RAND Public Safety and Justice.

16) Jiao, F., Gao, W., Chen, X., Cui, G., & Shan, S. (2002). A Face Recognition Method Based on Local Feature Analysis. *The 5th Asian Conference on Computer Vision.* Melbourne, Australia.

17) Kawaguchi, T., Rizon, M., & Hidaka, D. (2005). Detection of Eyes from Human Faces by Hough Transform and Separability Filter. *Electronics and Communications in Japan, Part 2 , 88* (5), 2190–2200.

18) Kirby, M., & Sirovich, L. (1990). Application of the Karhunen-Loève Procedure for the Characterization of Human Faces. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (pp. 103-108). IEEE.

19) Kittler, J., & Nixon, M. S. (2003). Expression-Invariant 3D Face Recognition. *Audio & Video-ased Biometric Person Authentication, LNCS 2688* , 62-70.

20) Low, A. (1991). *Introductory Computer Vision and Image Processing.* McGraw-Hill Book Company.

21) Meek, J. (2002, June 13). *Guardian.co.uk | Newham Facial Recognition*. Retrieved March 22, 2007, from Guardian.co.uk: http://www.guardian.co.uk/Archive/Article/0,4273,4432506,00.html

22) Moon, H., Chellappa, R., & Rosenfeld, A. (2002). Optimal Edge-Based Shape Detection. *IEEE Transactions on Image Processing. 11*, pp. 1209-1226. Maryland: IEEE Computer Society.

23) Osuna, E., Freund, R., & Girosit, F. (1997). Training Support Vector Machines: an Application to Face Detection. *IEEE Transactions for Pattern Recognition and Machine Learning* (pp. 130-136). Cambridge, MA: IEEE Computer Society.

24) Pezdek, K., Blandon-Gitlin, I., & Moore, C. (2003). Children's Face Recognition Memory: More Evidence for the Cross-race effect. *Journal of Applied Psychology , 88* (4), 760-763.

25) Roth, D., Yang, M.-H., & Ahuja, N. (2000). *A SNoW-Based Face Detector.* Urbana, Il: Department of Computer Science and the Beckham Institute.

26) Rowley, H. A., Baluja, S., & Kanade, T. (1998). Neural Network-Based Face Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence. 20.* IEEE.

27) Rowley, H. A., Baluja, S., & Kanade, T. (1998). Rotation Invariant Neural Network-based Face Detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 38-44). IEEE.

28) Samal, A., & Iyengar, P. A. (1992). Automatic recognition and analysis of human faces and facial expressions: a survey. *Pattern Recognition , 25* (1), 65-77.

29) Smith, I. L. (2002). *A tutorial on Principal Components Analysis.* Otago, NZ.

30) Statsoft. (2004). *Support Vector Machines (SVM)*. Retrieved April 12, 2007, from Statsoft.com: http://www.statsoft.com/textbook/stsvm.html

31) Torres, L. (2004). *Is there any hope for face recognition?* Barcelona, Spain: Technical University of Catalonia.

32) Turk, A. M., & Pentland, A. P. (1991). Face Recognition using Eigenfaces. *Computer Vision and Pattern Recognition* (pp. 586-591). Minnesota: IEEE Computer Society.

33) University College London; Oncology. (2006). *Principle Components Analysis (PCA)*. Retrieved April 11, 2007, from UCL.ac.uk: http://www.ucl.ac.uk/oncology/MicroCore/HTML_resource/PCA_1.htm

34) Vapnik, V. (1995). *The Nature of Statistical Learning Theory.* New York: Springer.

35) Vapnik, V., & Cortes, C. (1995). Support-Vector Networks. In L. Saitta (Ed.), *Machine Learning. 20*, pp. 273-297. Boston, MA: Kluwer Academic Publishers.

36) Vernon, D. (1991). *Machine Vision. Automated Visual Inspection and Robot Vision.* Prentice Hall.

37) Viola, P., & Jones, M. J. (2004). Robust Real-Time Face Detection. *International Journal of Computer Vision 57(2)* (pp. 137-154). Netherlands: Kluwer Academic Publishers.

38) Yang, J., Zhang, D., Frangi, A. F., & Yang, J.-y. (2004). Two-dimensional PCA: A New Approach to Appearance-Based Face Representation and Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence. 26*, pp. 131-137. Hong Kong: IEEE Computer Society.

39) Yang, M.-H. (2002). Kernel Eigenfaces vs. Kernel Fisherfaces: Face Recognition Using Kernel Methods. *Fifth IEEE International Conference on Automatic Face and Gesture Recognition.* Mountain View, CA: IEEE Computer Society.

40) Zelek, J. S., & Ersi, F. E. (2006). Local Feature Matching For Face Recognition. *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision* (pp. 4 - 4). Waterloo: University of Waterloo.

41) Zhao, S., & Grigat, R.-R. (2006). *An Automatic Face Recognition System in the near Infrared Spectrum.* Hamburg: Technical University Hamburg Harburg, Vision Systems.

42) Zhao, W., Chellappa, R., Rosenfeld, A., & Phillips, P. J. (2007). *Face Recognition: A Literature Survey.* Maryland: National Institute of standards and Technology.

