|  |
| --- |
| University of reading |
| A Face Authentication System |
| Final Year Project |
|  |

**Project Supervisor: Hong Wei**

**Project ID: A-FACE**

**Module Code: SE3IP11**

**Student Name: Tom Bedford**

**Student Number: 21009807**

**Submitted: TBD**

|  |
| --- |
|  |

**Abstract**

Computerised Face authentication is vastly being integrated into today’s technologies and systems. As an efficient and effortless method of character recognition this technology aspires to be prevalent in every-day activities in the aims of managing data and location access and delivering customized user experiences. This report illustrates the design, implementation and effectiveness of a proof of concept facial authentication system. The system utilises Viola-Jones methodology of feature extraction and PCA (Principal Component Analysis) to derive a fast and efficient means of digital face detection and verification. It features a configurable threshold acceptance value in order to handle a range of illumination levels which provide a challenging constraint in image analysis. The developed system can potentially complement an existing user authentication layer or operate as an independent authentication system. Its application can be easily extended to provide some form of greeting message or profile to future students attending a university open day

**Acknowledgments**

Hong Wei – project supervisor

Contents

[Glossary of Terms and Abbreviations 2](#_Toc449042861)

[Introduction 3](#_Toc449042862)

[Problem Articulation / Technical Specification 3](#_Toc449042863)

[The Solution 5](#_Toc449042864)

[Literature Reviews 6](#_Toc449042865)

[Implementation 8](#_Toc449042866)

[Login Interface 8](#_Toc449042867)

[Create Database Interface 12](#_Toc449042868)

[Training Interface 13](#_Toc449042869)

[Testing: Verification and Validation 17](#_Toc449042870)

[Discussion 19](#_Toc449042871)

[Conclusion 19](#_Toc449042872)

[Project Commentary 19](#_Toc449042873)

[Social, Legal, Health & Safety and Ethical Issues 19](#_Toc449042874)

[Reflection 20](#_Toc449042875)

[References 20](#_Toc449042876)

[Appendices 21](#_Toc449042877)

## Glossary of Terms and Abbreviations

HCI – Human Computer Interaction

LDA – Linear Detection Algorithm

PID – Project Initiation Document

GUI – Graphical User Interface

MVC- Model View Controller

API – Application Program Interface

OPENCV – Open Computer Vision

PCA – Principal Component Analysis

Eigen Faces – definition and ref

Haar-like feature – geometrical feature extraction

## Introduction

Biometric analysis has been a focused area of research for many years due to its distinctive features, ease of accessibility and potential industrial application such as government surveillance[1] and more commonly today HCI [2]. Technological advancements have provided more powerful and affordable hardware and open sourced software libraries allowing the technology to be researched more actively. This has resulted in the rise of fully automated recognition systems that are now defining their presence in the world today. Although biometric evaluation such as finger print recognition has delivered more reliable results than face recognition it requires additional expensive hardware to install and demands focused interaction from its user. With the least overhead for system instalment, invasiveness and efficiency, facial recognition has become the most popular choice of biometric analysis.

Facial recognition is the identification and verification of noticeable characteristics of a human face and in the field of image analysis it is leading the race in research as its speed and versatile application out weights previous methods.

Human to computer authentication is an integral functionality of many software systems as it manages data and/or location security. It is a task performed effortlessly by humans on a daily basis. The sheer complexity of the human brain can truly be admired whilst replicating this complex functionality in machine software. Distinctive features of the human face such as the nose, mouth and eyes are algorithmically recognised by object detection and associated to known geometrical shapes.

This project explores the application of face detection and verification and its effectiveness as an authentication based system using the open vision library developed by IBM [3]. The main algorithmic features of the developed face authentication system in this project focus on the detection and verification of a human user. The system efficiency, accuracy and speed is tested by serious of test on compiled face databases sourced from various establishments’ online repositories.

## Problem Articulation / Technical Specification

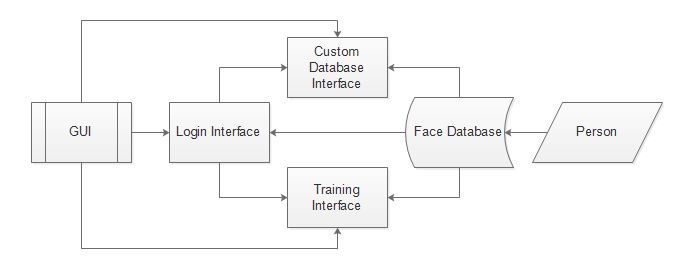
The fundamental goals of the developed system in this study is the successful detection of a user’s face and the verification of a known users within a targeted face databases. Acceptance criteria is satisfied by LDA analysis determining whether the users extracted geometrical features fall within a given threshold in order to grant them access to the application. The threshold of acceptance will be adaptable to suit the needs of the systems environment which substantially dictates acceptance rates of the recognition process. Illumination, user posture, direction, expression and face size are the greatest affecting factors in face recognition computation [4].

It is assumed that users will present there face at an offset no greater than a 35 degree angle and the camera will be of sufficient quality to capture images of an adequate resolution. The user will not be wearing items that obstruct large portions of the face. To provide a reliably secure authentication system recognition results must fall within a refined acceptance threshold that ensures access only to known users of the system. These assumptions and constraints where originally defined in the PID [Appendix 1] report referenced at the end of this document submitted at the beginning of this project.

To satisfy the project objectives derived in the PID each deliverable must meet its acceptance criteria.

The proposed solutions are based on addressing three main modules of the system’s functionality, the GUI, face detection and verification functionality.

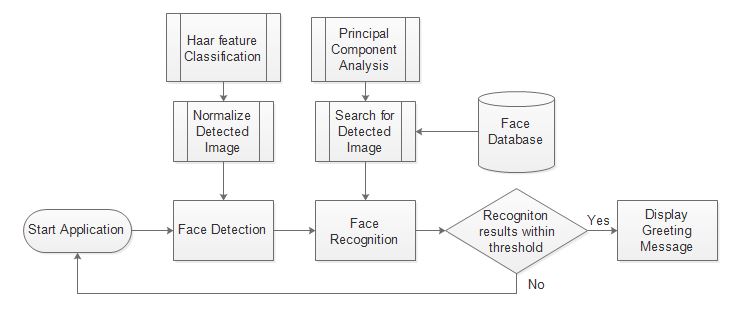
The GUI must be developed to provide a simple interface for the user to detect and capture images of their face to be submitted as part of the authentication process. The GUI will also need to host a training screen where the user can select, load and train face databases prior to user verification.



**Figure. 1** Face authentication graphical user Interface design.

Figure 1 illustrates the work flow of the proposed GUI. It can be seen that the programs user interface is built up of three main displays, the login, training and custom database screens. The login and training screens are essential to the systems core functionality for loading and training data and the custom database screen is to provide further customisation for creating and adapting databases. Developing the GUI in Java language will be less complex and time consuming than C++.

This solution will be developed in Java CV [5] a ported version of the natively C++ Open CV library. The Open CV java library essentially wraps the C++ functionality in Java so that it can be utilised in Java code. JavaFX [6] will be used to develop the systems GUI allowing it be implemented as an MVC framework to allow the systems API to be reused by other systems easily.



**Figure 2.** Recognition process of the facial authentication system.

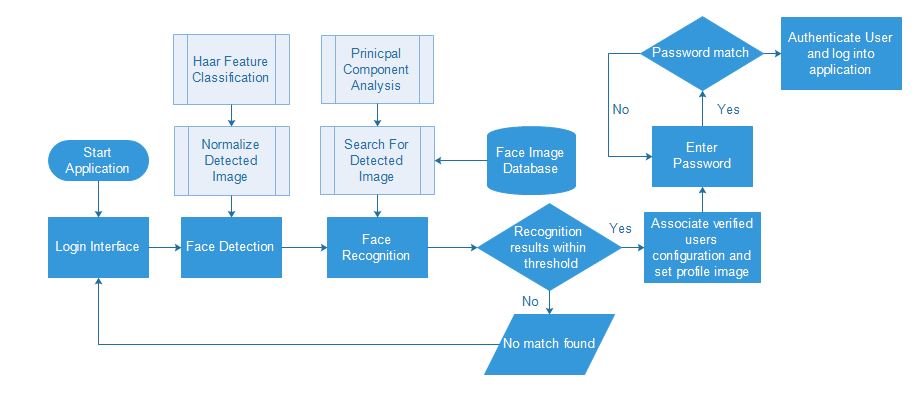
The face detection functionality will be capable of acknowledging a human face by recognising geometrical features of a human face with the use of trained Haar-like feature cascade classifiers provided by the Open CV library [7]. The classifier iterates through image pixel data verify any know geometrical shapes whilst optimising its search results using an AdaBoost algorithm [8]. To satisfy the systems acceptance criteria detection must successfully capture a frontal facing image within an offset of 35 degree angle at a success rate of 90%.

Detected images will then need to be normalised to conform to standardized dimensions of the system and to reduce computation. This is achieved by resizing the detected image and then converting the colour channels to an average greyscale channel.

Principal component analysis will be used reduce and optimise data representation. It is the most popular and efficient LDA’s (Linear Detection Algorithm) in developing facial recognition technologies today. The PCA approach

## The Solution

**Figure. 3** Proposed system solution design



The proposed solution as displayed in figure.3 will be developed using a Java wrapped Open CV library Java CV to enable simpler development of the GUI’s so that efforts can be focused towards the main algorithmic features of the system. JavaFX library will be used to implement an MVC structure allowing the main functionality of the system to be utilised or integrated into other systems easily.

The open vison library supplies a variety of classifiers that have been trained to detect geometrical feature shapes of a human face. The classifier ‘haarClassifier\_frontalFace\_alt.xml’ that will be used to perform face detection is based on viola jones methodology [9]. The detected image is then normalized by rescaling and greyscale conversion to reduce computation and conform to dimensional standards of the recognition algorithm. As the Java CV library does not incorporate the Face Recognizer class [10] a custom PCA class will be developed by the use of Open CV and third party math libraries.

This solution aims to verify the user and associate them to the relevant profile though the login interface. A password is then entered to ensure a secure authentication system. It was decided not to use the facial authentication system developed in this project as a stand-alone security system and should instead compliment an existing system as an extra layer of security.

The main algorithm of the system will be principal component analysis to calculate Eigen Decomposition and effectively derive Eigen Faces [11]. The system will initially provide functionality to load directories of face images from local memory and may be expanded to access online repositories depending on objectives deliverance meeting prosed deadlines.

The application GUI will be based on the module illustrated in figure.1 earlier in this document. The design will provide the necessary functionality the user will need to operate the applications features.

The original determined acceptance criteria of the PID documents technical products have been refined and illustrated below in figure.4.

|  |  |
| --- | --- |
| **Technical Product** | **Acceptance Criteria** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Acceptance testing through thresholding comparison of trained data sets. The principal components represented as Eigen values and vectors are computed on the database image sets. The user image Is then captured using face detection and added to the relevant database image set. PCA analysis is then run on the updated image set which delivers a deviation percentage from the original analysis. The new image set is accepted if the value deviation is still in the given threshold of acceptance.

Recognition Acceptance:

Recognition acceptance is tested by performing PCA on the original database image set. The new user image is then added to the images set where PCA analysis is performed again. If the new image set PCA analysis falls within an acceptable threshold derived from the original PCA analysis the image is verified and validated. This image is then associated to the relevant user profile and image set.

Eigen Value Decomposition

## Literature Reviews

Methods of human authentication have previously and still are performed using unique credentials such as personal ID cards, passports, unique passwords and phrases. These methods of authentication often require some form of physical interaction or focus from its user. All methods of computerised human identification operates on the basis that a set of uniquely identifiable data is submitted by a user to the system where it is compared and associated to its relevant data within a determined threshold of acceptance. The form of this uniquely identifiable data has largely been an extension of a person such as a bank card or password. Biometrics offer the convenience of no extra materials, assets or memory cognition as they are physical features that individually identify a person.

The face is the most exposed definitive characteristic of a human and therefore can be analysed with little demand of interaction from the person if any. A person need only look towards the scanning device which in this case is a camera for seconds and recognition can be successfully performed in real time or from a captured image frame. Face recognition is a popular method of identification in government surveillance systems [government surveillance]. It can be carried out on multiples entities simultaneously with speed which is extremely effective in crowds of people.

In the infantile stages of facial analysis feature extraction was performed manually. The automation of this process in real time was introduced by Paul Viola and Michael Jones in 2001 [voila jones ref]. Not only does the algorithm detect facial features in real time but performs this with a simpler complexity of computation allowing computers with low processing power to utilise the software. The Viola-Jones method of Haar-like feature classification has revolutionised face detection techniques.

In December last year the company bioID [bioID] released a facial recognition software solution for apple hardware. The software allows the use of face authentication to authorize access to devices such as the IPad and IPhone. It features a patented liveness detection algorithm [liveness detection] to determine whether the user is fact a real person or not. Liveness detection is become a common practice in face detection in the efforts to prevent spoofing and unauthorized access. The software also host a rich development package that can be easily integrated into software development to encourage developers to incorporate facial authentication into future software applications. Other programming language versions of this software are slowly becoming available.

Again last year there was the release of an advanced facial recognition system called DDFD (Deep Dense Facial Detection) [DDFD]. The detection algorithm has been extended from the foundations of Viola-Jones methodology of face detection [viola ref] and uses a convolution neural network to learn a composition of facial features. Impressively this algorithm is trained on databases containing over 21k images. This is one of the most robust face detection systems on the market today as it can successfully detect partial faces, upside down oriented faces and largely obstructed faces.

Cognitec is another company offering multipurpose face recognition system solutions for use in security authentication in fields such as law enforcement, border control and ID fraud protection. The company has been researching facial recognition technology in collaboration with the university of Surrey and Technical University of Dresden since 2010 [cognitec].

In a few months from now Microsoft windows 10 will be introducing a new component dubbed ‘Windows Hello’ [windows hello] to their popular operating system. This new functionality will allow windows users to login using face recognition technologies. Microsoft claim this will be more secure than older authentication methods like passwords and will strengthen computer security as a whole.

There are many organisations investing largely in face recognition development as the vision of effortless human computer interaction becomes a reality. The United States National Intelligence Agency launched a development program in early 2014 named Janis []. The aims of Janis is to compile a broad database of facial morphology derived from online digital media and sources to understand and develop more sophisticated and optimal facial analysis techniques. This rich database will provide an intensive testing plane for facial recognition algorithms and will accelerate their development.

Well renowned security conferences such as the Black Hat [Back Hat] and Def Con [Def Con] conferences attract some of the world’s leading specialists in information security under one roof to diverge and discuss security practices and technologies. An article posted shortly after the Black Hat conference back in 2009 in Washington DC voiced concerns over facial biometric use for authentication. The salient features of the report demonstrated how early versions of face authentication integrated into Lenovo, Asus and Toshiba machines could be hacked and did not fulfil the necessary requirements. Although facial analysis methods are becoming more sophisticated and reliable for security there are still concerns over the handling and management of biometric data due to its permanent nature.

A group of biometric computation experts demonstrated at the 2015 Black Hat conference how retinal analysis could be re-engineered and potentially hacked to falsify authentication [retinal hack black hat]. This form of re-engineering firstly required gaining access to the stored biometrical data but illustrated that even the most complex of biometric analysis could be exploited. Amongst the concerns of biometrical data management their also worries that the integration of facial recognition systems increase surveillance coverage and reducing the public privacy.

PCA is one of the most successful face detection algorithms to date. PCA is the method of reducing dimensionality of a data space also referred to as observational variables) and extracting data from the feature space (Independent Variables) which are needed to represent the data economically. The feature space in this case is the Eigen Space in which derived principal components are projected to deduce Eigen values.

* PCA is one of the most popular methods for face image analysis. It essentially optimizes the data needed to represent a defined set of data.

APPLICATIONS OF FACE RECOGNITION

Infrared technologies are currently being used in carparks to simulate a daytime like environment to extract car number plates in poor lighting conditions. This allows image recognition methods to extract the characters of the number plate. Infrared scanners used in this capacity are expensive to install and maintain but would provide more mobility of the system and more potential recognition environments.

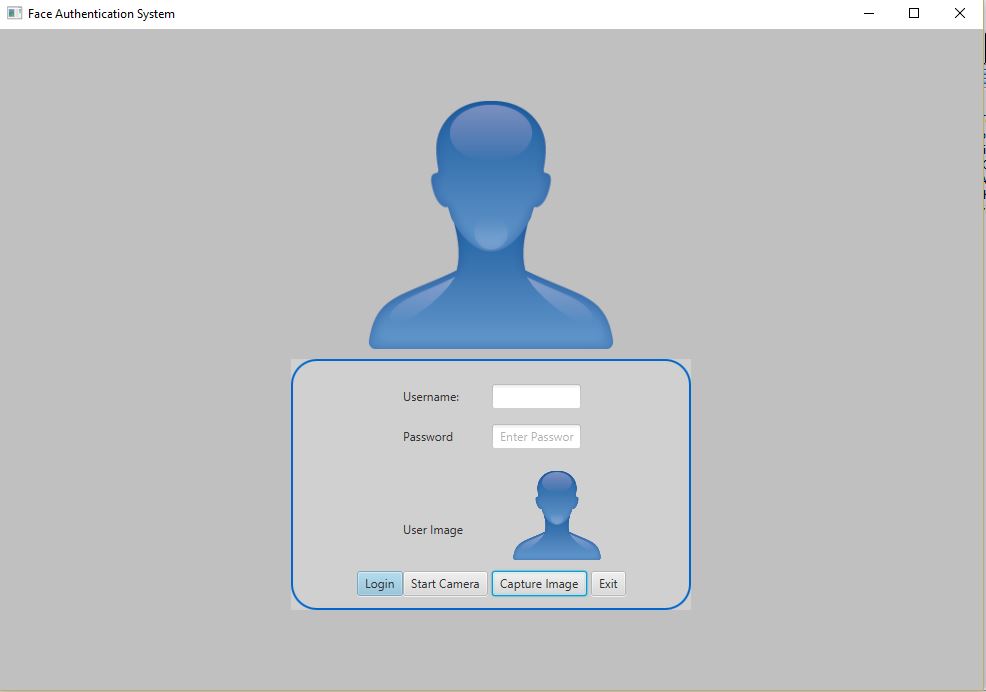
As the environments lighting conditions are a CCP (Critical Control Point) of the systems functionality its placement must be considered careful to cater for its needs.

* Face recognition has been inherited by many high street retailers to recognise VIP guests/customers so that a platinum service can be delivered.
* Most commonly face recognition is being used in surveillance and Security. Crowd control. Lost device services.
* Extended face recodnition research investigates the use of thermal cameras to collect image data. This allows a better representation of a 3D face model as it does not detect facial items such as glasses and records only heat signature data. <https://en.wikipedia.org/wiki/Facial_recognition_system>

## Implementation

### Login Interface

A basic GUI framework was initially implemented and later enhanced to satisfy the needs of interaction to the system. The JavaFX library used to develop the GUI allowed the creation of an MVC based model isolating the API and GUI allowing modular reuse of the system. Figure.1 illustrates a screen shot of the login interface.



**Figure.n** Screenshot of login screen graphical user interface

The login screen provides an area where the user can test user authentication. On first time login of the system access is granted using an administrative username and password so that an initial face image data set can be loaded and trained. After the trained data has been produced it is then written to memory so that then a known user can login using the facial verification functionality.

The login screen hosts a video capture stream activated by the ‘Start/Stop Camera’ button. The video capture objects implemented throughout the system is sourced from the ‘opencv.videoio’ library. Video stream data is handled in memory using OpenCV’s Matrices object format ‘Mat’. To comply with Java AWT [Java AWT] image display standards ‘Mat’ objects are converted to a Java.awt Image using OpenCV’s image codecs in order to display them in the GUI.

|  |
| --- |
| **protected** Image Mat2Image(Mat frame) {  // temporary buffer  MatOfByte buffer = **new** MatOfByte();  // encode image frame into PNG format  Imgcodecs.*imencode*(".PNG", frame, buffer);  // build image from encoded buffered data  **return** **new** Image(**new** ByteArrayInputStream(buffer.toArray()));  } |

**Figure.N** OpenCV Matrix to Java awt conversion function

Each grabbed frame from the video stream is processed using the FaceDetector.class developed for this application. The face detector class loads into memory a trained cascade classifier ‘haarcascade\_FrontalFace\_alt.xml’ supplied by the OpenCV library. The grabbed frame is converted to greyscale to reduce colour channels and effectively data to process. Greyscale conversion functionality is incorporated in OpenCV’s Imgproc module originally derived from OpenCV core module [OpenCv Core Mod]. During system runtime there are multiple areas of the application where image type conversion is computed. The class AppTools.class has been developed to manage these conversions and has been referenced in the Appendices of this report.

The face detection algorithm is performed on each grabbed frame of the video sequence to determine a human face through Haar-like feature cascade classification. This involves loading the classifier into the Cascade Classifier object supported by the Open CV library. The frame is then prepares for detection by converting the frame to greyscale and calculating the minimum face size. The following code seen in figure.n illustrates this preparation.

|  |
| --- |
| //create cascade classifier  CascadeClassifier faceDetector = **new** CascadeClassifier();  //absolute path to classifiers  String classifierPath = "C:\\Users\\user\\workspace\\AFaceAuthenticationSystem\\src\\application\\Resources\\HaarCascades\\haarcascade\_frontalface\_alt.xml";  //load classifiers  faceDetector.load(classifierPath);    //Create Mat canvas to store detections  MatOfRect faceDetections = **new** MatOfRect();  Mat greyScaleImg = **new** Mat();    // convert the frame in gray scale  Imgproc.*cvtColor*(imageMat, greyScaleImg, Imgproc.***COLOR\_BGR2GRAY***);  // compute minimum face size (20% of the frame height)  **if** (absoluteFaceSize == 0)  {  **int** height = greyScaleImg.rows();  **if** (Math.round(height \* 0.2f) > 0)  {  absoluteFaceSize = Math.round(height \* 0.2f);  }  } |

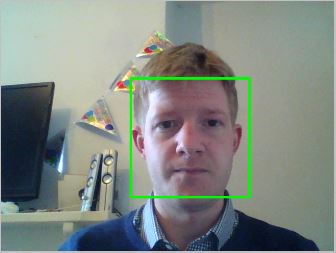
**Figure.n** cascade classification preparation

Detection is then performed on the prepared frame. This process iterates through the frames pixel data to distinguish known geometrical features of the classifier.

|  |
| --- |
| // detect faces  faceDetector.detectMultiScale(greyScaleImg, faceDetections, 1.1, 2, 0 | Objdetect.***CASCADE\_SCALE\_IMAGE***, **new** Size(  absoluteFaceSize, absoluteFaceSize), **new** Size());  //create array of face detections computed  Rect[] faceArray = faceDetections.toArray();  //iterate through the image  **for** (**int** i=0; i<faceArray.length; i++)  {  //find rectangle contours of faces  Imgproc.*rectangle*(imageMat, faceArray[i].tl(), faceArray[i].br(), **new** Scalar(0, 250, 0, 255), 3);  //crop image of face  Mat crop = imageMat.submat(faceArray[i]); |

**Figure.N** calculate face detections functionality from FaceDetector.class

Detected faces in each frame are then outlined with a green rectangle of the calculated absolute face dimensions in figure.N to signify face detection to the user.



**Figure.N** Face Detection

Once the user has logged in for the first time added there detected face to a dataset and computed Eigen faces through the training interface an input image can then be verified against the applications trained data on login authentication. This is done by a similar process to training the data set where firstly the average face is subtracted from the input image and then the weights and distances of the image minus averages are calculated. Figure.N and N demonstrate this processes logic.

|  |
| --- |
| **private** Matrix getWeights(Matrix matrix, **int** numOfEigenFaces) {  Matrix eigenFaces = **new** Matrix(cache.getEigenFaces());  Matrix selectedEigenFaces = eigenFaces.getMatrix(0,  eigenFaces.getRowDimension() <= numOfEigenFaces ? eigenFaces.getRowDimension() - 1 : numOfEigenFaces, 0,  eigenFaces.getColumnDimension() - 1);  **return** matrix.times(selectedEigenFaces.transpose());  } |

**Figure.N** calculate weights of input image functionality

|  |
| --- |
| **private** **double**[] getDistances(Matrix inputWeights) {  **double**[] weightsData = inputWeights.getArray()[0];  **double**[][] tempWeights = subtractFromEachRow(cache.getEigenWeights(), weightsData);  tempWeights = squareNonMatrix(tempWeights);  **double**[] distances = **new** **double**[tempWeights.length];  **for** (**int** i = 0; i < tempWeights.length; ++i) {  **double** total = 0;  **for**(**int** j=0;j<tempWeights[i].length ;++j) {  total += tempWeights[i][j];  }  distances[i] = total;  }  **return** distances;  } |

**Figure.N** calculate distances of input image functionality

Once the distance and weights of the input image have been determined the previously trained data set is read into the application in order to compare these values. Loading in the trained data into memory on the login process reduces computational time opposed to having to train datasets on each login attempt. The trained data is read back into memory with the functionality illustrated in figure.N.

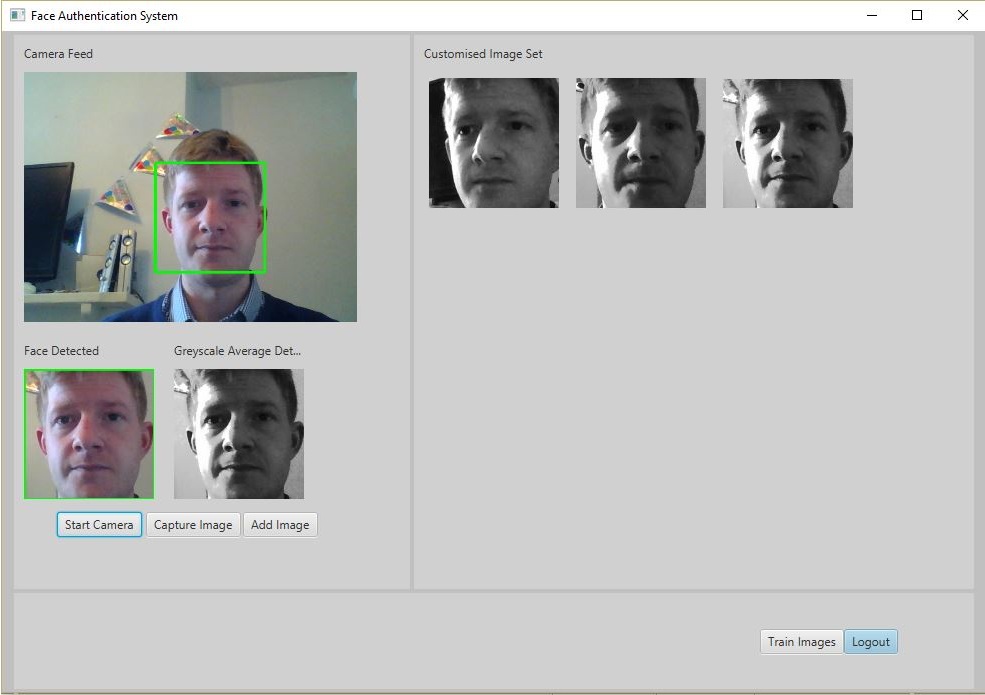
|  |
| --- |
| **private** **void** loadEigenCache() {  // read eigen cache into memory  FileInputStream fs;  **try** {  fs = **new** FileInputStream("C:\\Users\\user\\workspace\\AFaceAuthenticationSystem\\EigenCache\\eigenCache.db");  ObjectInputStream os = **new** ObjectInputStream(fs);  eigenCache = (EigenCache) os.readObject();  // close streams  os.close();  fs.close();  } **catch** (IOException e) {  e.printStackTrace();  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  }  } |

**Figure.N** Load cached trained data into memory functionality

Here a comparison can be made of the calculated Eigen Vectors and Values

### Create Database Interface

The create database interface provides an area where a first time or existing user can capture face detections and either create a customized database or add to an existing one. Essentially as set of images can be obtained and saved as displayed in figure.N. The interface utilises the Face detection algorithm and OpenCV video capture stream components used in the login screen.



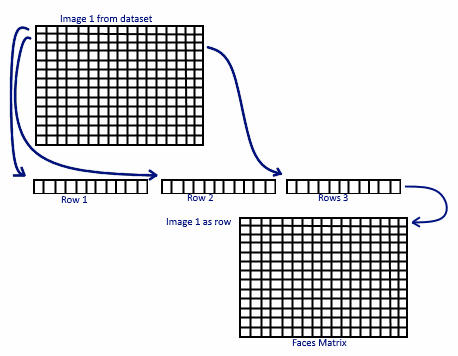
**Figure.N** Create database interface screenshot

### Training Interface

Limitations of the ported OpenCV library JavaCV where discovered here. The native C++ OpencV library supports a FaceRecognizer class that features PCA (Principal Component Analysis) computation functionality. As this module of the OpenCV has not yet been ported to the JavaCV library a custom PCA class was implemented in Java with the aid of third party apache commons Math [] and Jama Matrices libraries []. The custom PCA class developed for this project has referenced in the Appendices of at the end of the report.

The training interface provides the user the ability to load, train and write trained data sets to memory. It also provides a testing ground for trained data sets where the user can select an image to perform face verification on the loaded database. Matches, if any, are displayed along with their calculated thresholds. The thresholds represent the distance from the average face. These calculated thresholds are then accepted if they fall within a set threshold of acceptance. In order to manage the illumination changes of the different environments in which the system may be used a slider has been implemented in the training interface to allow the threshold of acceptance to be adjusted.

PCA concatenates each row of pixel data from an image of the dataset into a one dimensional array. Once each image is represented as a row a Matrix is built up of each image as a row. The logic to method can be seen in figure.n below.



**Figure.n** Face matrix preparation

Once the faces matrix has been constructed pixel values of the data are normalised between zero and one. During the greyscale averaging computation the RGB values are combined to create large decimal numbers. Normalization is performed to unify variable proportions and standardize data in the system. Figure.n displays the system normalization function.

|  |
| --- |
| **private** **double**[][] normalizeImageData(**int**[][] image) {  **double**[][] returnData = **new** **double**[image.length][image[0].length];  // Normalise data between 0 and 1  **for** (**int** faceInd = 0; faceInd < image.length; ++faceInd) {  **double** min = (**double**) getMinValue(image[faceInd]);  **double** max = (**double**) getMaxValue(image[faceInd]);  **for** (**int** j = 0; j < image[faceInd].length; ++j)  returnData[faceInd][j] = ((((**double**) image[faceInd][j])) - min) / (max - min);  }  **return** returnData;  } |

**Figure.n** Normalisation function

Next the mean is subtracted from each row and a covariance matrix is compiled. This was carried out with the aid of the Apache Commons Math library. The Apache library is then again used to calculate Eigen vectors and values. Eigen Vectors represent the direction of data and the associated Eigen Values represent the distance of the data from the average face. The average face used to determine Eigen decomposition is produced by finding the average data from the compiled face matrix as described in figure.n. The calculate Eigen decomposition function is illustrated below in figure.N.

|  |
| --- |
| **private** **void** calculateEigenVectorsAndValues() {  // Log.append("Computing covariance matrix...");  // Compute covariance matrix  RealMatrix matrix = **new** Covariance(**new** BlockRealMatrix(faceMatrixMinusAverages).transpose())  .getCovarianceMatrix();  // Log.append("Computing eigen decomposition...");  // Get the eigenvalues and eigenvectors  EigenDecomposition eigen = **new** EigenDecomposition(matrix);  eigenValues = eigen.getRealEigenvalues();  // Transpose so that eigenvalues are in vectors/columns  eigenVectors = eigen.getV().transpose().getData();  } |

**Figure.N** calculate Eigen decomposition function

The Eigen vectors that represent the most variance are then removed and a set amount of Eigen vectors are selected to represent the Principal Components. A function was developed to compute the principal components and then sort them into descending order prior to selection as displayed in figure.N

|  |
| --- |
| **private** **void** calculatePrincipalComponents() {  **int** numOfComponents = eigenVectors.length;  // Get principle components  ArrayList<PrincipalComponent> principalComponents = **new** ArrayList<PrincipalComponent>();  **for** (**int** i = 0; i < numOfComponents; i++) {  **double**[] eigenVector = **new** **double**[numOfComponents];  **for** (**int** j = 0; j < numOfComponents; j++) {  eigenVector[j] = eigenVectors[i][j];  }  principalComponents.add(**new** PrincipalComponent(eigenValues[i], eigenVector));  }  // sort components  Collections.*sort*(principalComponents);  Iterator<PrincipalComponent> iterator = principalComponents.iterator();  **int** count = 0;  **double**[][] tempVectors = **new** **double**[3][eigenVectors.length];  **double**[] tempValues = **new** **double**[3];  **while** (iterator.hasNext()) {  PrincipalComponent pc = iterator.next();  **if** (count < 3) {  tempVectors[count] = pc.eigenVector;  tempValues[count] = pc.eigenValue;  } **else** {  eigenVectors[count - 3] = pc.eigenVector;  eigenValues[count - 3] = pc.eigenValue;  }  count++;  }  } |

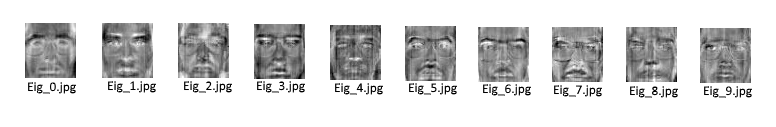
**Figure.N** calculate Principal Component function

|  |
| --- |
| **private** **void** calculateEigenFacesAndWeights () {  **int** pixelTotal = faceMatrixMinusAverages[0].length;  **int** imageTotal = faceMatrixMinusAverages.length;  **int** vectorTotal = eigenVectors.length;  // calculate eigen faces  **double**[][] eigenFace = **new** **double**[vectorTotal][pixelTotal];  **for** (**int** i = 0; i < vectorTotal; ++i) {  **double** squaredSum = 0;  **for** (**int** j = 0; j < pixelTotal; ++j) {  **for** (**int** k = 0; k < imageTotal; ++k) {  eigenFace[i][j] += faceMatrixMinusAverages[k][j] \* eigenVectors[i][k];  }  squaredSum += eigenFace[i][j] \* eigenFace[i][j];  }  **double** norm = Math.*sqrt*(squaredSum);  **for** (**int** j = 0; j < pixelTotal; j++) {  eigenFace[i][j] /= norm;  }  }  // get specified amount of eigen faces  **this**.eigenFaces = **new** Matrix(eigenFace).getMatrix(0,  eigenFace.length <= numOfEigenFacesSelected ? eigenFace.length - 1 : numOfEigenFacesSelected, 0,  eigenFace[0].length - 1).getArray();  **this**.eigenWeights = **new** Matrix(faceMatrixMinusAverages).times(**new** Matrix(eigenFaces).transpose()).getArray();  } |

**Figure. N** calculate Eigen face and weights function

Figure.N illustrates the programming logic use to calculate the Eigen faces and corresponding weights. The use of a third party library called JAMA was used to support matrices computation. The derived Eigen faces are converted back to Java AWT buffered images and written as Jpeg format into memory in order to visualize the variance. A sample of 10 computed Eigen faces are displayed in figure.N.

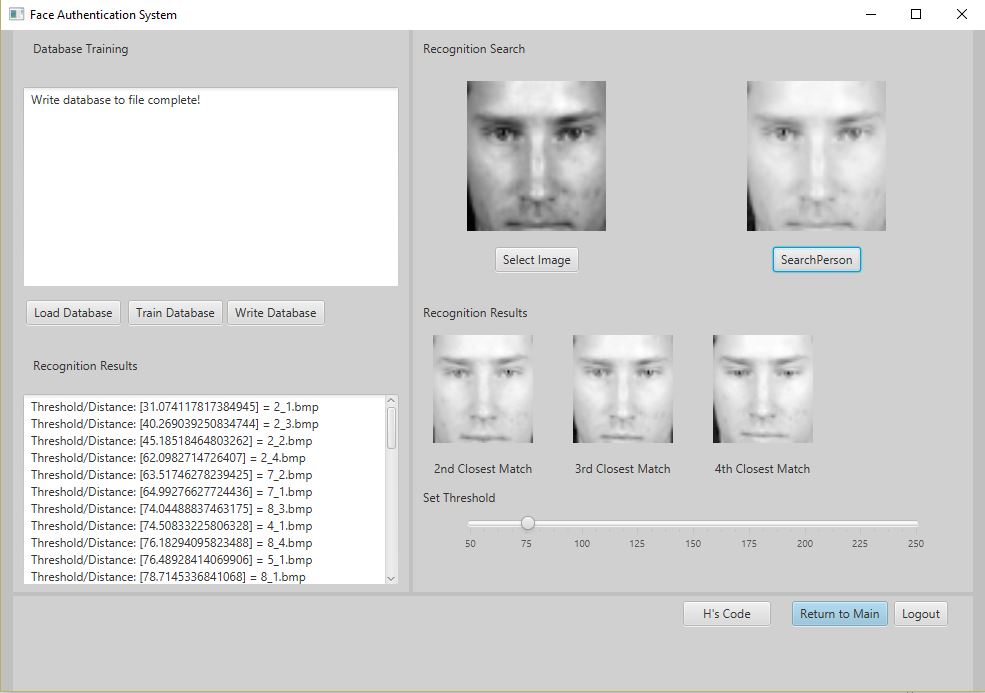
|  |
| --- |
| // construct buffered image from eigen faces  BufferedImage[] constructedEFaces = **new** BufferedImage[faceMatrix\_array.length];  **for** (**int** i = 0; i < faceMatrix\_array.length; i++) {  constructedEFaces[i] = getImageByEigenValues(eigenWeights[i], eigenFaces);  } |

**Figure.N** Constructing Eigen Faces to Jpeg images.

**Figure.N** Calculated Eigen faces for visualizing variance.

From observing the Eigen faces in Figure.n It can be seen that the first few Eigen faces represent the most amount of variance leading to the least variance.

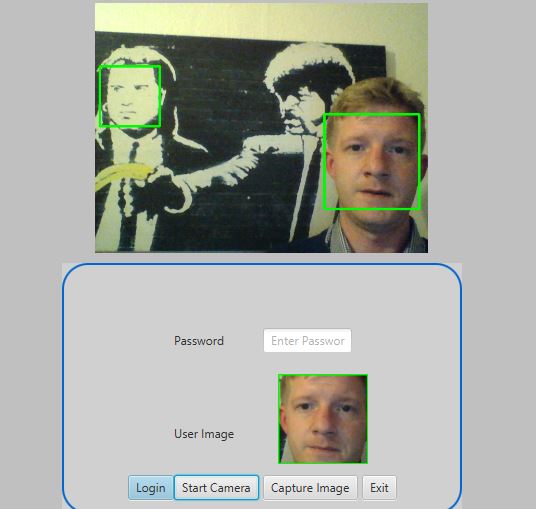
Figure.N illustrates a screen shot of face verification matches and thresholds results after performing a search.



**Figure.N** Screen shot of Face verification results after search performed

## Testing: Verification and Validation

During face detection runtime multiple faces are detected and highlighted in the appropriate manner as demonstrated in Figure.N.



**Figure.N** Multiple faces detected

As the system only wishes to process one user at a time the most prominent face is selected as the detected face. Also as seen in figure.N

With large data sets Eigen decomposition can take a substantial amount of time as the discussed calculations will be performed on a comprised matrix of face images which its size is determined by the amount of images in the data set.

A serious of tests where carried using the XM2VS database compliments of the University of Surrey and the AT&T face database supplied by the University of Cambridge. The statistics and length of time taken to train of each database is recorded in figure.N

|  |  |  |  |
| --- | --- | --- | --- |
| **Database** | **Number of Images** | **Training Time (minutes)** | **Training Time Per Image (seconds)** |
| Xm2vts | 2,361 | 12.654 | 0.322 |
| Xm2vts Sub Sample |  |  |  |
| AT&T | 401 | 2.352 |  |

The face recognition system was tested with a range of thresholds to determine percentages of acceptance rates. The chart illustrated in figure.N shows these results.

2

**Figure.N** Recognition results for xm2vts and AT&T database

The results show better acceptance rates for the xm2tvs compared to the AT&T database. Illumination levels largely dictate the effectiveness of feature extraction. Images in the AT&T library are slightly lower quality of illumnitation than those in the xm2tvs library which could explain the lower acceptance rates of the results.

Show recognition results and thresholds. Discuss false negative matches, why and how can be countered.

## Discussion

Some data is being lost when computing the Eigen faces. It is suspected that during the normalization process variables of type double are being converted to integers resulting in the rounding of decimal numbers and thus this loss of data that can be seen. This small loss of data has a minimal effect on the face recognition results although could be refined to optimise the algorithms effectiveness.

The strength of a stand-alone face recognition system used as a main authentication tool doesn’t appear to be the safest method of computer recognition as the face detection can be spoofed.

When opencv was ported too javaCV the contrib module was not part of the implementation. This module contained useful functionlkity for FaceRecognize class. A JNI (Java Native Library) could be used to access opencv functionality in c++ although this would be more resource intensive

Developing a custom PCA class consumed more project time than originally designated. This had an effect on the amount of extra features that could have been implemented in the system.

## Conclusion

This system can be fooled by holding an image which is then successfully detected. So if an unknown user is attempting to login they would simply have to find a reasonable high resolution image of the user they wish to impersonate and present this to the face detection process whilst producing a login image.

## Project Commentary

## Social, Legal, Health & Safety and Ethical Issues

There are many ethical arguments to the analysis and storage of biometrical data. Biometrics uniquesly identify a person with biometric data and can’t be changed easily. If your authentication password or forms of identity such as bank card become compromised they can easily be changed although human like features such as the face are much harder to modify.

The ethical and social use of face recognition has been questioned much already as the public feel that their civil liberties and privacy are being invaded by the continuous surveillance and gathering of public data. With the ever growing technical strive to innovate methods of human computer interaction biometric analysis.

Inanimate images/photos can be held in front of the camera and a face is successfully detected. Open CV’s trained classifiers are robust enough to detect images with very little features. This poses a security issue as user recognition can be spoofed without the necessary precautions being implemented to counter such forgery.

## Reflection

Naturally the originally determined success criteria in the PID document are very general and needed further definition.

To ensure a reliable and secure authentication system scenarios such as user image spoofing where the unknown user attempting to log in holds up an image of a known user must be consider.

The project deliverable defined in the PID where unrealistic. Which ones why, what took up time.

JNI (Java Native Interface) can be used to utlise C++ code. The native C++ openCV library offers a very robust FaceRecognizer class that can determine the age and gender of a user. It also features live detection functionality that can differentiatre between a live and inanimate subject. This functionality cleverly exploits the movement of eyes and skin tone changes to conclude whether the user is a real person [ref to live detctor]

## References

[1] Jose Pagliery. (2014). *FBI launches face recognition system.* Available: http://money.cnn.com/2014/09/16/technology/security/fbi-facial-recognition/. Last accessed 12th Jan 2016.

[2] Reza Azad, Babak Azad, Nabil Belhaj Khalifa, Shahram Jamali. (2014). Real-Time Human-Computer Interaction Based on Face and Hand Gesture Recognition. *International Journal in Foundations of Computer Science & Technology*. 4 (4), 11.

[3] OpenCV Developers Team. (2016). *About OpenCV.* Available: http://opencv.org/about.html. Last accessed 13th Jan 2016.

[4] Hewlett-Packard Development Company. (2015). *HP IDOL.* Available: https://my.vertica.com/docs/IDOL/Servers/IDOLServer/10.9/Guides/html/Expert/index.html#IDOLExpert/Improve/FaceDetect\_factors.htm. Last accessed 21st Jan 2016.

[5] OpenCV Developers Team. (2016). *OpenCV 3.0.0.* Available: http://docs.opencv.org/java/3.0.0/. Last accessed Feb 2nd 2016.

[6] Oracle Developers Team. (2011). *Oracle Java Documentation.*Available: http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784. Last accessed Feb 2nd 2016.

[7] Open CV Dev Team. (2016). *Cascade Classification.* Available: http://docs.opencv.org/2.4/modules/objdetect/doc/cascade\_classification.html. Last accessed Feb 10th 2016.

[8] The MathWorks, inc. (2016). *AdaBoost.* Available: http://uk.mathworks.com/discovery/adaboost.html?refresh=true. Last accessed Feb 20th 2016.

[9] Wikipedia. (2016). *Viola–Jones object detection framework.* Available: https://en.wikipedia.org/wiki/Viola%E2%80%93Jones\_object\_detection\_framework. Last accessed Feb 20th 2016.

[10] Open CV Dev Team. (2016). *Face Recognizer.* Available: http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec\_api.html. Last accessed Feb 21th 2016.

[11] sis26@drexel.edu. (Unknown). *Eigen Face Tutorial.* Available: http://www.pages.drexel.edu/~sis26/Eigenface%20Tutorial.htm. Last accessed Feb 27th.

[12]

[] Network World, US intelligence wants to radically enhance facial recognition software [online]. Available at : <http://www.networkworld.com/article/2225788/applications/us-intelligence-wants-to-radically-advance-facial-recognition-software.html>

[DDFD]

[conigtec] <http://www.cognitec.com/research.html>

[JavaFX]

[MVC] Model View Controller

[bioID]

[Windows Hello]

[Black Hat]

[Def Con]

[retinal hack black hat]

[Apache commons library]

[Jama Library]

[OpenCv Core Mod]

## Appendices

### Appendix 1

PID