JAVA BASED FACE RECOGNITION SYSTEM

MASTERS PROJECT

Submitted in the partial fulfillment of the requirements
of the degree of Master of Science

by

Kalpana Pal
palk1@mail.montclair.edu

Advisor: Dr. Stefan Robila
Department of Computer Science
Montclair State University
Montclair, NJ
11 Dec 2008

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to all those at Montclair state University who contributed towards an enriching educational experience I had during my Masters a Montclair State University. I would like to specially thank Dr. Stefan Robila for helping me and guiding me throughout the project with his expertise in the area of Pattern Recognition.

I would like to express my appreciation to Dr. James Benham who encouraged me deciding to pursue a Master's Project. Many thanks to other members of graduate committee, Drs. Wang, Jenq, and Antoniou in the department of Computer Science for their incredible support and a good learning experience they gave me throughout my studies at MSU.

I also would like to thank Dr. Roman Zaritski for being my advisor in first year of my research assistantship at MSU. I wish to acknowledge Department of Computer Science, MSU for their financial support.

Special thanks to my husband and my family whose love and support made everything possible.

ABSTRACT

Facial recognition systems are pattern recognition based applications, which are widely used in many real world applications. They play important roles in the areas of security systems, biometrics, law enforcement, surveillance, smart cards, and many more. Apart from its wide use at commercial level, in recent times, face recognition algorithms have continued to be developed and used in research works related to image analysis and recognition.

In this project, we aim to implement a real time facial recognition system using Java programming language, JMF (Java Media Framework) library, and JAMA (A Java Matrix Package). The Oracle 10g database is used to store the set of training face images as BLOBs (Binary Large Objects). The system includes an off-the-shelf webcam and will use a face recognition algorithm based on eigenfaces.

This application runs a simple GUI (Graphical User Interface), which shows a live video from a regular webcam, and allows users to capture a snapshot and compare it (using eigenface-based face detection) against the database of saved faces. The application also allows user to add additional captured images to the database. At any time, the user has the choice to perform Principal Component Analysis (PCA) computations.

Keywords:

Face Recognition, Eigenface, Eigenvectors, Principal Component Analysis, Image Capture, Image Processing, Covariance Matrix, Coefficient Vectors.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivation

The motivation behind this project was Dr. Robila with whom I took a class of Pattern Recognition last year. He inspired me with the idea of working on face recognition and how it is being widely used in the areas such as, security systems, biometrics, law enforcement, surveillance, smart cards, and many more on commercial level. Development of the facial recognition system has helped me to understand and appreciate its usage in various areas. During this project, I have acquired valuable skills in the areas of image capture, database processing as well as the pattern recognition. This project has led to a fully functional application package that will be made available for download.

1.2 Project Overview

The face recognition problem can be described as follows: "*Given images of a scene, identify or verify one or more persons in the scene using a stored database of faces*" [6]. The basic strategy is to identify a set of facial features in a digitally captured image of a person's face and compare it against a set of facial database [8, 9]. There exist many algorithms for image capture and facial recognition implementation. The procedure for face image capturing through Webcam is referenced from existing Java codes available online [10, 11]. Also, the algorithm for face matching against a data set of images is referenced from Eigenface based face recognition algorithm [12].

Overview of the procedure

Step 1. Display live video

A Logitech QuickCam Pro 3000 camera shown in Figure 1.1 is used to display a live video of the scene. The software uses a MediaLocator object to get the media device information connected to the system and calls startPlayer(MediaLocator ml) function to display a live video.

Figure 1.1: A picture of Logitech QuickCam Pro 3000

The software displays the live video in a Java Frame as shown in Figure 1.2 using Java Media Framework class libraries [2]

Figure 1.2: Live video displayed in JFrame

Step 2. Capture a snapshot

The software then allows user to capture a snapshot and save it to the local disk. I modified existing software freely available online [10, 11] to crop the captured image and save it to local disk [23]. A snapshot of face image saved as a JPG images on local disk is show in Figure 1.3:

Figure 1.3: Snapshot captured through webcam

Step 3. Add snapshot image to Face database

The software allows user to add the snapshot image saved on local disk to the Face database where rest of the training images are stored. All the training face images are stored in face database in BLOBS table. Figure 1.4 shows the training data set of 40 images obtained online [20].

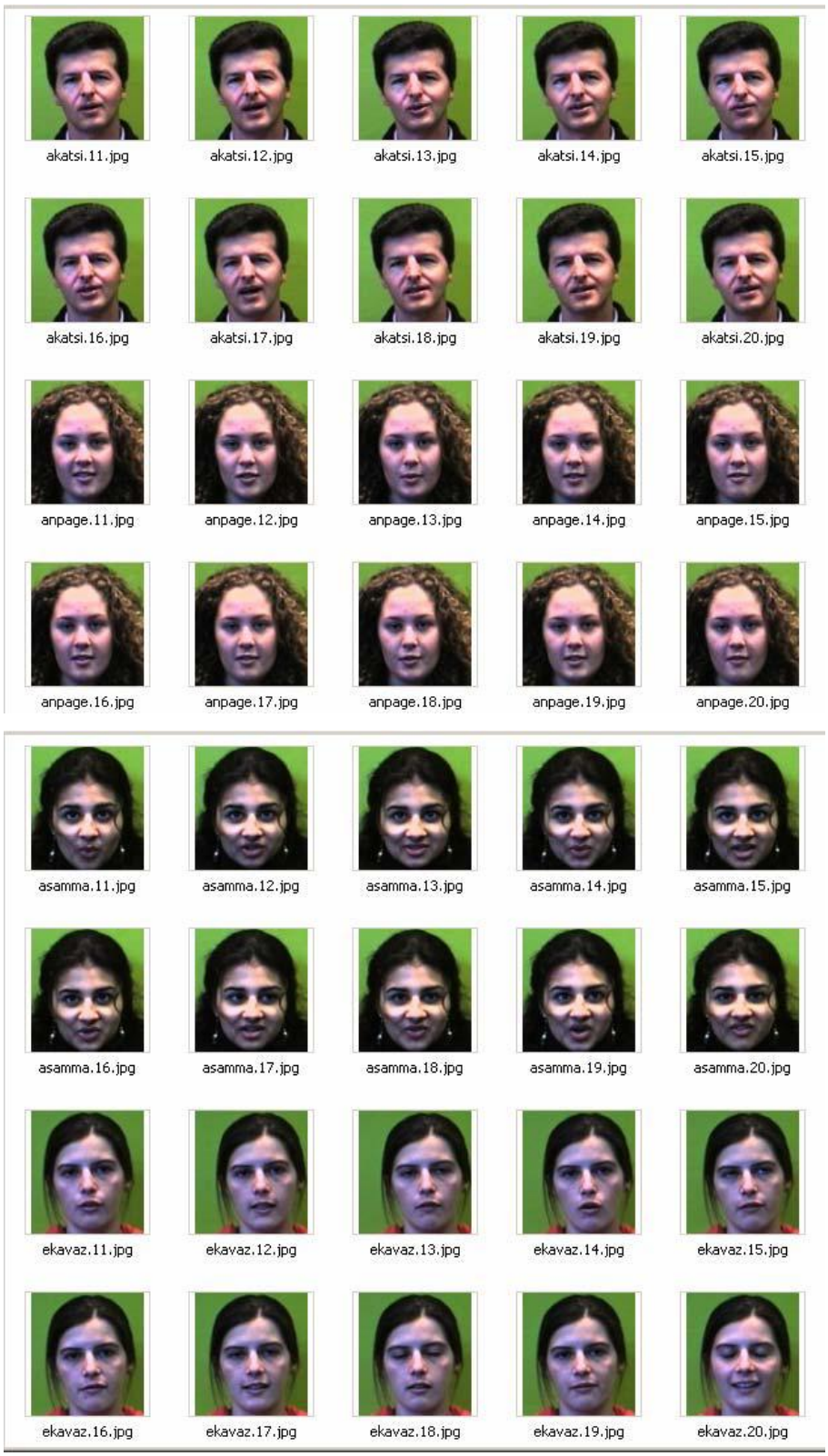

Figure 1.4: 40 images stored in face database

Step 4. Compute eigenfaces and the weight coefficient vectors for training face images

The training set of JPG face images is stored as Binary Large Objects in Face database. Eigenfaces and weight coefficients are then extracted out of the training set of face images using the Principal Component Analysis (PCA). The eigenfaces and coefficient vectors are the principal components in a face space. Below are the steps needed to obtain the eigenfaces and the coefficient vectors:

i. Compute the average image and normalize them

ii. Compute the covariance matrix for the normalized images

iii. Compute the eigenvectors with the highest eigenvalues obtained from covariance matrix

iv. Choose four eigenfaces from eigenvectors with highest eigenvalues

v. Compute coefficient vectors for training images

For each subject in our database, we next compute the average coefficient vector by averaging the coefficient vectors for all the images corresponding to the subject. This information is stored in the database.

Step 5. Identify the snapshot against the face database

Compute the coefficient vectors (**w**) for test image. Calculate the Euclidean distance of test coefficient vector from average coefficient vector for each subject in the database. The smaller the distance, the closer the test image is to the face image in the training set.

1.3 Scope

The scope of this application can be extended up to a commercial face recognition application. Increasing the database size of face images can significantly decrease the execution time of application in real time. Since, in this application, the face images are stored in Oracle database, a large number of images can be stored without sacrificing the execution time by implementing the indexing feature of Oracle database. Indexes can be created on the database tables, which can significantly decrease the retrieval time of the face images and coefficient vectors. In addition, the security features of Oracle database can be implemented on the database for restricted access to the Oracle Face database.

Further improvements can be done in the Oracle Face database design by storing the average face and eigenface vectors to the Oracle Face database instead of storing them externally in text files on local disk because of their large size. These files store 36000 double values to represent an average face vector and 4 X 36000 double values to represent four eigenface vectors. These array vectors can be stored in Oracle database using VARRAYS [24].

A web enabled GUI interface can also be implemented in future work for this application. The application can be modified to recognize most of the commercially used sophisticated Webcams for image capturing.

CHAPTER 2

REQUIREMENTS

2.1 Functional Requirements

Functional requirements of the application are shown below:

1. It should be able to recognize and connect to the Webcam attached to the machine and display the live video

2. It should be able to capture the snapshot of the live video and save it to local disk

3. It should be able to add the face images to the Face database as BLOBs object

4. It should be able to store the face images in Face database in Oracle

5. It should be able to compute the average face vector, eigenvalues, eigenvectors, eigenfaces, and coefficient vectors for all training images and store them representing a face-space

6. It should be able to identify the test image against database of face images by computing the smallest Euclidian distance

2.2 System Requirements

System requirements of the application are shown below:

Operating System: Windows XP

Database: Oracle 10g Personal/Standard/Enterprise Edition

Tools: Logitech QuickCam Pro 3000

Language: JAVA, PL/SQL

Librareis: JMF (Java Media Framework), JAMA (A Java Matrix package)

CHAPTER 3

DESIGN

3.1 Execution Flow

Execution flow of the application is depicted in Figure 3.1 below:

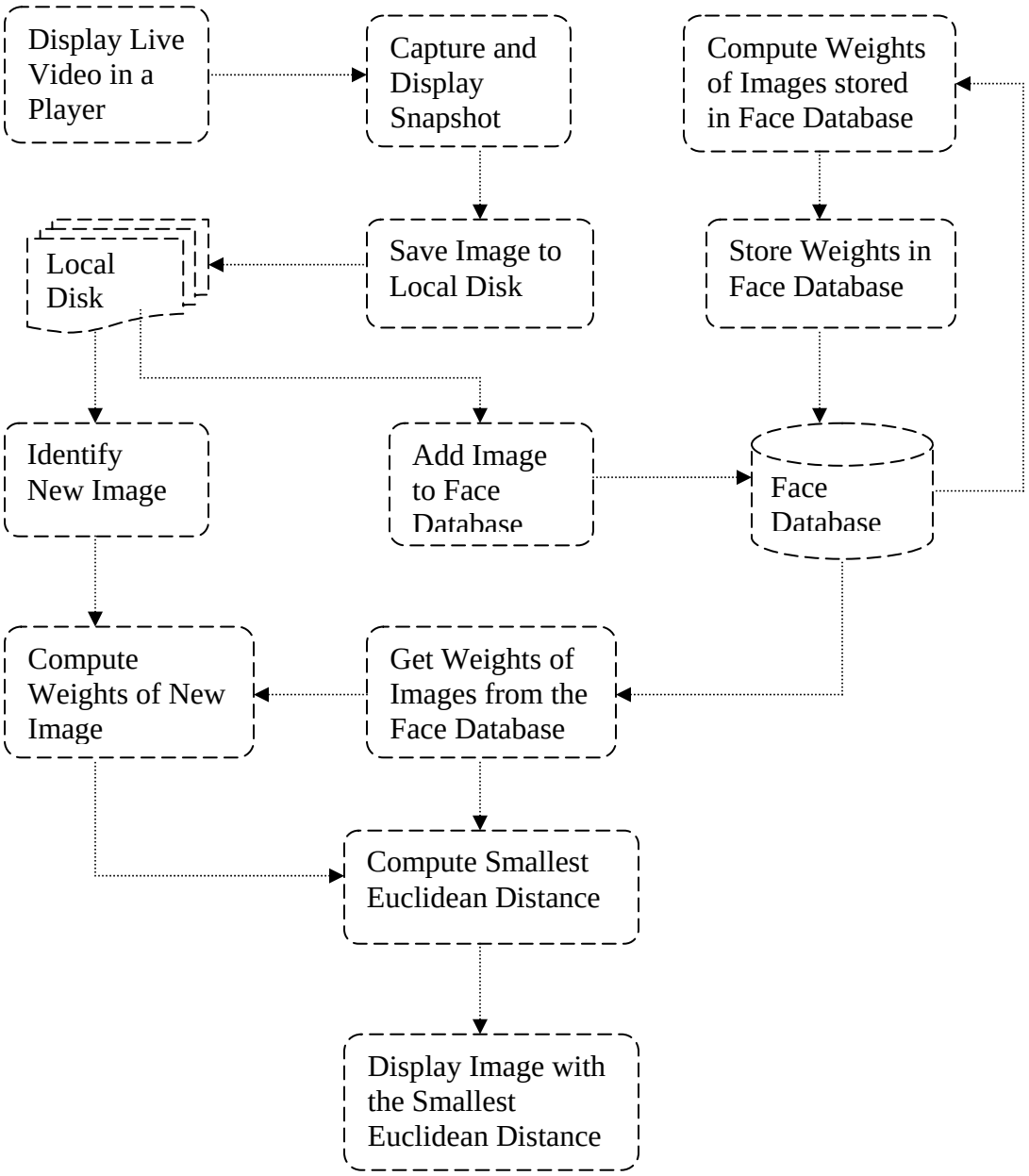

Figure 3.1: Data Flow Diagram

3.2 Database Design

The facial database consists of one master table: BLOBS and four children:

COEFF_VECTOR
AVG_COEFF_VECTOR
avgFace.txt
eigVector.txt

All the images captured through Webcam are stored in BLOBS table. Two child tables COEFF_VECTOR, and AVG_COEFF_VECTOR are populated with weights for each image during runtime dynamically when the program does the PCA computations.

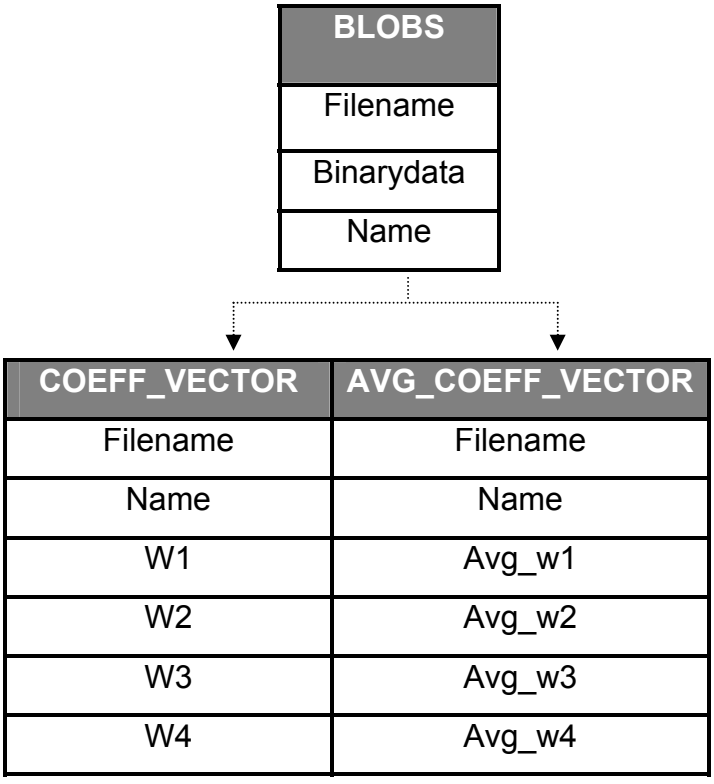

Figure 3.2: Tables in Face database

The other two children, avgFace.txt and eigVector.txt, are stored as text files on local disk. The avgFace.txt file represents average face vector of all the 40 images stored in BLOBS table. It stores 36000 double values to represent an average face vector. Similarly, the eigVector.txt file represents four eigenface vectors with highest eigenvalues. It stores 4 X 36000 double values to represent four eigenface vectors. Since Oracle has a limitation to store maximum 1000 column, ; these large vectors were stored externally in a text file.

avgFace.txt: The avgFace.txt is created during runtime when computation of average face vector is done. It is populated with 36000 double values for average face vector (Figure 3.2.2).

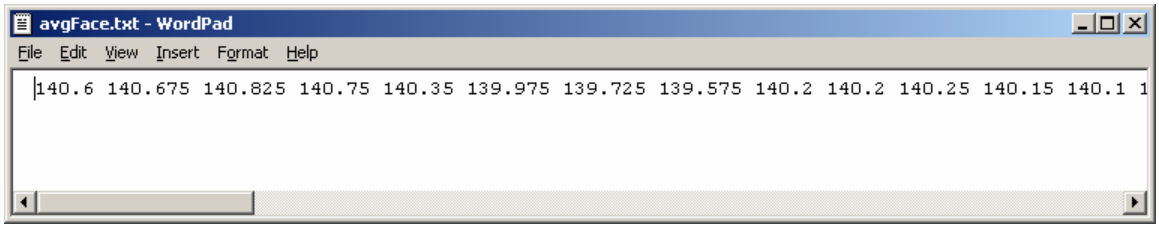

Figure 3.3: avgFace.txt to store double values of average face vector

eigVector.txt: *The* eigVector.txt is created during runtime when computation of eigenvectors is done. It is populated with 4 X 36000 double values for corresponding four eigenface vectors (Figure 3.2.3).

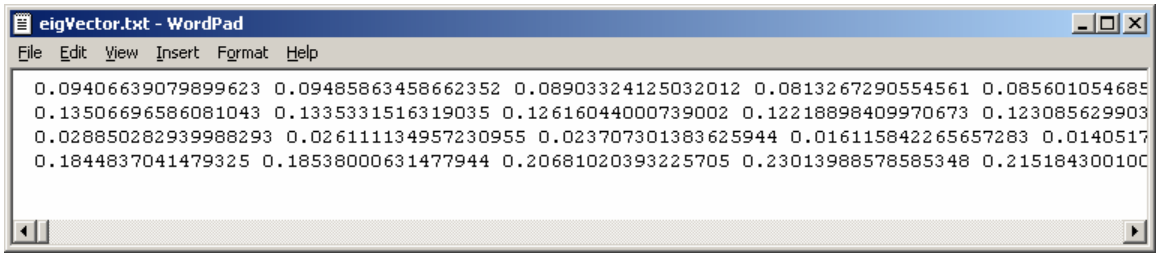

Figure 3.4: eigVector.txt to store double values of four eigenface vectors

Tables Description and Structure

BLOBS

Table Description:

When an image captured through webcam is added to the Face database, it is stored in BLOBS master table. It stores all face images in 'BINARYDATA' column as BLOBS (Binary Large Objects) in a table with the filename and name of the person for the corresponding binary image. Forty (40) training images are stored in this table for testing the results.

Table creation SQL script:

```
SQL> CREATE TABLE BLOBS (FILENAME VARCHAR2(30), BINARYDATA
     BLOB, NAME VARCHAR2(20);
```

BLOBS Structure:

No	Column Name	Data Type	Length	Description
1	FILENAME	VARCHAR2	30	Saved image name
2	BINARYDATA	BLOB	<Default>	Image stored in binary
3	NAME	VARCHAR2	20	Name of the person

Table 3.5: BLOBS table structure

Note:

BLOBS data type in Oracle is used to store unstructured binary data (such as text, graphic images, video clips, and sound waveforms) [21]. This data type can store up to 10 terabytes of binary data. The length for column 'BINARYDATA' is assigned internally.

Sample of records in BLOBS data:

FILENAME	NAME
akatsi.11.jpg	akatsi
ekavaz.18.jpg	ekavaz
ekavaz.20.jpg	ekavaz

Table 3.6: Sample Records for the BLOBS table data. Column FILENAME cannot be displayed because of its binary nature

COEFF_VECTOR

Table Description:

Stores four weights of each coefficient vector for the corresponding image along with the image name and name of the person to which image belongs. Forty (40) records are stored. Each record stores image filename, name of the person, and four weights of the coefficient vector to the corresponding image stored in master table BLOBS.

Table creation SQL script:

```
SQL> CREATE TABLE COEFF_VECTOR (FILENAME VARCHAR2(30), NAME
     VARCHAR2(20), W1 FLOAT, W2 FLOAT, W3 FLOAT, W4 FLOAT);
```

COEFF_VECTOR Structure:

No	Column	Data Type	Length	Description
1	FILENAME	VARCHAR2	30	Saved image name
2	NAME	VARCHAR2	20	Name of the person
3	W1	FLOAT	126	Weight of 1st Coefficient
4	W2	FLOAT	126	Weight of 2nd Coefficient
5	W3	FLOAT	126	Weight of 3rd Coefficient
6	W4	FLOAT	126	Weight of 4th Coefficient

Table 3.7: COEFF_VECTOR table structure

Sample of records in COEFF_VECTOR data:

FILENAME	NAME	W1	W2	W3	W4
akatsi.13.jpg	akatsi	268980.523	40195.1627	147244.246	166403.523
ekavaz.19.jpg	ekavaz	108121.301	14684.2928	64727.4818	53729.6304
akatsi.12.jpg	akatsi	266377.857	39333.989	151518.566	167731.909

Table 3.8: Sample records from the COEFF_VECTOR table data

AVG_COEFF_VECTOR

Description: Store four records representing average coefficient vectors for four persons. Training Face database has images of four persons, so four average coefficient vectors (each average coefficient vector has four weights: AVG_W1, AVG_W2, AVG_W3, AVG_W4) are computed from the images of each person. Each average coefficient vector is stored along with the name of the person and its 1st image name.

Table creation SQL script:

SQL> CREATE TABLE COEFF_VECTOR (FILENAME VARCHAR2(30), NAME VARCHAR2(20), W1 FLOAT, W2 FLOAT, W3 FLOAT, W4 FLOAT);

AVG_COEFF_VECTOR Structure:

No	Column	Data Type	Length	Description
1	FILENAME	VARCHAR2	30	Saved image name
2	NAME	VARCHAR2	20	Name of the person
3	AVG_W1	FLOAT	126	Weight of 1st Average Coefficient
4	AVG_W2	FLOAT	126	Weight of 2nd Average Coefficient
5	AVG_W3	FLOAT	126	Weight of 3rd Average Coefficient
6	AVG_W4	FLOAT	126	Weight of 4th Average Coefficient

Table 3.9: AVG_COEFF_VECTOR table structure

AVG_COEFF_VECTOR Data Records:

FILENAME	NAME	AVG_W1	AVG_W2	AVG_W3	AVG_W4
asamma.20.jpg	asamma	228821.18	76628.6221	199288.014	216843.27
anpage.20.jpg	anpage	137002.271	21898.7002	24733.4082	8860.25494
akatsi.20.jpg	akatsi	251437.281	41816.645	146099.711	168420.729
ekavaz.20.jpg	ekavaz	114386.17	14952.0711	77921.7108	57282.7961

Table 3.10: AVG_COEFF_VECTOR table data with four records representing four eigenface vectors

CHAPTER 4

FACE DETECTION USING EIGENFACES

4.1 PCA Approach

Principal Component Analysis (PCA) is a statistical approach, which is commonly used in many face recognition systems for face identification. The basic approach of PCA is to extract relevant information (Principal Components) in a feature space by reducing the large dimensionality of data space. The same approach can be implemented in face recognition by reducing the dimensionality of a data set of facial images from 2-D to 1-D and construct eigenfaces using the eigenvectors (Principal Components) that correspond to the highest eigenvalues [8].

4.2 Eigenvectors and Eigenvalues

In image processing, processed images of faces can be seen as vectors whose components are the brightness of each pixel. The dimension of this vector space is the number of pixels [16]. Every eigenvector has a value associated with it, which is called as eigenvalue. When an eigenvector, **u**, is multiplied by a square matrix, for example **M**, the result is always an integer multiple of that vector. This integer value is the corresponding eigenvalue of the eigenvector. This relationship can be described by following equation

$$\mathbf{M} \times \mathbf{u} = \lambda \times \mathbf{u} \qquad (4.2.1)$$

where λ is the eigenvalue of eigenvector, u which is an eigenvector of the matrix **M** [17]. Principal eigenvectors are those, which have the highest eigenvalues [8].

4.3 Eigenfaces

“Eigenfaces refer to an appearance-based approach to face recognition that seeks to capture the variation in a collection of face images and use this information to encode and compare images of individual faces in a holistic (as opposed to a parts-based or feature-based) manner” [5]. The eigenvectors of the covariance matrix associated to a large set of normalized pictures of faces are called eigenfaces [16]. Figure 4.3.1 shows the eigenfaces with the highest eigenvalues obtained from the training set of 40 images.

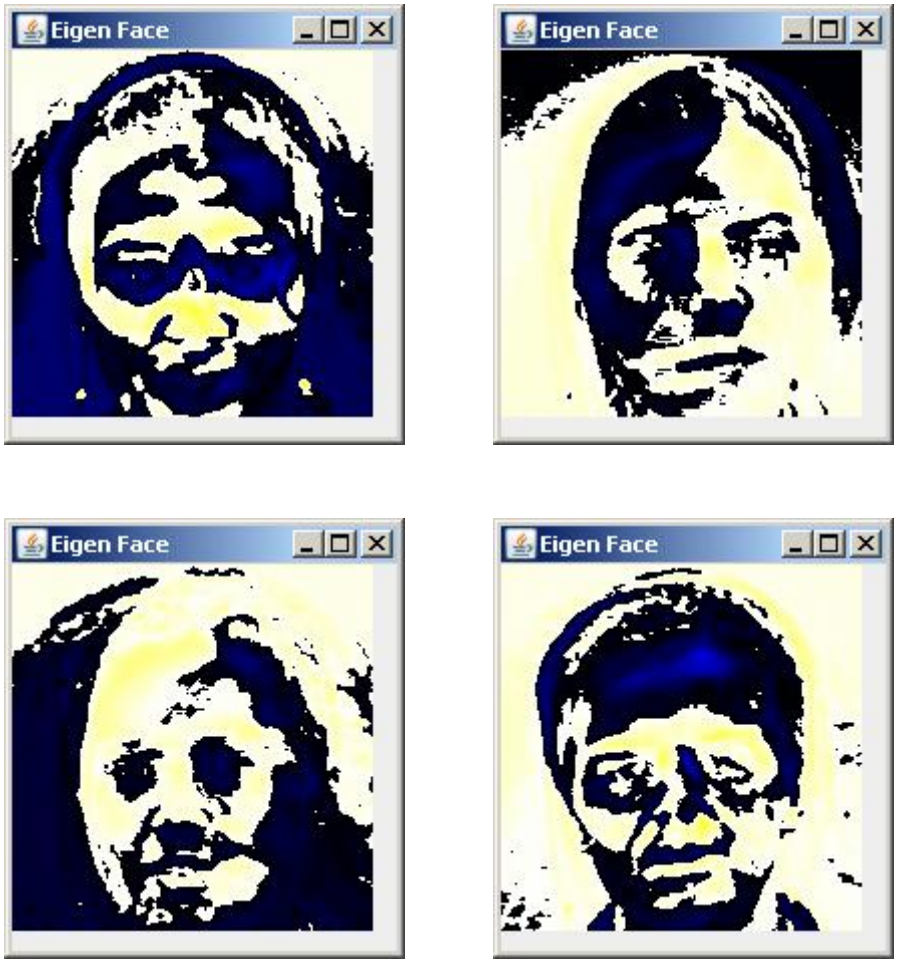

Figure 4.1: Four eigenfaces with the highest eigenvalues obtained from the training set of 40 images

Below are the steps to obtain the eigenfaces and the coefficient vectors from the training set of face images:

(i) Compute average face image (Ψ) from training set of face images

Given a set of m = 40 training images $\{\Gamma_1,\Gamma_1,\ldots,\Gamma_m,\}$ we compute the average face image [12]:

$$\Psi = \frac{1}{m}\sum_{i=1}^{m}\Gamma_i \qquad \text{where i = 1 … m} \qquad (4.1)$$

Figure 4.2: Average face of all images in the training data set

(ii) Normalize all the training face images

Subtract the average face from each face to normalize them:

$$\Phi_i = \Gamma_i - \Psi \qquad \text{where i = 1 … m} \qquad (4.2)$$

*(iii) Compute Covariance matrix (**C**)*

Calculate the covariance matrix for the training data:

$$C = \frac{1}{2} m \sum_{i=1}^{m} \Phi_i \Phi_i^T = AA^T \quad (4.3)$$

where:

$$A = \{\Phi_1, \Phi_2, ..., \Phi m\} \quad (4.4)$$

(iv) Compute Eigenfaces (E) of Covariance matrix (C)

Find eigenvectors and eigenvalues for *C*, and choose the *e* = 4 eigenvectors with the highest associated eigenvalues. Combine the normalized training set of images to produce *e* eigenfaces. Store these eigenfaces for later use.

The eigenvalues and eigenvectors of Covariance matrix C are λ and U = A V $\lambda^{-1/2}$, where U = $\{u_i\}$ is the collection of four eigenfaces for i = 1… e.

*(v) Compute Coefficient vectors (**Ω**) and average coefficient vectors for training images and new test image*

Compute coefficient vector **Ω** = (ω_1, ω_2, …,ω_e) for each image Γ_i and store these e weight coefficients for each image in database by computing the dot product of the normalized image with each selected Eigenface:

$$w_i = E_i^T * (\Gamma - \Psi) \text{ where i = 1 … e} \quad (4.5)$$

Compute coefficient vector (ω_1^{Test}, ω_2^{Test}, …. , ω_e^{Test}) for the new test image Γ^{Test} in a similar fashion as in equation 4.3.5.

(vi) Compute average coefficient vectors for each person in training set

For each person in our Face database, compute the average coefficient vector ($\varpi_1, \varpi_2, \ldots, \varpi_e$) by averaging the weights of coefficient vectors for all the images corresponding to the person stored in the Face database.

4.4 Euclidean Distance

The Euclidean distance between coefficient vectors of two images in a face space provides a measure of similarity between the corresponding two images. The smaller the Euclidean distance, the closer the images are.

Since, the training set of face images stores images of four persons in Face database; compute average coefficient vector for four persons as discussed in step (vi) of section 4.3. For each person, compute Euclidean distance between the Test Coefficient vector ($\omega_1^{Test} \ldots . \omega_e^{Test}$) and Average coefficient vector ($\varpi_1, \varpi_2 \ldots \varpi_e$) as shown below:

$$\text{Euclidean Distance} = \sqrt{(w_1^{Test} - \varpi_1)^2 + \ldots + (w_e^{Test} - \varpi_e)^2} \quad (4.6)$$

Now, we get Euclidean distance for four persons with the test image and the person that gets the smallest value matches the test image.

CHAPTER 5

IMPLEMENTATION

5.1 Class Diagram

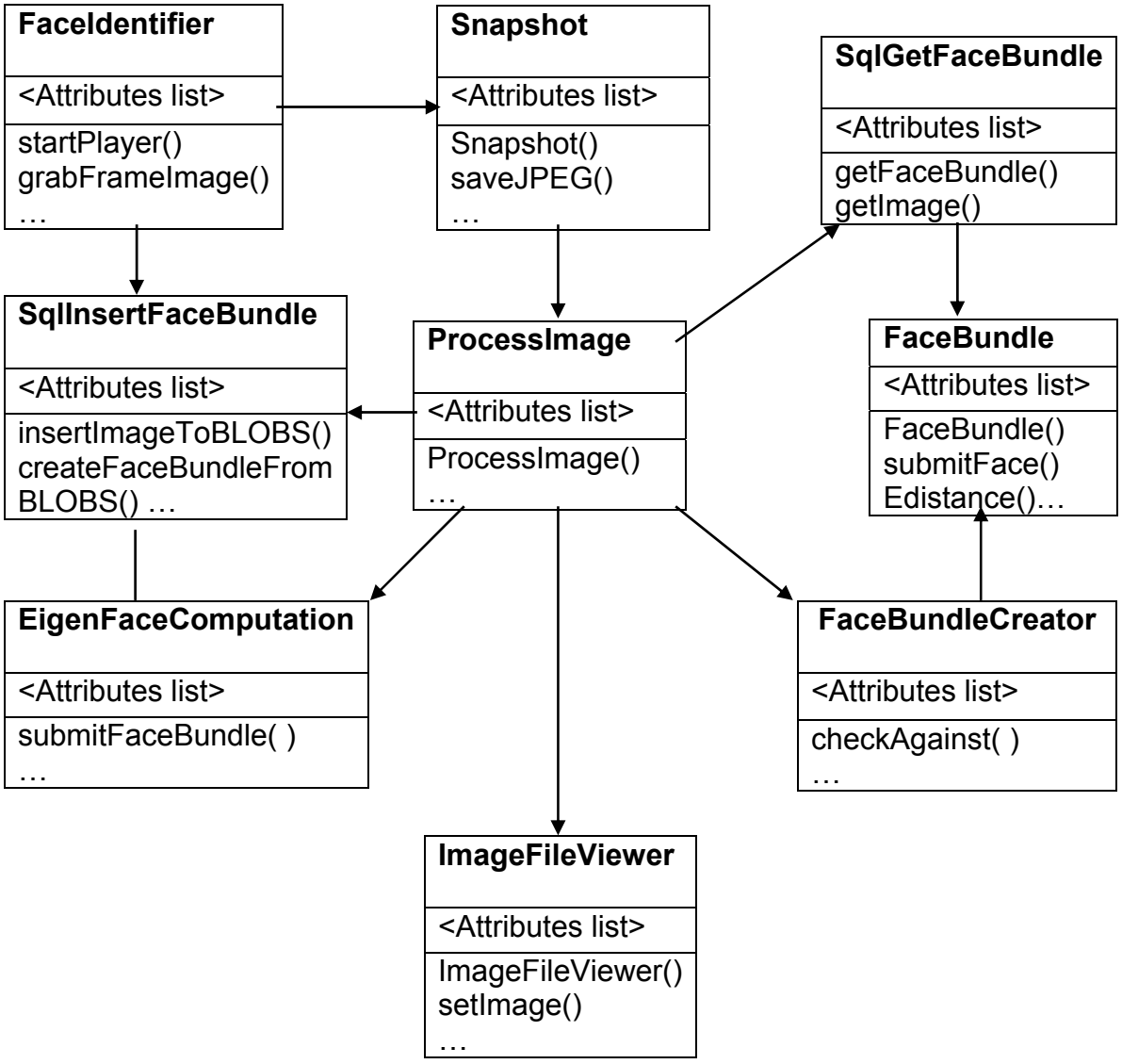

Figure 5.1: Class Diagram

5.2 Class Description

The code consists of nine main classes discussed below which further calls methods and Java classes from JMF and JAMA packages:

FaceIdentifier
Snapshot
ProcessImage
SqlInsertFaceBundle
EigenFaceComputation
SqlGetFaceBundle
FaceBundle
FaceBundleCreator
ImageFileViewer

FaceIdentifier

Subclasses and functions listing:

Snapshot
SqlInsertFaceBundle
main()
startPlayer()

The main() function in the FaceIdentifier class calls a JMF (Java Media Framework package) classes such as , CaptureDeviceInfo and MediaLocator, to get the information about the media player and create a DataSource for this device. The main() function calls startPlayer() function and passes a MediaLocator object to it. Given a MediaLocator, the startPlayer() creates an instance of Processor (another JMF class) as a player to playback the media. Once the media player is working, it displays the live video in a JFrame.

It also calls another function getImage() in class SqlGetFaceBundle to retrieve the images stored in the database for computation.

Snapshot

Subclasses and functions listing:

ProcessImage
Snapshot()
saveJPEG()

Snapshot class extends a JFrame to display the snapshot and allows the user to capture a face snapshot and save it to the local disk. It calls ProcessImage class to do the further processing of the saved image.

ProcessImage

Subclasses and functions listing:

SqlInsertFaceBundle
SqlGetFaceBundle
ImageFileViewer
ProcessImage()

ProcessImage extends a JFrame to display the options to process the image. It initializes the JFrame in ProcessImage() constructor. It has options for the user to add the image to Face database, identify the image, or exit the current JFrame. SqlInsertFaceBundle class is instantiated when user chooses to add the image to face database. SqlGetFaceBundle class is instantiated when user wants to

identify the unknown image. It passes the unknown image to checkAgainst() function in class FaceBundleCreator to test captured face image against the face images stored in the database.

SqlInsertFaceBundle

Subclasses and functions listing:

EigenFaceComputation
insertImageToBLOBS()
insertImageCoeffVector()
computeAvgCoeffVector()
createFaceBundleFromBLOBS()

SqlInsertFaceBundle class directly interacts with Oracle Face database through JDBC connectivity. It calls a function insertImageToBLOBS() to insert saved snapshot as a Binary Large Object (BLOB) to the Oracle Face database.
It also calls a function createFaceBundleFromBLOBS() to submit stored training set of face images to class EigenFaceComputation to create face space. The two functions, insertImageCoeffVector(), and computeAvgCoeffVector(), are used to insert coefficients and average coefficient vectors to Face database.

EigenFaceComputation

SqlInsertFaceBundle
submitFaceBundle()

EigenFaceComputation class calls the submitFaceBundle() function to pass all the face images as 2-D double array face-space. An average face is computed

from th2 2-D double array of faces to normalize the face images. Eigenvalues and eigenvectors are computed from the Covariance matrix derived from face-space. JAMA package classes, and functions such as Matrix and EIgenvalueDecomposition are used to do the above computation and get the eigenface with highest eigenvalues. The submitFaceBundle() also calls functions insertImageCoeffVector(), and computeAvgCoeffVector() in SqlInsertFaceBundle class to insert coefficients and average coefficient vectors to Face database.

SqlGetFaceBundle

FaceBundle
getImage()
getFaceBundle()

SqlGetFaceBundle class directly interacts with Oracle Face database through JDBC connectivity. Function getImage() retrieves the images stored as Binary Large Objects in the BLOBS table. Function getFaceBundle() retrieves all the relevant information required to match the unknown test image such as eigenvectors, coefficient vectors and average coefficient vectors. It then returns a FaceBundle object with all the above required information to identify an unknown image.

FaceBundle

FaceBundle()
submitFace()

FaceBundle() constructor is initialized to store all the relevant information such as average face, average coefficient vector, and eigenvector, for a face-space returned from SqlGetFaceBundle.getFaceBundle(). The submitFace() function

gets the test face image as an argument and computes coefficient vectors for the test image. It then computes the Euclidean distance of test image coefficient vector from image coefficient vectors for all the images in the face-space. It returns the image id with the smallest Euclidian distance.

FaceBundleCreator

FaceBundle
checkAgainst()

Function checkAgainst() in FaceBundleCreator accepts FaceBundle object as an argument and calls submitFace() function from class FaceBundle. submitFace() accepts test image as an argument and returns the id of the image from within the face-space which is closest in Euclidean distance with the test image captured through Webcam. If smallest distance is above the threshold value, then the test image does not match any image in Face database.

ImageFileViewer

ImageCanvas
ImageFileViewer()
setImage()

ImageFileViewer class extends JFrame to display the JPG image. It creates an ImageCanvas which extends a Canvas to display the image. The setImage() function passes the JPG image or double array as an argument to ImageCanvas to display the image in a Canvas.

CHAPTER 6

EXPERIMENTAL RESULTS

6.1 Application Testing

The application is tested using face data set available online. Total 40 images are taken from this data set and inserted into BLOBS table in Oracle Face database. These 40 images belong to 4 persons: Anpage, Asamma, Ekavaz, Akatsi, where each person has 10 face images with slightly varying facial expressions. All the images are of same size 180X200 and have same background and lighting conditions [20].

The application is tested with 20 new face images which are taken from the same data set as above [20]. First 10 out of 20 images belong to 4 persons whose images exist in Face database are tested to identify them against images in Oracle Face database. Rest of the 10 face images belongs to 4 persons whose images do not exist in Oracle Face database.

No.	Image Name	Smallest Euclidean Distance by Average Coefficients	Result	Smallest Euclidean Distance by Coefficients	Result
1	test_anpage.8.jpg	0.03	Correct	0.02	Correct
2	test_asamma.3.jpg	0.21	Correct	0.16	Correct
3	test_asamma.2.jpg	0.22	Correct	0.16	Correct
4	test_anpage.10.jpg	0.02	Correct	0.01	Correct
5	test_ekavaz.7.jpg	0.29	Correct	0.13	Correct
6	test_akatsi.18.jpg	0.03	Correct	6.90E-17	Correct
7	test_ekavaz.10.jpg	0.29	Correct	0.16	Correct
8	test_akatsi.7.jpg	0.04	Correct	0.01	Correct
9	test_akatsi.1.jpg	0.07	Correct	0.48	Correct
10	test_anpage.15.jpg	0.009	Correct	3.75E-17	Correct
11	test_admars.1.jpg	0.747	Correct	0.506	Correct

12	test_ajones.11.jpg	0.472	Correct	0.349	Correct
13	test_elduns.1.jpg	0.086	Incorrect	0.055	Incorrect
14	test_mjhans.2.jpg	0.384	Correct	0.33	Correct
15	test_klclar.4.jpg	0.086	Incorrect	0.074	Incorrect
16	test_cchris.1.jpg	0.184	Incorrect	0.108	Incorrect
17	test_kdjone.1.jpg	0.585	Correct	0.533	Correct
18	test_jabins.1.jpg	0.43	Correct	0.39	Correct
19	test_njmoor.1.jpg	0.393	Correct	0.406	Correct
20	test_ccjame.1.jpg	0.47	Correct	0.43	Correct

Figure 6.1: Test results of 20 face images belong to 4 persons. First 10 images exists in Oracle Face database and last 10 do not exists in Oracle Face database

6.2 Testing Results

Eighty Five percent correct results are achieved as shown above in Figure 6.1. 17 out of 20 images are correctly identified. Rest of the three incorrectly identified images do not exist in the Oracle database. Their Euclidean distance fall below the empirically determined threshold value 0.3; hence they were incorrectly matched to the image in the database with the smallest Euclidean distance.

CHAPTER 7

COMPUTATIONAL TIME ANALYSIS

Figure 7.1 shows the computation time to perform the PCA operations, add an image to the Face database and compute coefficient vector for 40 training images.

System configuration used for time measurement:

Operating System: Windows XP Home Edition Version 2002 Service Pack 2
Processor: Intel (R) Pentium (R) M
Processor speed: 1.60 GHz
RAM: 1 GB

Face database of 40 images of size 180X200	Elapsed Time (Seconds)
PCA operations (computation of Covariance matrix, Eigenfaces, Coefficient vectors)	5.428
Add a new face image of size 180X200	0.12
Identify an unknown image of size 180X200 by average coefficients for each person	4.737
Identify an unknown image of size 180X200 by coefficients for each image	4.766

Figure 7.1: Elapsed time in computation

Complexity of adding an image to Face database:

With respect to adding the image to the database, this should be constant time O(1), since it does not involve any computation.

Complexity of computing the average coefficients:

p = number of pixels
e = number of eigenfaces
n = number of images
m = number of persons

It is difficult to evaluate the complexity of the PCA algorithm since it is computed using JAMA package classes: EigenvalueDecomposition. Since it involves

computation of the Covariance matrix, it can be at least O (p^2). Barring the eigenface computation, the complexity of the following steps is needed.

To get average coefficients for each person, we do:

1. Subtract the mean O(n*p)
2. Compute the coefficients O(e*n*p)
3. Compute the averages O(n*e)

Overall, the complexity for this is O(p^2+p*n*e) that is quite large given that p is the number of pixels.

Complexity of identification of the image:

To get the coefficients for the new image you do

1. Subtraction of the average O(p)
2. Dot product with each eigenface: O(e*p)
3. Compare with the other coefficients:

a) if doing it for persons, we would get O(m)
b) if doing it for each image we would get O(n)

The complexity of
a) Identification by comparing with person average is: O(p)+O(e*p)+O(m) or approximately O((e+1)p+m)

b) Identification by comparing with each image: O(p)+O(e*p)+O(n) or approximately O((e+1)p+n)

In our experiment, p is the large number (number of pixels) since we have only a handful of persons and images. This makes the complexity to be dependent largely on p. If the image number becomes larger and larger, the complexity will be dependent largely on n and m.

CHAPTER 8

CONCLUSIONS/SUMMARY

A real time facial recognition application using Java programming language, JMF (Java Media Framework) library, and JAMA (A Java Matrix Package) was developed and implemented. In this application, Oracle 10g database was used to store the set of training face images as BLOBs (Binary Large Objects). An off-the-shelf webcam was used for face recognition employing an algorithm based on eigenfaces.

Using GUI, face images are captured and stored to the existing training Face database in Oracle. After that PCA operations are performed to compute an average image, eigenfaces, and coefficient vectors for stored training face images. Once a new image has been added to the database, PCA analysis is performed again to compute average image, eigenfaces, and coefficient vectors. To identify an unknown test image taken through a Webcam, its coefficient vector is computed using eigenfaces and average face vectors stored in text files. The spatial distances between test image coefficient vector and the training images coefficient vectors are computed to find out the Euclidean distances of test image with the images stored in Face database. Based on the smallest Euclidean distance of the test image with the images in the database, application identifies (or indicates no match when smallest Euclidean distance is above the threshold value) the test image.

Application showed 85% correct results for twenty (20) test images. Seventeen out of 20 images are correctly identified. Rest of the three incorrectly identified images does not exist in the Oracle database. Their Euclidean distance falls below the empirically determined threshold value 0.3; hence they were

incorrectly matched to the image in the database with the smallest Euclidean distance.

In summary, a real time facial recognition application using Java programming language, JMF (Java Media Framework) library, and JAMA (A Java Matrix Package) was developed, and its real time use was demonstrated.

REFERENCES

[1] K. Arnold, J. Gosling, and D. Holmes, *The Java programming language*, Addison Wesley, 2006

[2] Sun Microsystems, Java Media Framework, available online at http://java.sun.com/javase/technologies/desktop/media/jmf/index.jsp, (accessed 9/7/2008), 2008

[3] Oracle, *Binary Large Objects (BLOB)* class, available online at http://download-west.oracle.com/otn_hosted_doc/jdeveloper/905/jdbc-javadoc/oracle/sql/BLOB.html, (accessed 8/30/2008)

[4] ***, "Handling Large Objects (LOB) - Oracle Database 10g", Online blog entry available online at http://advait.wordpress.com/2007/06/03/handling-large-objects-lob-oracle-database-10g/, (accessed 7/30/2008)

[5] S. Zhang and M. Turk, "Eigenfaces", *Scholarpedia,* 3(9):4244, 2008, available online at http://www.scholarpedia.org/article/Eigenfaces, (accessed 8/10/2008)

[6] ***, *Face Recognition Homepage*, General Info, available online at http://face-rec.org/general-info/ (accessed 9/5/2008)

[7] W. Zhao, R. Chellappa, A. Rosenfeld, P.J. Phillips, "Face Recognition: A Literature Survey", *ACM Computing Surveys*, 2003, pp. 399-458

[8] Ravinder Mann, "CSC 492 Face Recognition Report", available online at http://individual.utoronto.ca/rav/FR/FRindex.htm (accessed 8/30/2008)

[9] R. Gross, J. Shi, J. Cohn, “Quo vadis Face Recognition? - The current state of the art in Face Recognition”, *Technical Report, Robotics Institute*, Carnegie Mellon University, Pittsburgh, PA, USA, 25 pages, available online at http://face-rec.org/interesting-papers/General/gross_ralph_2001_4.pdf (accessed 9/4/2008)

[10] Sun Microsystems, “FaceIdentifier.java”, available online at http://java.sun.com/javase/technologies/desktop/media/jmf/2.1.1/solutions/FaceIdentifier.java (accessed 7/1/2008)

[11] ***, “JWebCam.java”, available online at http://vase.essex.ac.uk/software/ecj-imaging/code/ac/essex/ooechs/imaging/commons/apps/webcam/JWebCam.java.html (accessed 7/1/2008)

[12] M. Turk, A. Pentland, “Eigenfaces for Recognition”, *Journal of Cognitive Neurosicence*, Vol. 3, No. 1, 1991, pp. 71-86

[13] Oracle Corporation, “Oracle Database 10g Release 2 for Microsoft Windows”, available online at http://www.oracle.com/technology/software/products/database/oracle10g/htdocs/10201winsoft.html (accessed 7/1/2008)

[14] J. Hicklin, C. Moler, P. Webb, “JAMA: Java Matrix Package”, The Math Works, available online at http://math.nist.gov/javanumerics/jama/ (accessed 9/1/2008)

[15] K. Kim, “Face Recognition using Principal Component Analysis”, *University of Maryland,* available online at

http://www.umiacs.umd.edu/~knkim/KG_VISA/PCA/FaceRecog_PCA_Kim.pdf (accessed 10/27/2008)

[16] Wikipedia, “Eigenvalue, eigenvector, and eigenspace”, available online at http://en.wikipedia.org/wiki/Eigenvalue (accessed 10/29/2008)

[17] D. Pissarenko, “Eigenface - based facial recognition”, available online at http://openbio.sourceforge.net/resources/eigenfaces/eigenfaces-html/facesOptions.html (accessed 10/30/2008)

[18] S. Ambler, *“Introduction to Data Flow Diagrams (DFDs)”,* The Object Primer 3rd Edition: Agile Model Driven Development with UML 2, http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm (accessed 12/04/2008)

[19] S. Ambler, “*Introduction to UML2 Class Diagrams*”, The Object Primer 3rd Edition: Agile Model Driven Development with UML 2, http://www.agilemodeling.com/artifacts/classDiagram.htm (accessed 12/04/2008)

[20] L. Spacek*, “Collection of Facial Images: Faces94”,* Computer Vision Science Research Projects, http://cswww.essex.ac.uk/mv/allfaces/faces94.html (accessed 12/05/2008)

[21] Oracle Corporation, “Native Datatypes”, Oracle® Database Concepts 10g Release 2, http://download.oracle.com/docs/cd/B19306_01/server.102/b14220/datatype.htm#i3237 (accessed 12/06/2008)

[22] P. Linden, “Java Programmers FAQ”, available online at http://burks.brighton.ac.uk/burks/language/java/jprogfaq/faq_b2.htm#I/O (accessed 11/26/2008)

[23] Java Source and Support, *"Image crop: Image « 2D Graphics GUI « Java",* available online at http://www.java2s.com/Code/Java/2D-Graphics-GUI/Imagecrop.htm (accessed 12/04/2008)

[24] Oracle Corporation, *"Data Types",* Oracle ® Database SQL Reference 10g Release 2 (10.2), available online at http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/sql_elements001.htm#i54330 (accessed 12/07/2008)

APPENDIX A

APPLICATION DEPLOYMENT AND USAGE

A.1 Application Deployment

Before running the application following steps are required to be completed:

1. Install JMF (Java Media Framework) package. Download is available online [2].

2. Install JAMA (Java Matrix) package. Download is available online [14]

3. Connect Logitech QuickCam Pro 3000 to computer

4. Install Oracle software and database. Download is available online [13]

5. Connect to Oracle and run SQL script to create table in database

SQL> CREATE TABLE BLOBS (FILENAME VARCHAR2(30), BINARYDATA BLOB, NAME VARCHAR2(20);

SQL> CREATE TABLE COEFF_VECTOR (FILENAME VARCHAR2(30), NAME VARCHAR2(20), W1 FLOAT, W2 FLOAT, W3 FLOAT, W4 FLOAT);

SQL> CREATE TABLE COEFF_VECTOR (FILENAME VARCHAR2(30), NAME VARCHAR2(20), W1 FLOAT, W2 FLOAT, W3 FLOAT, W4 FLOAT);

6. Copy all the application files in a new folder 'Project' in C: drive. Make sure to change the Oracle userid and password in JDBC connectivity code in SqlInsertfaceBundle.java and SqlGetfaceBundle.java and save and compile them. Compile all the .java files using JAVAC compiler as follows:

C:\Project> javac FaceIdentifier.java (similarly compile all .java files)

7. Set up the classpath for SQL connection as follows:

C:\Project> set
CLASSPATH=.;C:\ORACLE\product\10.2.0\db_1\jdbc\lib\classes12.jar

8. Run the application as follows:

C:\Project> java FaceIdentifier

A.2 Application Usage

1. When the application is started, a window will appear as shown in Figure A.2.1

Figure A.1: Player displaying live video

2. Click 'Capture Snapshot' to take a snapshot

Figure A.2: Snapshot

3. Click ‘Save’ button to save the snapshot in Snapshot window

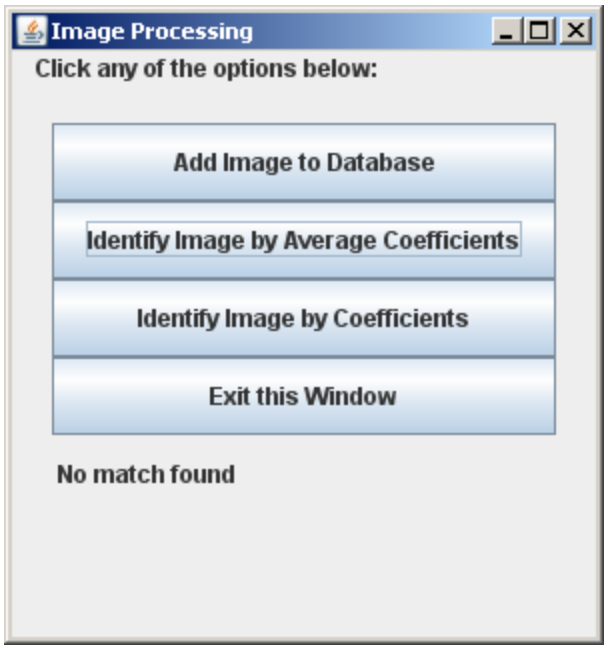

Figure A.3: Image processing window

4. Click ‘Add Image to Database’ shown in Image Processing window shown in Figure A.2.3 to add the image to Face database.

5. Click ‘Identify Inage by Average Coefficients’ Image Processing window shown in Figure A.2.3 to identify the image by average coefficients

6. Click ‘Identify Inage by Average Coefficients’ Image Processing window shown in Figure A.2.3 to identify the image by coefficients

7. When a match is found, the matched image will be displayed in another window, otherwise ‘No match found’ message will be displayed as in Figure A.2.4.

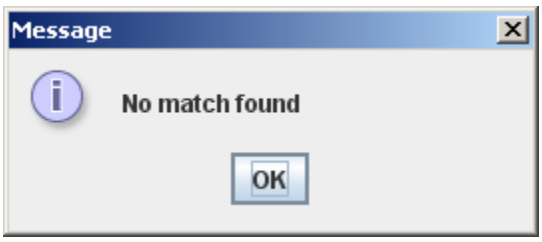

Figure A.4: Message

APPENDIX B

INSTALLING ORACLE SOFTWARE AND DATABASE

Download a free version of Oracle Database 10g Release 2 (10.2.0.1.0) for Microsoft Windows [13]. Sign up to Oracle Technology Network to download the file, if you are not already a member. Accept the license before downloading the zip file "10201_database_win32.zip" (size ~ 655 MB) to local disk [13] as shown in figure A.1. Once the Zip file download is complete, open it and unzip it.

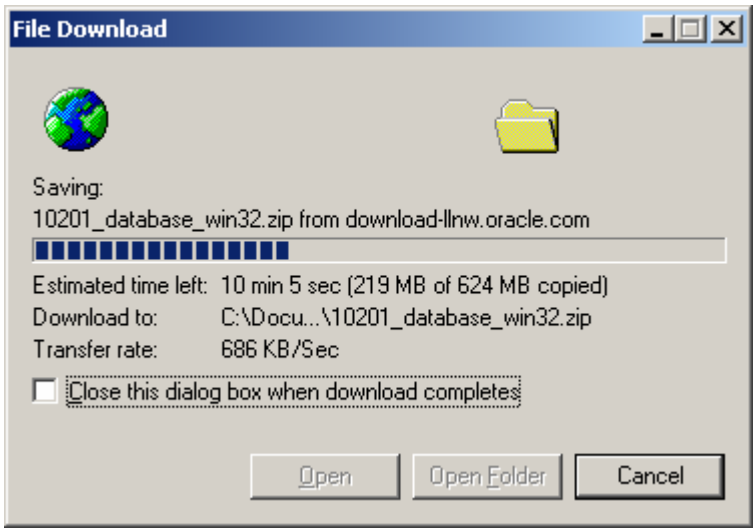

Figure B.1: Downloading file 10201_database_win32.zip (655,025,354 bytes)

Double-click the setup.exe to start the Oracle Universal Installer (OUI) which interactively guide you through the installation process. Figure A.2 shows the Oracle Universal Installer Select Installation method window.

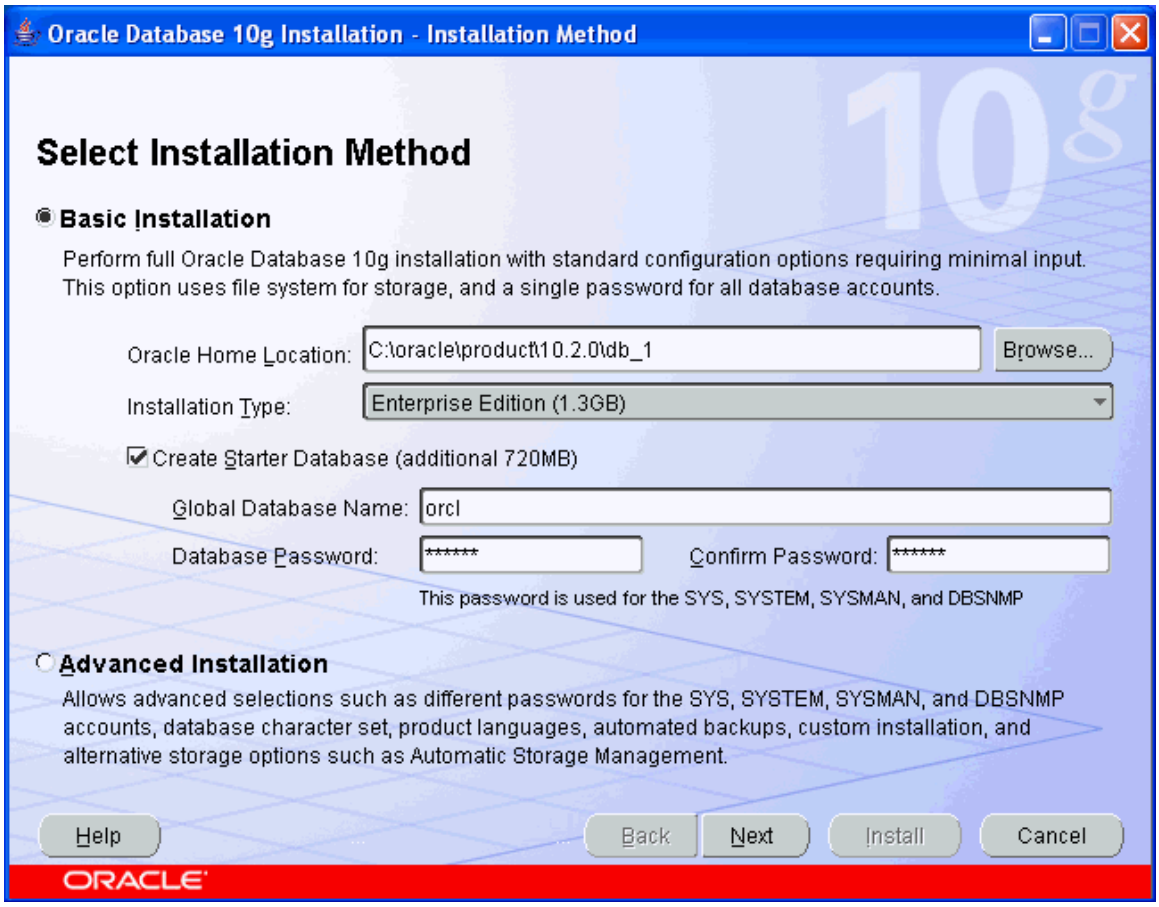

Figure B.2: Oracle Universal Installer Select Installation Method Window

Select the following for basic installation of Standard Edition:

Oracle Home Location: Enter the directory in which to install the Oracle Database 10g software
Installation Type: Select Standard Edition (Windows Only)
Create Starter Database: Check this box to create a database during installation

Click ‘Next’ to continue. Figure B.3 shows the next window as ‘Specify File Location’

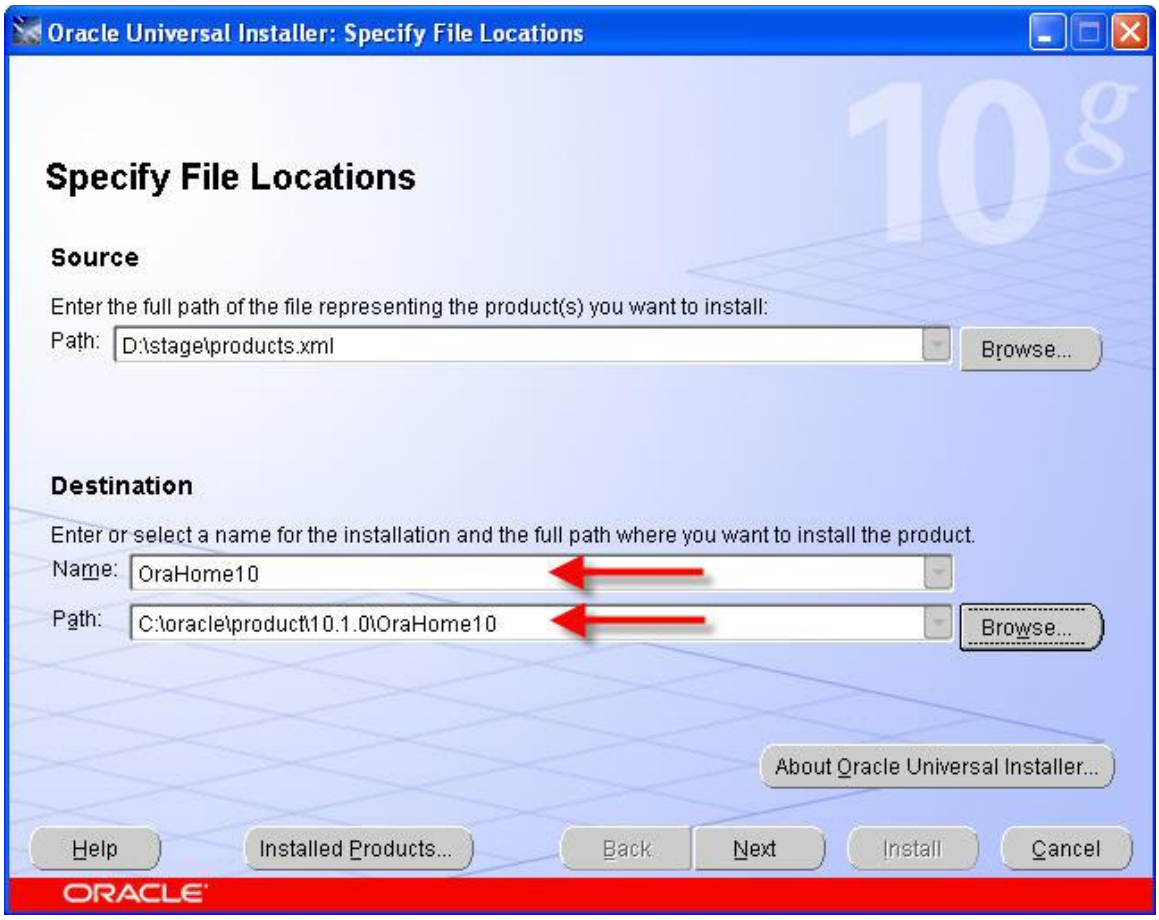

Figure B.3: Oracle Universal Installer Specify File Location

Select the source and destination path for specifying file locations and click next to continue. Figure B.4 shows the next widow to choose installation type. Choose Personal Edition for installation for personal use.

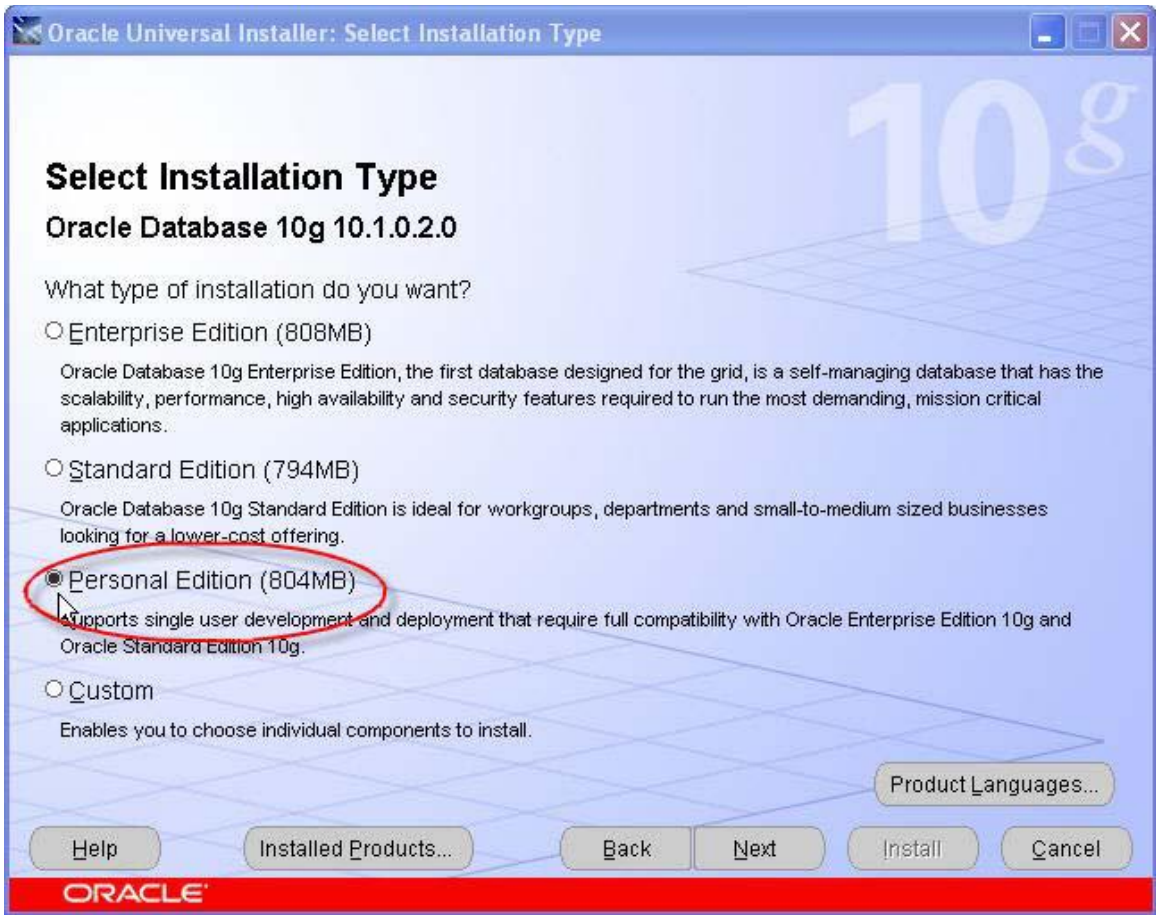

Figure B.4: Oracle Universal Installer: Select Installation Type

Click ‘Next’ to continue. Figure B.5 show the data configuration window.

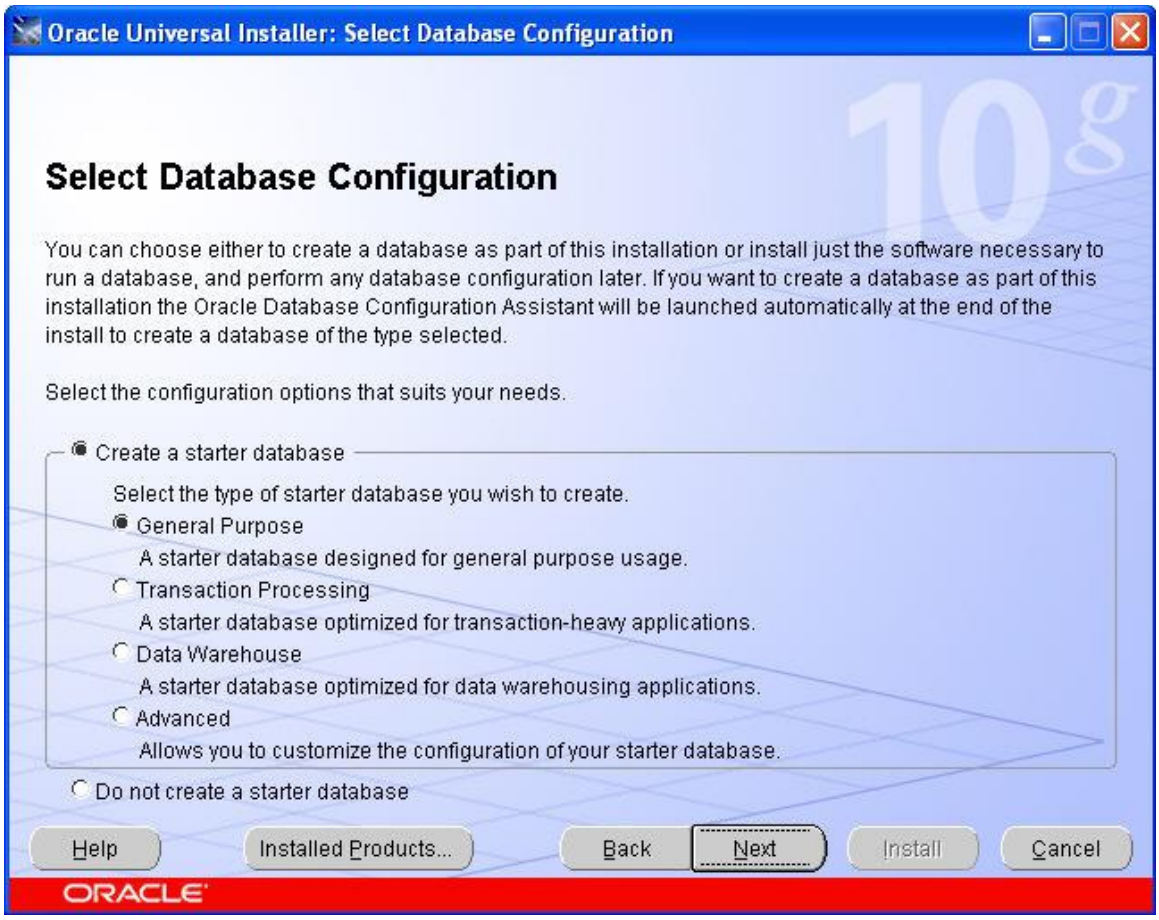

Figure B.5: Oracle Universal Installer: Select Database Configuration

Choose 'Create a starter database' to create the database automatically at the end of installation. Database Configuration Assistant will be automatically launched to create the database automatically. Select 'General Purpose' and click 'Next' to continue by keeping the defaults for database configuration as and complete the installation process. Figure B.6 shows the ongoing installation.

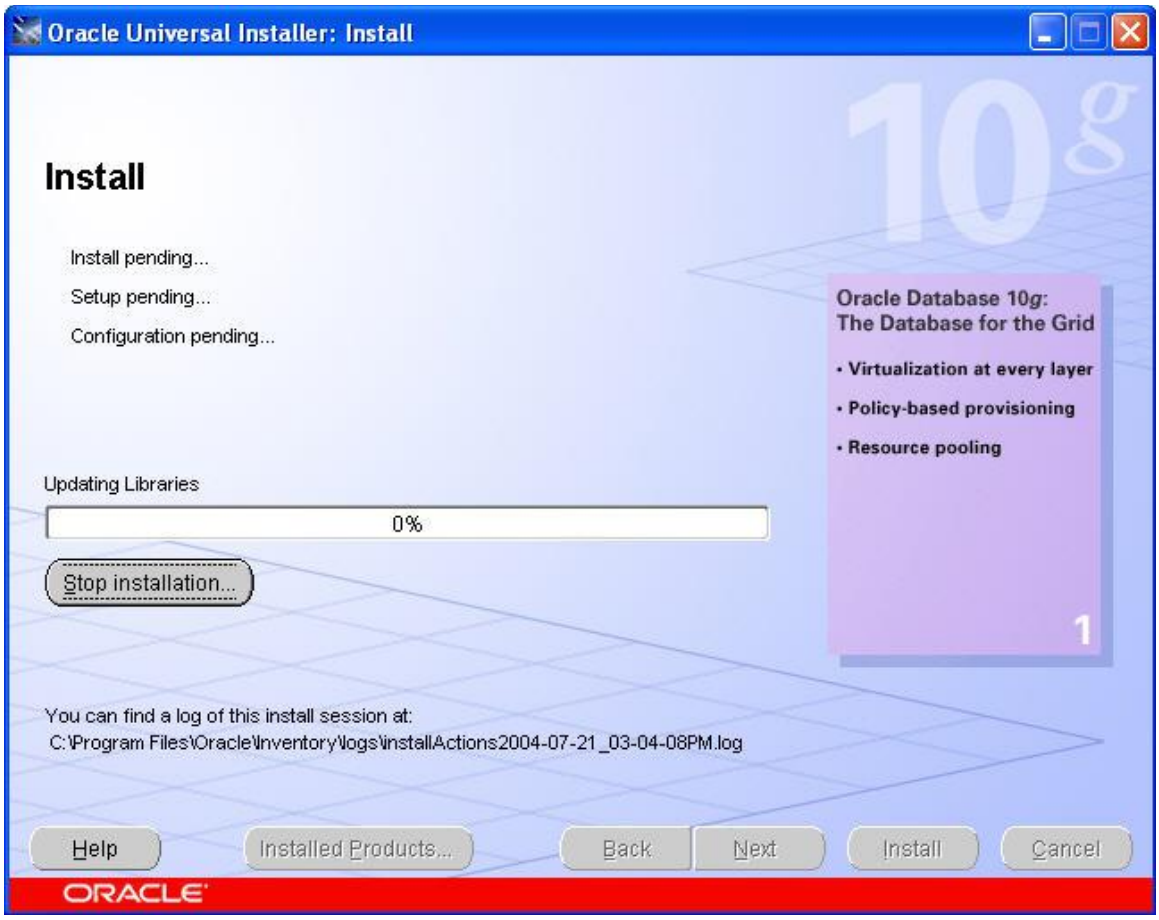

Figure B.6: Oracle Universal Installer: Install

Database Configuration Assistant window will be launched to create the database automatically on completion of installation. No action required. Note down the Global database name to remember as shown in Figure B.7.

Figure B.7: Database Configuration Assistant

Figure B.8 shown the end of installation window. Click ‘Exit’ when End of installation window appears to exit the installation upon its completion.

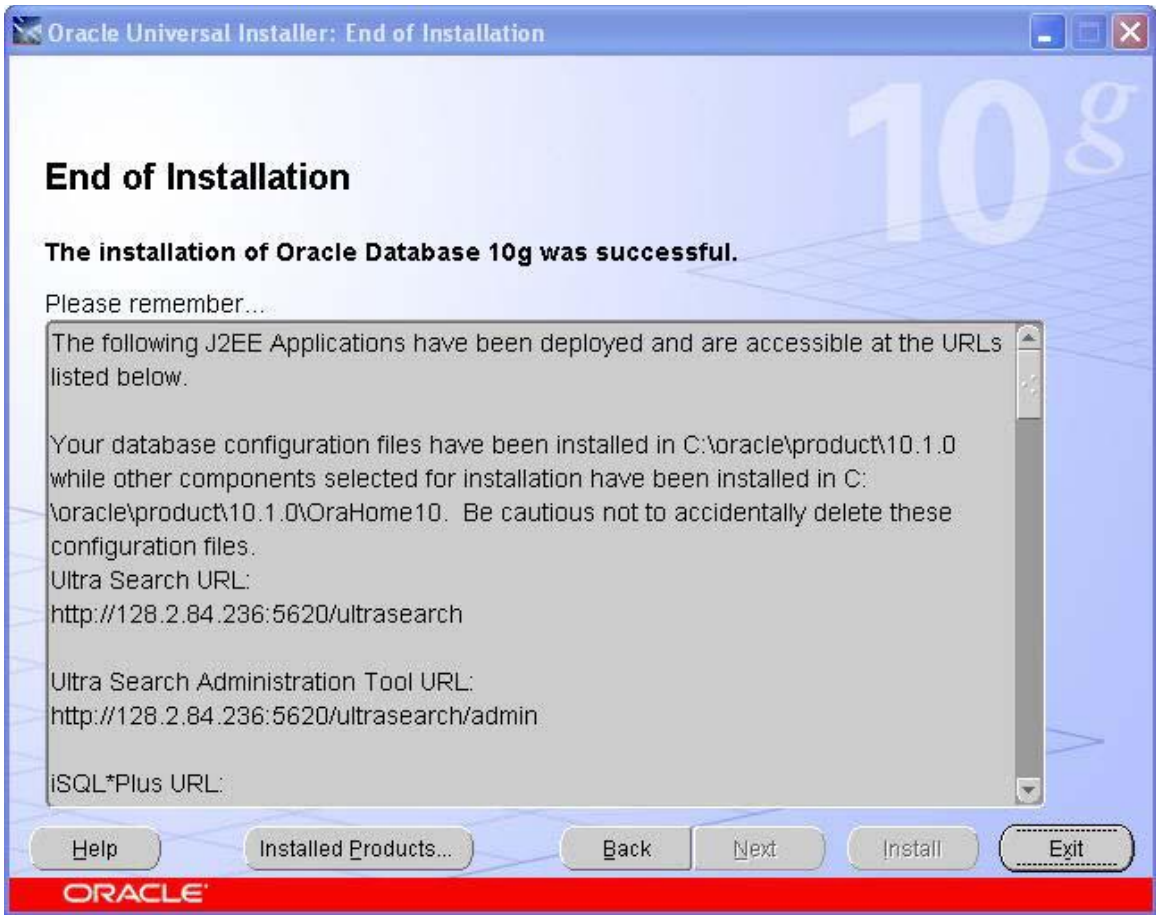

Figure B.8: Oracle Universal Installer: End of installation

APPENDIX C

SOURCE CODE LISTING

C.1 Start the media player to display the live video

```
MediaLocator ml;
String str2 = "vfw:Microsoft WDM Image Capture (Win32):0";
CaptureDeviceInfo di = CaptureDeviceManager.getDevice(str2);
ml = di.getLocator();

startPlayer(MediaLocator ml)
 {
       p = Manager.createProcessor(ml);

       p.addControllerListener(this);

       // Put the Processor into configured state.
       p.configure();

       // So I can use it as a player.
       p.setContentDescriptor(null);

       // Obtain the track controls.
       TrackControl tc[] = p.getTrackControls();

       // Search for the track control for the video track.
       TrackControl videoTrack = null;
       for (int i = 0; i < tc.length; i++) {
          if (tc[i].getFormat() instanceof VideoFormat) {
             videoTrack = tc[i];
             break;
          }
       }

       // Realize the processor.
       p.prefetch();
       if (!waitForState(p.Prefetched)) {
          System.err.println("Failed to realize the processor.");
          return false;
       }

         CanvasPanel = new Panel();
```

```
        add(CanvasPanel);

      // Display the visual component

      if ((vc = p.getVisualComponent()) != null) {
            CanvasPanel.add(vc);
      }

      p.start();
      setVisible(true);

   } // End of startPlayer()
```

C.2 Capture a snapshot from the live video

```
Image snap = grabFrameImage (p);

Snapshot snapshot = null;
snapshot = new Snapshot ( snap, new Dimension ( imageSize ) );

public Image grabFrameImage ( Processor p1)
{
        Processor p2;
        p2 = p1;
        Buffer buffer = new Buffer();
        buffer = grabFrameBuffer(p2);

      if ( buffer != null )
      {
         // Convert it to an image
         BufferToImage btoi = new BufferToImage (
(VideoFormat)buffer.getFormat() );
          if ( btoi != null ) {
             Image image = btoi.createImage ( buffer );
             if ( image != null ) {
                return ( image );
             }
          }
      }
 }
```

C.3 Display snapshot

```
public Snapshot ( Image grabbedFrame, Dimension imageSize )
{
            photo = grabbedFrame;

             photo = createImage(new FilteredImageSource(photo.getSource(),new
CropImageFilter(73, 63, 180, 200)));

             setVisible ( true );
}
```

C.4 Compute the PCA statistics

```
EigenFaceComputation .submitFaceBundle(double[][] face_v, int width, int
height, String[] id, String[] name, int MAGIC_NR)
```

C.4.1 Compute average face of all of the faces

```
        for ( pix = 0; pix < length; pix++)
        {
                temp = 0;
                for ( image = 0; image < nrfaces; image++) {
                temp +=  face_v[image][pix];
                }

                avgF[pix] = temp / nrfaces;
        }
```

C.4.2 Compute difference

```
   for ( image = 0; image < nrfaces; image++)
   {
        for ( pix = 0; pix < length; pix++)
                face_v[image][pix] = face_v[image][pix] - avgF[pix];
   }
```

C.4.3 Build Covariance matrix

```
   Matrix faceM = new Matrix(face_v, nrfaces,length);
   Matrix faceM_transpose = faceM.transpose();
```

```
/* Covariance matrix - its MxM (nrfaces x nrfaces) */

Matrix covarM = faceM.times(faceM_transpose);

double[][] z = covarM.getArray();

/* Compute eigenvalues and eigenvector. Both are MxM */

EigenvalueDecomposition E = covarM.eig();

double[] eigValue = diag(E.getD().getArray());
double[][] eigVector = E.getV().getArray();

 /*  Sort eigValue to get largest associated values of the eigenvalues */

int[] index = new int[nrfaces];
double[][] tempVector = new double[nrfaces][nrfaces];

for ( i = 0; i <nrfaces; i++)
      index[i] = i;

doubleQuickSort(eigValue, index,0,nrfaces-1);

/* Put the index in inverse */

int[] tempV = new int[nrfaces];
for ( j = 0; j < nrfaces; j++) {
      tempV[nrfaces-1-j] = index[j];
}
index = tempV;

 /* Put the sorted eigenvalues in the appropiate columns. */

for ( col = nrfaces-1; col >= 0; col --) {
      for ( rows = 0; rows < nrfaces; rows++ ){
      tempVector[rows][col] = eigVector[rows][index[col]];
      }
}

eigVector = tempVector;
tempVector = null;
eigValue = null;
```

C.4.4 Compute eigenfaces (eigVector)

```
Matrix eigVectorM = new Matrix(eigVector, nrfaces,nrfaces);
Matrix eigFaceM = new Matrix(MAGIC_NR, nrfaces);
eigFaceM = eigVectorM.getMatrix(0, MAGIC_NR-1, 0, nrfaces-1);
eigVector = new double[MAGIC_NR][nrfaces];
eigVector = eigFaceM.times(faceM).getArray();
```

C.4.5 Compute coefficient vectors (wk)

```
   double[][] wk = new double[nrfaces][MAGIC_NR]; // 5 rows, 4 columns

   for (image = 0; image < nrfaces; image++)
   {
        for (j  = 0; j <  MAGIC_NR; j++)
        {
        temp = 0.0;
        for ( pix=0; pix< length; pix++) {
                temp += eigVector[j][pix] * faces[image][pix];
                }
                wk[image][j] = Math.abs( temp );
        }
   }
```

C.4.6 Insert coefficient vectors into BLOBS table in Face database

```
        query=new StringBuffer("insert into coeff_vector(filename, name, w1, w2,
w3, w4) values (?,?,?,?,?,?)");

        Class.forName("oracle.jdbc.driver.OracleDriver");
        conn =
DriverManager.getConnection("jdbc:oracle:thin:@trishul:1521:orcl","palk","palk");

        pstmt=conn.prepareStatement(query.toString());

        pstmt.setString(1, filename);
        pstmt.setString(2, name);
        pstmt.setDouble(3, w1);
        pstmt.setDouble(4, w2);
        pstmt.setDouble(5, w3);
        pstmt.setDouble(6, w4);

        pstmt.executeUpdate();
```

C.4.7 Compute average coefficient vectors

```
   for(i=0; i<n; i++)
        {
                query = "select * from coeff_vector where name='" + pname[i] +"'";
                System.out.println(query);
                ResultSet res1 = stmt.executeQuery(query);

                w = 3;
                avg_w = new double[MAGIC_NR];

                for( j=0; j<MAGIC_NR; j++)
                {
                        temp = 0.0;
                        nrfaces = 0;
                        res1 = stmt.executeQuery(query);

                        while(res1.next())
                        {
                                temp += res1.getDouble(w);
                                nrfaces++;
                        }
                        w++;
                        avg_w[j] = Math.abs( temp / nrfaces);

                }

                insertAvgCoeff(fname[i], pname[i], avg_w[0], avg_w[1], avg_w[2],
avg_w[3]);
        }
```

C.4.8 Insert average coefficient vectors in Face database

```
        query=new StringBuffer("insert into
        avg_coeff_vector(filename,name,avg_w1,avg_w2,avg_w3,avg_w4) values
        (?,?,?,?,?,?)");

        Class.forName("oracle.jdbc.driver.OracleDriver");
        conn =
DriverManager.getConnection("jdbc:oracle:thin:@trishul:1521:orcl","palk","palk");

        pstmt=conn.prepareStatement(query.toString());
```

```
        pstmt.setString(1, filename);
        pstmt.setString(2, name);
        pstmt.setDouble(3, avg_w1);
        pstmt.setDouble(4, avg_w2);
        pstmt.setDouble(5, avg_w3);
        pstmt.setDouble(6, avg_w4);

        pstmt.executeUpdate();
```

C.5 Add a JPG image to Face database in Oracle

```
 public void insertImageToBLOBS(String img_name, String face_name)
 {

        String filename =img_name;
        String name =face_name;

        query=new StringBuffer("insert into blobs(filename,binarydata,name)
values (?,?,?)");
        File file= new File(filename);

        Class.forName("oracle.jdbc.driver.OracleDriver");
        conn =
DriverManager.getConnection("jdbc:oracle:thin:@trishul:1521:orcl","palk","palk");

        pstmt=conn.prepareStatement(query.toString());
        pstmt.setString(1, filename);
        pstmt.setBinaryStream(2, new FileInputStream(filename),(int)file.length());
        pstmt.setString(3,name);
        pstmt.executeUpdate();

}
```

C.6 Get the average face, eigenfaces, and coefficient vectors from Face database

```
FaceBundle b = sql_get_fb.getFaceBundle(len);

wk = new double[nrfaces][MAGIC_NR];
avg_wk = new double[nrpersons][MAGIC_NR];
avgF = new double[length];
eigVector = new double[MAGIC_NR][length];

 FaceBundle b = new FaceBundle(avgF, avg_wk, wk, eigVector , file_names,
person_names, img_names, face_names );
```

C.6.1 Get coefficient vector from BLOBS table

```
res = stmt.executeQuery("select * from coeff_vector");
img_names = new String[nrfaces];
face_names = new String[nrfaces];
i=0;
while(res.next() )
{
        img_names[i] = res.getString(1);
        face_names[i] = res.getString(2);
        wk[i][0] = res.getDouble(3);
        wk[i][1] = res.getDouble(4);
        wk[i][2] = res.getDouble(5);
        wk[i][3] = res.getDouble(6);
        i++;
}
```

C.6.2 Get average coefficient vector from BLOBS table

```
res = stmt.executeQuery("select * from avg_coeff_vector");
file_names = new String[nrpersons];
person_names = new String[nrpersons];
i = 0;
while(res.next() )
{
        file_names[i] = res.getString(1);
        person_names[i] = res.getString(2);
        avg_wk[i][0] = res.getDouble(3);
        avg_wk[i][1] = res.getDouble(4);
        avg_wk[i][2] = res.getDouble(5);
        avg_wk[i][3] = res.getDouble(6);
```

```
            i++;
        }
```

C.6.3 Get avgFace vector from text files

```
        in1 = new FileInputStream("avgFace.txt");
        stok = new StreamTokenizer(in1);
        stok.parseNumbers();
        stok.eolIsSignificant(true);
        stok.nextToken();

       pix=0;
        while (stok.ttype != StreamTokenizer.TT_EOF)
        {
          while (stok.ttype != StreamTokenizer.TT_EOL)
          {
                avgF[pix] = stok.nval;

                stok.nextToken();
                pix++;
       }
          stok.nextToken();
        }
        in1.close();
```

C.6.4 Get eigenface Vectors from text files

```
        in2 = new FileInputStream("eigVector.txt");
        st = new StreamTokenizer(in2);
        st.parseNumbers();
        st.eolIsSignificant(true);
        st.nextToken();

        image = 0;
        while (st.ttype != StreamTokenizer.TT_EOF)
        {
          pix = 0;
          while (st.ttype != StreamTokenizer.TT_EOL)
          {
                switch (st.ttype)
                {
               case StreamTokenizer.TT_NUMBER:
                     double num = st.nval;
                     int exp = 0;
                     st.ordinaryChars('\0', ' ');
                     st.nextToken();
```

```
                    st.whitespaceChars('\0', ' ');
                    if (st.ttype == StreamTokenizer.TT_WORD &&
Character.toUpperCase(st.sval.charAt(0)) == 'E') {
                        try {
                            exp = Integer.parseInt(st.sval.substring(1));
                        } catch (NumberFormatException e) {
                            st.pushBack();
                        }
                    } else if (st.ttype < 0 || st.ttype > ' ')
                        st.pushBack();

                            eigVector[image][pix] = num * Math.pow(10, exp);
                    break;
                case StreamTokenizer.TT_WORD:
                        break;
               default:
                        break;
                } // end switch
                pix++;
        st.nextToken();
       }
         image++;
         st.nextToken();

       }

       in2.close();
```

C.7 Submit an unknown face to FaceBundle representing face-space

```
id = b.submitFace(img);
```

C.7.1 Subtract the image from the average image

```
       for ( pix = 0; pix < inputFace.length; pix++) {
       inputFace[pix] = inputFace[pix] - avgFace[pix];
       }

       double[] input_wk = new double[MAGIC_NR];
       double temp = 0;

       for (j = 0; j < MAGIC_NR; j++) {
               temp = 0.0;
```

```
            for ( pix=0; pix <inputFace.length; pix++) {
                temp += eigVector[j][pix] * inputFace[pix];
                }
                input_wk[j] = Math.abs( temp );
        }
```

C.7.2 Finds the minimun Euclidean distance of the new test image coefficient (input_wk) with the average coefficients (avg_wk)

```
        double[] Edistance = new double[avg_wk.length];
        double minEdistance = 0.0;

        for (image = 0; image < avg_wk.length ; image++)
        {
        temp = 0.0;
                for (j = 0; j < MAGIC_NR; j++) {
                temp = temp + (Math.abs((input_wk[j] - avg_wk[image][j]) *
(input_wk[j] - avg_wk[image][j])));
                }

                Edistance[image] = Math.sqrt(temp);

                if(image==0) {
                        minEdistance = Edistance[image];
                        id[0] = file_ids[image];
                        id[1] = person_names[image];
                }

                if(Edistance[image] == 0.0) {
                        id[0] = file_ids[image];
                        id[1] = person_names[image];
                        minEdistance = Edistance[image];
                        break;
                }

                if (minEdistance > Edistance[image]) {

                        id[0] = file_ids[image];
                        id[1] = person_names[image];
                        minEdistance = Edistance[image];
                }
        }

        minED = minEdistance;
```