

**SP06077**



# Face Detection & Recognition System

Submitted By

**BHAVNISH GUPTA (061228)**

**GAURAV JAIN (061240)**

**RAJAT GUPTA (061441)**



**JULY-MAY 2009-10**

**Submitted in partial fulfillment of the Degree of Bachelor of  
Technology**

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING  
& INFORMATION TECHNOLOGY**

**JAYPEE UNIVERSITY OF INFORMATION  
TECHNOLOGY-WAKNAGHAT**

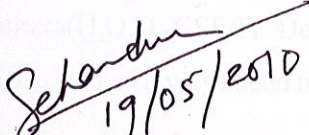
**Dist.- Solan (H.P.)**

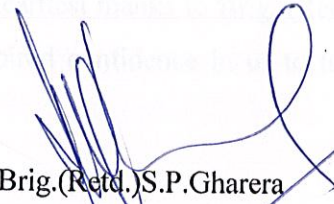
**Pin-173215**



## CERTIFICATE

This is to certify that the work entitled, "Face Detection & Recognition System" submitted by Gaurav Jain (061240), Bhavnish Gupta (061228), Rajat Gupta (061441) in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

  
19/05/2010  
Satish Chandra  
(Senior Lecturer)

  
Brig. (Retd.) S.P. Gharera  
(HOD-CSE/IT)

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING  
& INFORMATION TECHNOLOGY

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY  
WAKNAGHAT Dist. – Solan (H.P.)  
Pin-173215



## ACKNOWLEDGMENT

Apart from the efforts by us, the success of this project depends largely on encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We would like to show our greatest appreciation to our supervisor Sr. Lec. Mr. Satish Chandra. We feel motivated and encouraged every time we get his encouragement. For his coherent guidance throughout the tenure of the project, we feel fortunate to be taught by him, who gave us his unwavering support.

We feel grateful working together as a group. We owe our heartiest thanks to Brig. (Retd.) S.P. Ghrera (H.O.D.-CSE/IT Department) who've always inspired confidence in us to take initiative. He has always been motivating and encouraging.

Finally, thanks to all our family members who supported us in our every grim phase.

Project Group No.-6

*Bhavnish Gupta*

Bhavnish Gupta (061228)

*Gaurav Jain*

Gaurav Jain (061240)

*Rajat Gupta*

Rajat Gupta (061441)

B.TECH (CSE)



## **TABLE OF CONTENTS**

<b>TOPIC</b>	<b>PAGE NO.</b>
<b>I) CERTIFICATE</b>	<b>I</b>
<b>II) ACKNOWLEDGMENT</b>	<b>II</b>
<b>III) LIST OF FIGURES</b>	<b>VII</b>
<b>IV) ABBREVIATIONS</b>	<b>IX</b>
<b>V) ABSTRACT</b>	<b>X</b>
<b>CHAPTER I: INTRODUCTION TO FACE DETECTION AND RECOGNITION SYSTEM</b>	<b>1-10</b>
1.1 Problem Statement	1
1.2 Introduction to FDRS	3
1.3 Objective	5
1.4 Process Description	6
1.4.1 Level 0 DFD	6
1.4.2 Level 1 DFD	6
1.4.3 Level 2 DFD	7
1.5 Timeline of Project	8
1.6 Gantt Chart of Project	9
1.7 Tasks of Project	9
<b>CHAPTER II: METHODOLOGY</b>	<b>11-15</b>
2.1 Steps in the System	11
2.1.1 Face Detection	11
2.1.2 Extraction ROI	12
2.1.3 Query Image	12
2.1.4 Face Recognition	13
2.2 Flow of the System	14



<b>CHAPTER III: FACE DETECTION</b>	<b>16-26</b>
3.1 Proposed Method for Face Detection	19
3.2 Face Detection Algorithm	20
3.2.1 Color space transformation and lighting compensation	20
3.2.2 High frequency noisy removing	22
3.2.3 Skin-Color Blocks	22
3.2.4 Height to width ratio detection	23
3.2.5 Mouth detection	23
3.2.6 Eyes detection	24
<b>CHAPTER IV: EXTRACTING REGION OF INTEREST</b>	<b>27-29</b>
4.1 Idea behind Region of Interest analysis	27
4.2 ROI Selection	28
4.3 How to select a region?	28
<b>CHAPTER V: FACE RECOGNITION</b>	<b>30-45</b>
5.1 Problems and Considerations	35
5.2 Eigenfaces	37
5.3 Eigenface Generation	37
5.4 Use in Facial Recognition	38
5.5 Principal Component Analysis (PCA)	40
5.5.1 Flowchart	40
5.5.2 Algorithm	41
5.5.3 Face Recognition Using Eigenfaces	43
5.6 Advantages of PCA	44
5.7 Disadvantages of PCA	44
<b>CHAPTER VI: PARTICLE SWARM OPTIMIZATION (PSO)</b>	<b>46-48</b>
6.1 Algorithm PSO	47
6.2 Feature Optimization Using PSO	48
<b>CHAPTER VII: NODAL POINTS</b>	<b>49-50</b>



7.1 Capture	49
7.2 Extraction	49
7.3 Comparison	49
7.4 Matching	50
7.5 Face Matching Using the Nodal Points	50
<b>CHAPTER VIII: FEATURES OF FACE</b>	<b>51</b>
8.1 Color	51
8.2 Edge	51
8.3 Texture	51
<b>CHAPTER IX: SNAPSHOTS OF IMPLEMENTATION</b>	<b>52-58</b>
9.1 Snapshots of FDRS	52
9.1.1 Interface of FDRS	52
9.1.2 Loading of Input Image	53
9.1.3 Fetching Query Image	54
9.1.4 Observed Outputs	55
9.2 Snapshots of Face Recognition Implementation	59
9.2.1 Interface	59
9.2.2 Input Image	59
9.2.3 Testing Images	60
9.2.4 Matched O/P	60
<b>CHAPTER X: TECHNOLOGY &amp; APPLICATIONS</b>	<b>61-62</b>
10.1 Implementation Details	61
10.2 Applications	61
10.2.1 Image Search Engine	61
10.2.2 Face Detection & Recognition on Facebook/Orkut	61
10.2.3 Security Application	61
10.2.4 Person Recognition	61
10.3 How to make it Better?	62
10.4 Alternate Technology	62



<b>CHAPTER XI: FUTURE ASPECTS</b>	<b>63-69</b>
11.1 Future of Face Detection	63
11.1.1 Explanation	63
11.1.2 Reverse-engineering artist busts face detection tech	64
11.1.3 Nokia N86 to have face detection in future firmware update	66
11.1.4 Secure USB Drive relies on recognizing faces	67
11.2 Future of Face Recognition	68
11.2.1 Mobile authentication	69
11.2.2 IR-based technology	69
11.2.3 3D Face Recognition	69
 <b>CHAPTER XII: CONCLUSION &amp; RESULTS</b>	 <b>70</b>
 <b>CHAPTER XIII: PROJECT CODING</b>	 <b>71-94</b>
13.1 GUI Code	71
13.2 Matching Function Code	73
13.3 Function Regioncrop Code	75
 <b>BIBLIOGRAPHY</b>	 <b>95-96</b>



## LIST OF FIGURES

<b>FIGURES</b>	<b>PAGE NO.</b>
Figure 1.1. Basic outline of the system -----	2
Figure 1.2. DFD Level 0-----	6
Figure 1.3. DFD Level 1-----	6
Figure 1.4. DFD Level 2-----	7
Figure 1.5. TIME LINE OF PROJECT-----	8
Figure 1.6. GANTT CHART OF PROJECT-----	9
Figure 2.1. Input Image -----	11
Figure 2.2. ROI Images -----	12
Figure 2.3. Query Image -----	12
Figure 2.4. Face Recognized Image -----	13
Figure 2.5. Flow of the system -----	14
Figure 2.6. Block Diagram of the Proposed Model -----	15
Figure 3.1. The flow of proposed method for face detection. -----	19
Figure 3.2. An example of bright image compensation-----	21
Figure 3.3. An example of dark image compensation-----	22
Figure 3.4. Noise removal by 5x5 low pass filter -----	22
Figure 3.5. Example of mouth detection -----	25
Figure 3.6. Example of eyes detection-----	25
Figure 3.7. Face detection results on cartoon & real human-----	26
Figure 3.8. Face detection results on dark and bright light vision-----	26
Figure 4.1. Extracted faces from an input image-----	29
Figure 5.1. Original Faces-----	39
Figure 5.2. Eigen Faces-----	39



Figure 5.3.Flow chart-----	40
Figure 5.4.Algorithm:NxN image-----	41
Figure 5.5.Eigenfaces obtained from the faces detected-----	45
Figure 6.1.Optimized faces after using PSO-----	48
Figure 7.1.Representation of nodal Points on Human Face-----	49
Figure 7.2.Final matched image after Nodal point analysis-----	50
Figure 9.1.Interface of FDRS-----	52
Figure 9.2.Loading of input image-----	53
Figure 9.3.Fetching of query image-----	54
Figure 9.4.Observed output1-----	55
Figure 9.5.Observed output2-----	55
Figure 9.6.Observed output3 (false negative) -----	56
Figure 9.7.Observed output4 (true negative) -----	57
Figure 9.8.Observed Output5 (false positive) -----	58
Figure 9.9.Face Recognition Interface-----	59
Figure 9.10.Face Recognition: input image-----	59
Figure 9.11.Testing images-----	60
Figure 9.12.Matched o/p image-----	60
Figure 11.1.Image Geometric Features-----	63
Figure 11.2.Window Search -----	63
Figure 11.3.Viola Jones Detection -----	65
Figure 11.4.N-86 Face Detection GUI-----	67
Figure 11.5.Lock Face Opener -----	68
Figure 12.1.Results of 10 runs on 4 image sets-----	70



## ABBREVIATIONS

1.	<b>FDRS</b>	Face Detection & Recognition System
2.	<b>FD</b>	Face Detection
3.	<b>FR</b>	Face Recognition
4.	<b>RTS</b>	Real Time System
5.	<b>PSO</b>	Particle Swarm Optimization
6.	<b>QI</b>	Query Image
7.	<b>PCA</b>	Principal Component Analysis
8.	<b>LDA</b>	Linear Discriminant Analysis
9.	<b>ICA</b>	Independent Component Analysis
10.	<b>SVM</b>	Support Vector Machine
11.	<b>ROI</b>	Region of Interest
12.	<b>DFD</b>	Data Flow Diagrams
13.	<b>EEG</b>	Electroencephalogram
14.	<b>FMRI</b>	Functional Magnetic Resource Imaging
15.	<b>ML</b>	Maximum Likelihood
16.	<b>AAM</b>	Active Appearance Model
17.	<b>GA</b>	Genetic Algorithms
18.	<b>DIP</b>	Digital Image Processing
19.	<b>IPT</b>	Image Processing Toolbox
20.	<b>Open CV</b>	Open Computer Vision
21.	<b>USB</b>	Universal Serial Bus



## ABSTRACT

Human face detection and recognition is essential parameter in video surveillance and image database management. The emergence of high resolution digital cameras for recording of still images and video streams has had a significant impact on how communication and entertainment have developed during the recent years. This project provides an improved version of face recognition and detection which can be used to extract a particular face from an image consisting of group of faces. The model proposed in this project gives good performance with complex background, filtering out irrelevant parts such as cartoons, other persons' faces etc.

Whether or not the purpose is entertainment or dead serious surveillance, tasks like detection and recognition of faces are solved using the same methods. Due to the varying and generally adverse conditions under which images are recorded there is a call for algorithms capable of working in an unconstrained environment. The model exhibits good performance under the different constraints generously with high speed and high detection ratio. The model is effective on facial variations such as dark or bright vision, close eyes, open mouth, a half-profile face and pseudo faces.

Automated feature selection for the face detection (FD) has been an interesting topic in face recognition (FR) systems. It becomes more complex whenever one has to differentiate between large groups of images spatially for resembling images. The main goal of any face detection technique is to detect human faces from the captured scene in Real time system (RTS). In the past several years many methods have been developed with different goals achieving high performance rates with face localization and the recognition.

The project proposes a new model, Face Detection and Recognition System (FDRS) for the purpose of face recognition. In this model an input image undergoes face detection process, followed by face recognition which is done on the basis of provided query image. Fig. 1 shows the basic outline of the system, where an input image is taken as the query image which undergoes face detection to find out the faces in the image.



# INTRODUCTION TO FACE DETECTION & RECOGNITION SYSTEM (FDRS)

### 1.1 Problem Statement

There was need for automated surveillance system for security, monitoring the people entering the premises. Surveillance is very useful to governments and law enforcement to maintain social control, recognize and monitor threats, and prevent/investigate criminal activity. Image search technology can be used to implement such intelligent surveillance systems.

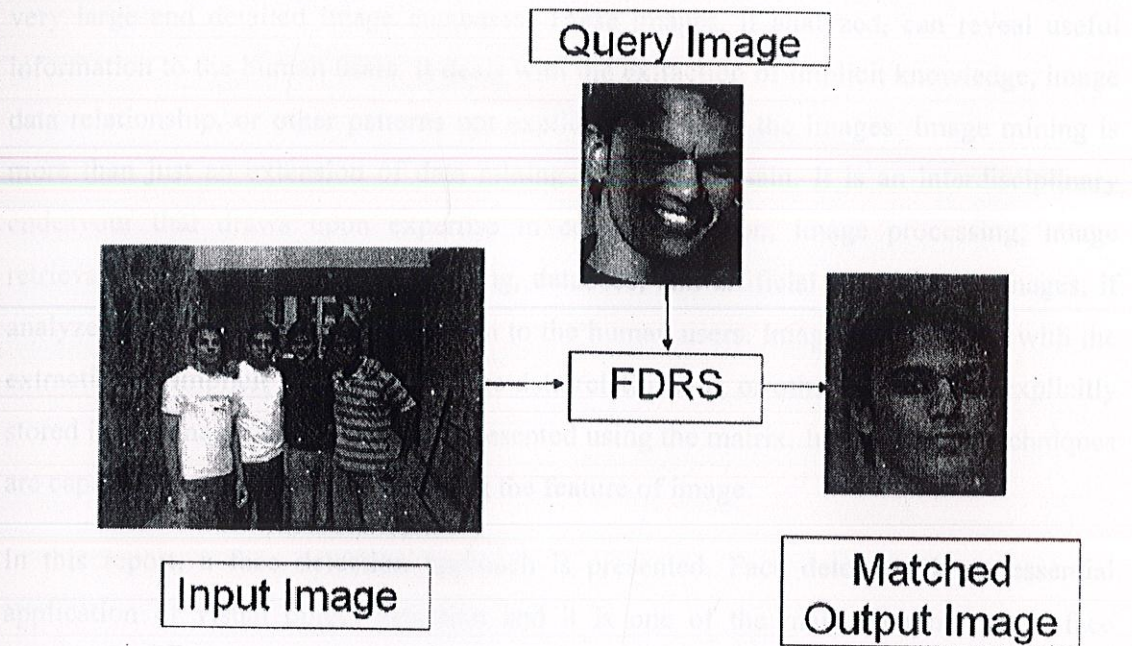
Human face detection is of paramount importance in video surveillance and image database management. This report provides an improved version of face recognition and detection which can be used to extract a particular face from an image consisting of group of faces. The model proposed in this project gives good performance with complex background, filtering out irrelevant parts such as cartoons, other persons' faces etc. It gives good performance under the different lighting conditions with high speed and high detection ratio. The project is effective on facial variations such as dark or bright vision, close eyes, open mouth, a half-profile face and pseudo faces.

Automated feature selection for the face detection (FD) has been an interesting issue to face recognition (FR) systems. It becomes more complex whenever one has to differentiate between large groups of images specially, for resembling images. The main goal of any face detection technique is to detect human faces from the captured ones in Real time system (RTS). In the past several years many methods have been developed with different goals achieving high performance rates with face localization and face recognition.

The project proposes a new model - Face Detection and Recognition System (FDRS) for the purpose of face recognition. In this model an input image undergoes face detection process, followed by face recognition which is to be done on the basis of provided query image. Fig. 1 shows the basic outline of the system where an input image is taken as the query image which undergoes face detection techniques resulting in chopping out the 4



faces, now these faces get matched with the query image and the result gets displayed as the matched output image.



**Figure 1.1.** Basic outline of the system



## 1.2 Introduction to FDRS

Advances in image acquisition and storage technology have led to tremendous growth in very large and detailed image databases. These images, if analyzed, can reveal useful information to the human users. It deals with the extraction of implicit knowledge, image data relationship, or other patterns not explicitly stored in the images. Image mining is more than just an extension of data mining to image domain. It is an interdisciplinary endeavour that draws upon expertise in computer vision, image processing, image retrieval, data mining, machine learning, database, and artificial intelligence. Images, if analyzed, can reveal useful information to the human users. Image mining deals with the extraction of implicit knowledge, image data relationship, or other patterns not explicitly stored in the images. Image can be represented using the matrix. Image mining techniques are capable of detecting and classifying the feature of image.

In this report, a face detection approach is presented. Face detection is an essential application of visual object detection and it is one of the main components of face analysis and understanding with face localization and face recognition. It becomes a more and more complete domain used in a large number of applications, among which we find security, new communication interfaces, biometrics and many others. The goal of face detection is to detect human faces in still images or videos, in different situations. In the past several years, lots of methods have been developed with different goals and for different contexts. We will make a global overview of the main of them and then focus on a detector which processes images very quickly while achieving high detection rates. This detection is based on a boosting algorithm called AdaBoost and the response of simple Haar-based features used by Viola and Jones. The motivation for using Viola's face detection framework is to gain experience with boosting and to explore issues and obstacles concerning the application of machine learning to object detection.

In our project we have primarily focused on the hybridization of face detection and face recognition techniques. Face detection (FD) is done using the eye detection, mouth detection, skin block detection which leads to the extraction of the most informative part in the form of rectangular grids which are further used to enhance the efficacy of the detection. Face recognition (FR) is done by obtaining the eigenfaces and on the basis of nodal points located on the biometric human faces. These techniques also help in



differentiating twin faces. In this project we have used Particle Swarm Optimization (PSO) for the optimization of eigenfaces. As faces are non-rigid objects, the problem of face detection becomes a difficult issue.

The model proposed in our project also filters out cartoon and irrelevant human faces, half profile faces etc. with high speed and high detection ratio. The main problem is to find efficient techniques for the feature extraction and extracting eigen vectors from query image (QI) which further helps to optimize the face detection applications. In addition, this would also help understanding the face localization and face recognition. Particle Swarm Optimization (PSO) will be used to select the optimal features.

In the past several years, numerous methods have been developed for Face Recognition. These methods are:

Principle Component Analysis (PCA), Face Recognition Using LDA-Based Algorithms, Independent component analysis and support vector machines, Face recognition using holistic Fourier invariant features, Face Recognition Using Particle Swarm Optimization-Based Selected Features. Existing methods for Face Detection are Face detection in complex background, Voila Jones Face Detection, Face Detection Using AdaBoost, Omni Face Detection.

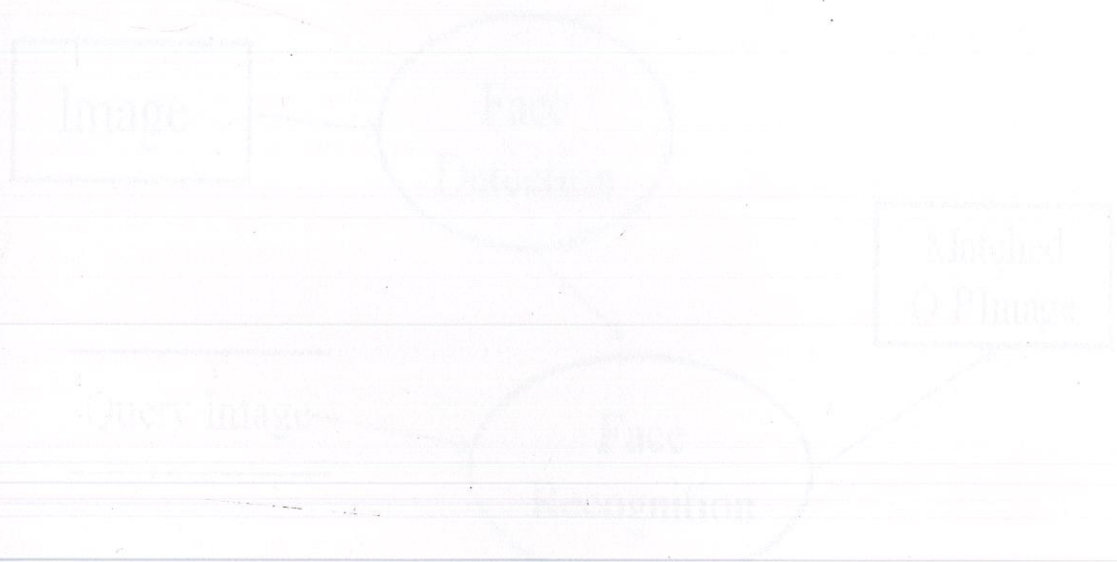
The present work proposes face detection methods along with the optimal feature extraction and selection of eigen vector values. There is a section which presents the problem definition in detail and in another section we present a model based on hybridization of eigen vectors, feature vectors, PSO and nodal points concept.



### 1.3 Objective

Given an input image, objective is to detect the faces from that image and then send that face images for the face recognition with respect to a query image, based on some features extracted from the input image.

The input is image of group of objects/Faces, then this image is fed into the system, then the system detects the Faces present in the image. Next step is to extract the Region of Interests (ROI) from the input image. After extracting the cropped Region of Interest (ROI) from the image it is compared with the query image. On comparing the features extracted from the given images and the cropped region of Interest (ROI) the best possible matched is set as the output. Image Processing techniques to extract the features from the image which is not directly present or available in the image.





## 1.4 Process Description

### 1.4.1 Level 0 DFD

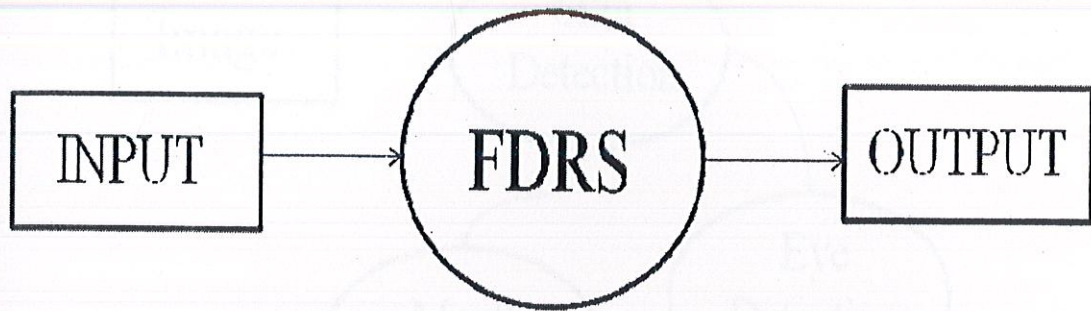


Figure 1.2.DFD Level 0

### 1.4.2 Level 1 DFD

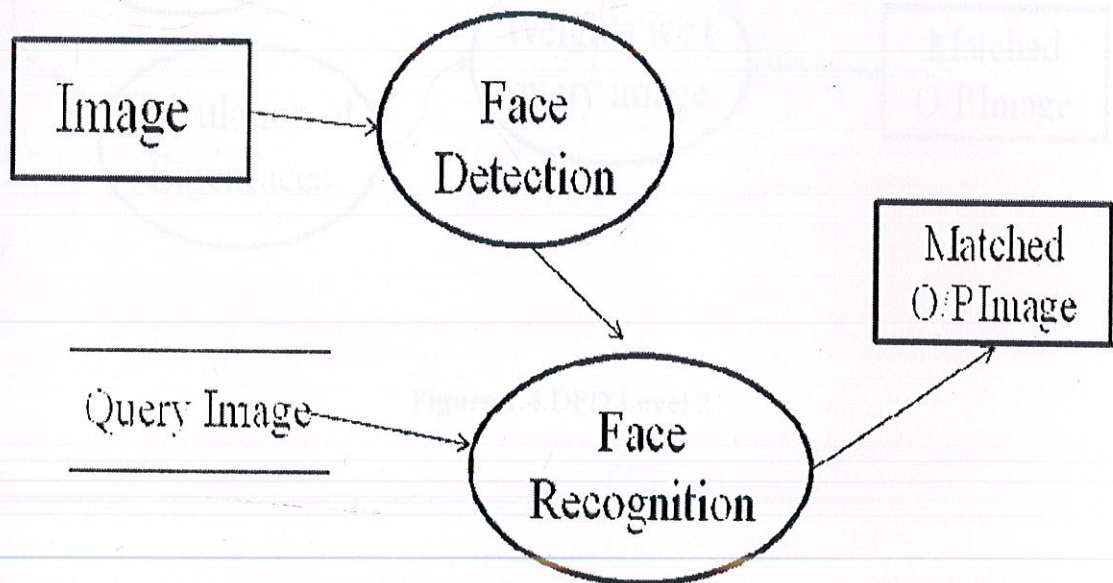


Figure 1.3.DFD Level 1



### 1.4.3 Level 2 DFD

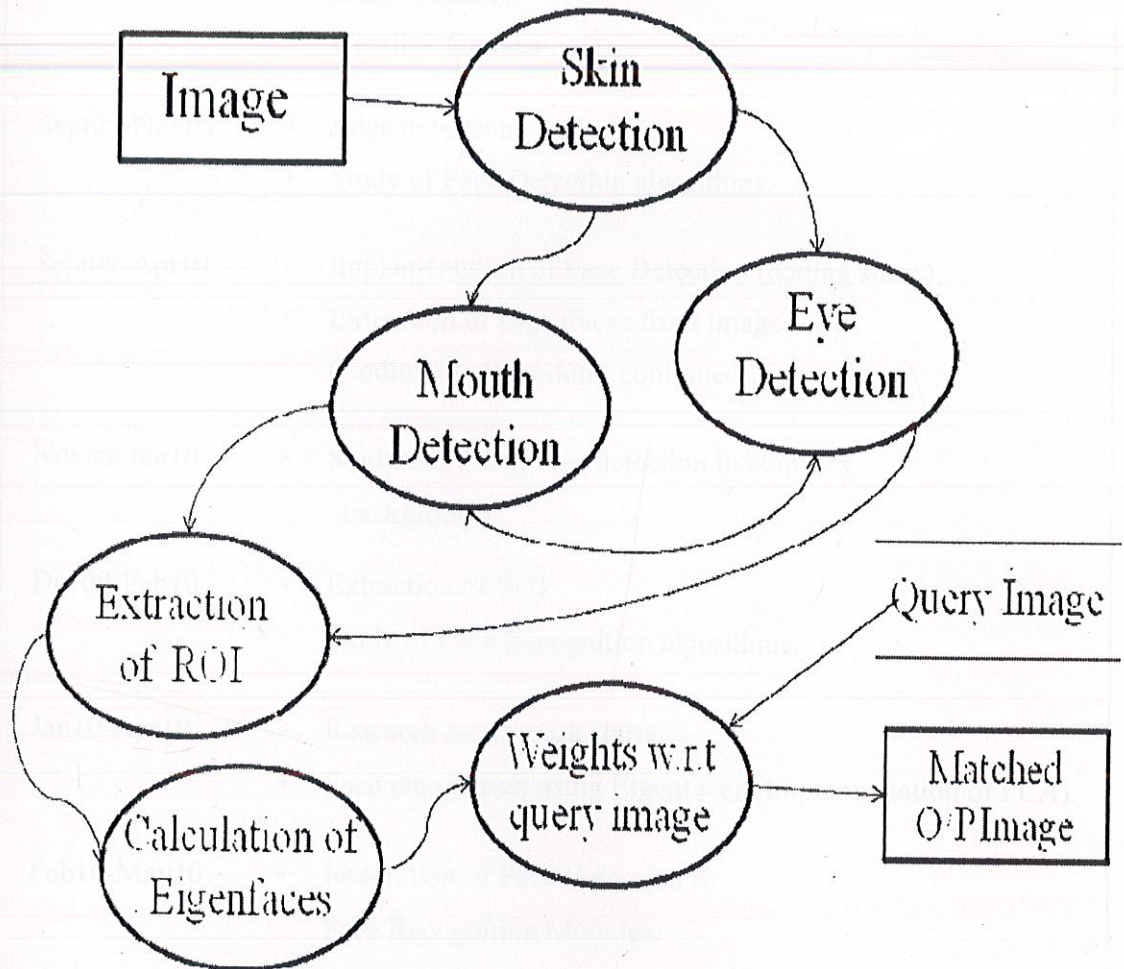


Figure 1.4.DFD Level 2



### 1.5 Time Line of Project

Month	Task
July09-Sept09	<ul style="list-style-type: none"> <li>• General study of images.</li> <li>• Image features.</li> <li>• Haralick features.</li> </ul>
Sept09-Nov09	<ul style="list-style-type: none"> <li>• Edge detection.</li> <li>• Study of Face Detection algorithms.</li> </ul>
Sept09-Apr10	<ul style="list-style-type: none"> <li>• Implementation of Face Detection (coding starts).</li> <li>• Extraction of Eigenfaces from images (Coding of all modules continued).</li> </ul>
Nov09-Jan10	<ul style="list-style-type: none"> <li>• Study of PCA &amp; face detection in complex background.</li> </ul>
Dec09-Feb10	<ul style="list-style-type: none"> <li>• Extraction of ROI.</li> <li>• Study of Face Recognition algorithms.</li> </ul>
Jan10-Apr10	<ul style="list-style-type: none"> <li>• Research paper work starts.</li> <li>• Face recognition using Eigenfaces (Implementation of PCA).</li> </ul>
Feb10-May10	<ul style="list-style-type: none"> <li>• Integration of Face Detection &amp; Face Recognition Modules.</li> </ul>
March10	<ul style="list-style-type: none"> <li>• GUI Design.</li> </ul>
Mar10-Apr10	<ul style="list-style-type: none"> <li>• Testing &amp; results.</li> <li>• Enhancement of results.</li> </ul>
Apr10-May10	<ul style="list-style-type: none"> <li>• Submission of Research paper.</li> </ul>

**Figure 1.5.**Time Line of Project



## 1.6 Gantt Chart of Project

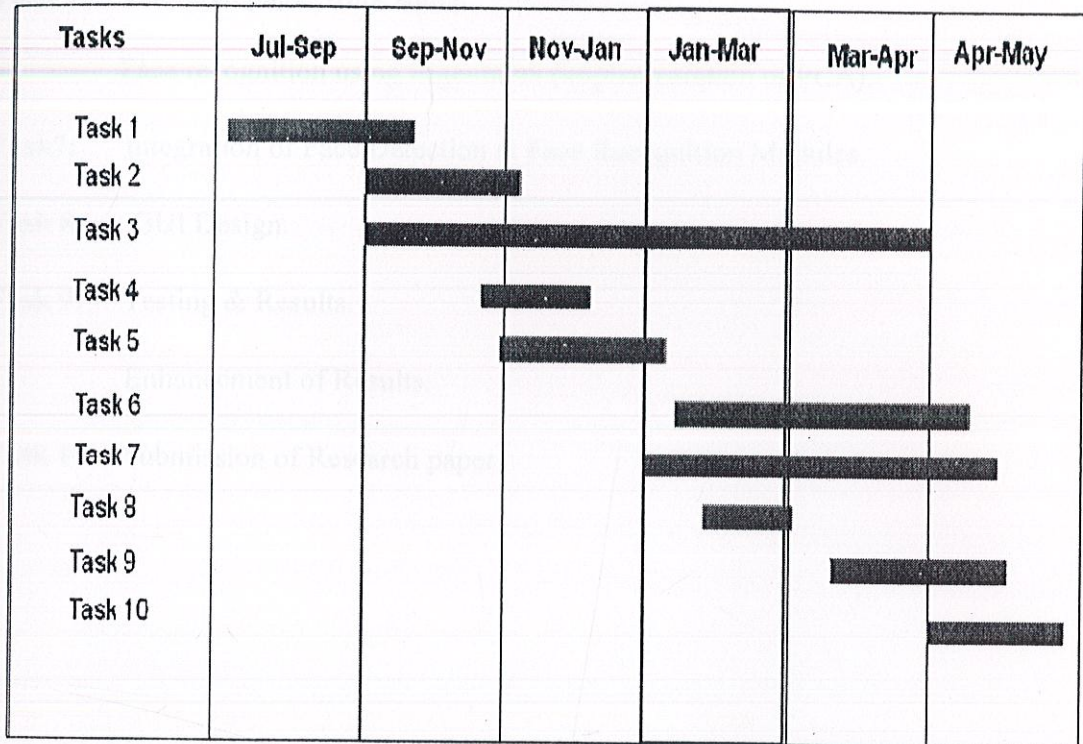


Figure 1.6. Gantt Chart of Project

## 1.7 Tasks of Project

**Task1:** General study of images.

Image features.

Haralick features.

**Task 2:** Edge detection.

Study of Face Detection algorithms

**Task 3:** Implementation of Face Detection (coding starts).

Extraction of Eigenfaces from images (coding continued of all modules).



**Task 4:** Study of PCA & face detection in complex background.

**Task 5:** Extraction of ROI.

Study of Face Recognition algorithms.

**Task 6:** Research paper work starts.

Face recognition using Eigenfaces (implementation of PCA).

**Task7:** Integration of Face Detection & Face Recognition Modules.

**Task 8:** GUI Design.

**Task 9:** Testing & Results.

Enhancement of Results.

**Task 10:** Submission of Research paper.



Implementation includes,

1. Basic Prototype.
2. Face Detection.
3. Region of interest (ROI).
4. Face Recognition.
5. Optimization using PSO

Each module will be processed individually one after the other and relevant techniques are discussed in the coming sections.

#### 2.1 Steps in Face Detection & Recognition System (FDRS)

##### 2.1.1 Face Detection in Input Image.



**Figure2.1. Input Image**



Rectangular boxes coloured red shows that these are the face images detected from the input image. These detected faces will go recognition process with respect to the provided query image.

### 2.1.2 Extracting ROI (Region of Interest) from the Input Image

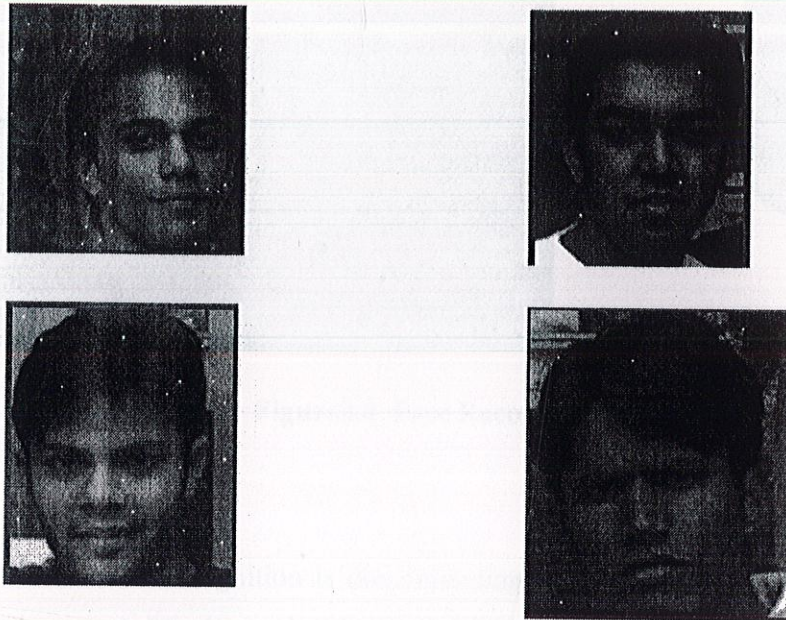


Figure2.2. ROI Images

### 2.1.3 Query Image.

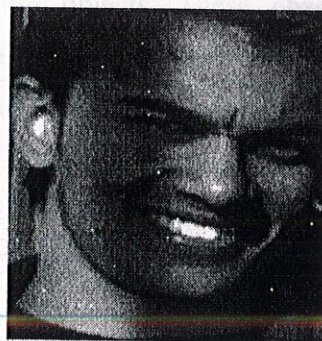
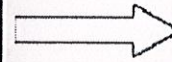
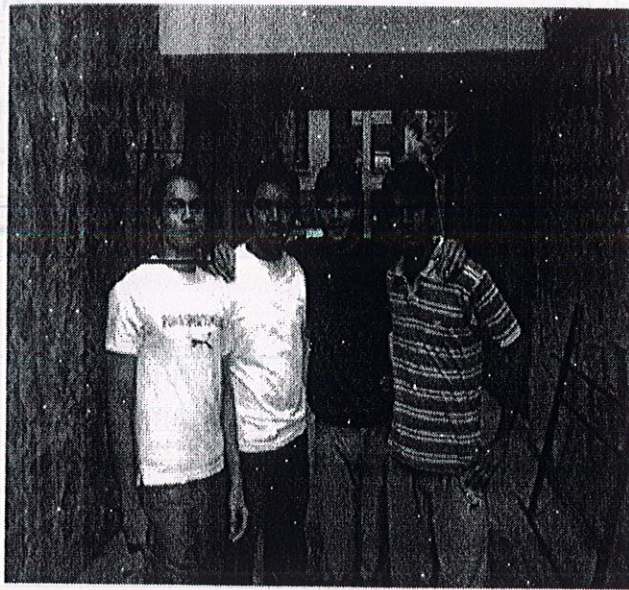


Figure2.3. Query Image

### 2.1.4Face Recognition





**Figure2.4. Face Recognised Image**

The task of facial recognition is discriminating input signals (image data) into several classes (persons). The input signals are highly noisy (e.g. the noise is caused by differing lighting conditions, pose etc.), yet the input images are not completely random and in spite of their differences there are patterns which occur in any input signal. Such patterns, which can be observed in all signals, could be - in the domain of facial recognition - the presence of some objects (eyes, nose, mouth) in any face as well as relative distances between these objects. These characteristic features are called eigenfaces in the facial recognition domain (or principal components generally). They can be extracted out of original image data by means of a mathematical tool called Principal Component Analysis (PCA).



## 2.2 Flow of the System

The basic flow of our proposed model includes the different phases in which we take an input image in our first phase, which undergoes the face detection and the extraction of the region of interest (ROI) of that input image. In further phases we recognize and optimize the extracted faces on the basis of query image and then find the best possible match for our query image.

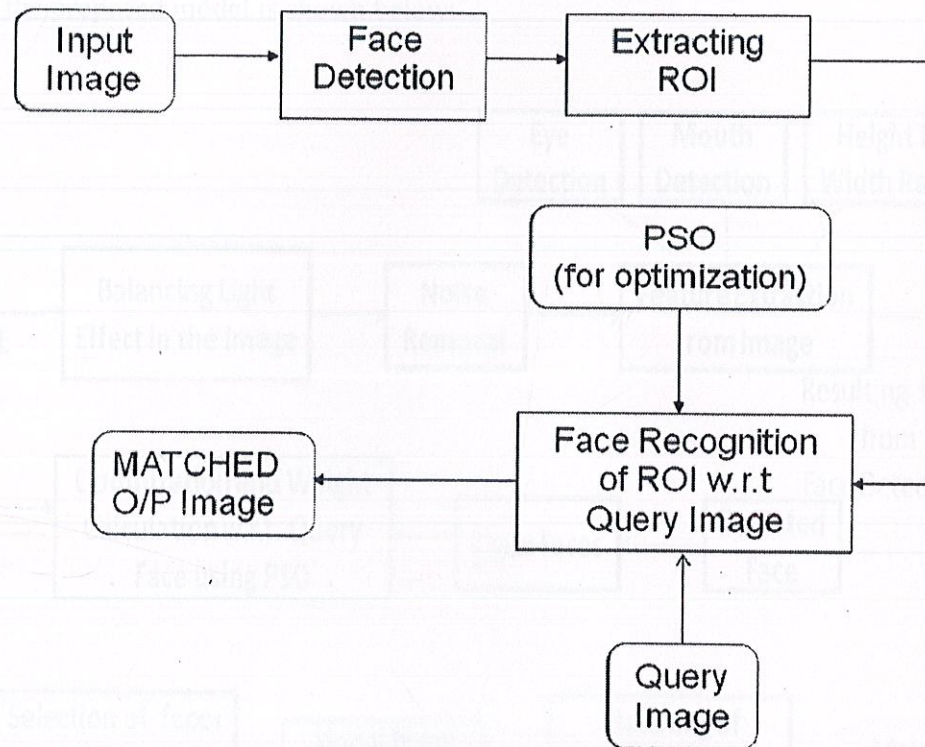


Figure 2.5. Flow of the system

The normal flow of the system is described by the following pseudo code:

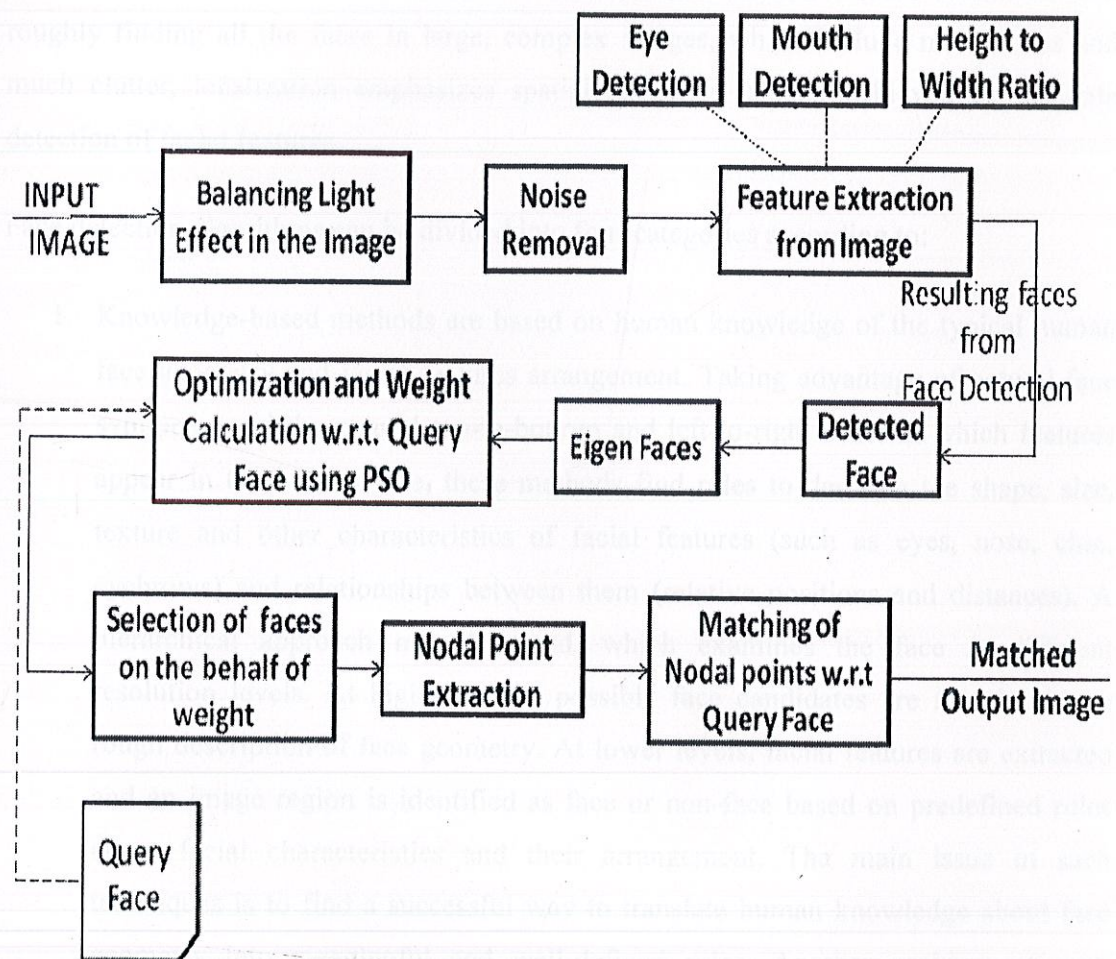
*Pseudo code:*

- i) Input the image in the system.
- ii) Get the present face detected in the input image.
- iii) Extract the ROI (Region Of Interest).
- iv) Go under the face recognition with respect to the query image and optimizing with PSO simultaneously.



- v) Show the results according to the matched outputs.
- vi) Go to step 1.

This model takes an image as input for face detection and face recognition. The image is intensified for balancing the light. After balancing the light, the noise in the image is removed and then the features like eye, mouth, and height to width ratio etc are extracted. With the help of the extracted features, faces are detected which are further processed to obtain eigenfaces for face recognition. The eigenfaces thus obtained are optimized using PSO and after that nodal point analysis is done to obtain the final efficient result. The flow of the proposed model is shown below:



**Figure 2.6.**Block Diagram of the Proposed Model



### FACE DETECTION

Face detection is the first stage of an automatic face recognition system, since a face has to be located in the input image before it is recognized. A definition of face detection could be: given an image, detect all faces in it (if any) and locate their exact positions and size. Usually, face detection is a two-step procedure: first the whole image is examined to find regions that are identified as "face". After the rough position and size of a face are estimated, a localization procedure follows which provides a more accurate estimation of the exact position and scale of the face. So while face detection is most concerned with roughly finding all the faces in large, complex images, which include many faces and much clutter, localization emphasizes spatial accuracy, usually achieved by accurate detection of facial features.

Face detection algorithms can be divided into four categories according to:

1. Knowledge-based methods are based on human knowledge of the typical human face geometry and facial features arrangement. Taking advantage of natural face symmetry and the natural top-to-bottom and left-to-right order in which features appear in the human face, these methods find rules to describe the shape, size, texture and other characteristics of facial features (such as eyes, nose, chin, eyebrows) and relationships between them (relative positions and distances). A hierarchical approach may be used, which examines the face at different resolution levels. At higher levels, possible face candidates are found using a rough description of face geometry. At lower levels, facial features are extracted and an image region is identified as face or non-face based on predefined rules about facial characteristics and their arrangement. The main issue in such techniques is to find a successful way to translate human knowledge about face geometry into meaningful and well-defined rules. Another problem of such techniques is that they do not work very well under varying pose or head orientations.



2. Feature invariant approaches aim to find structural features that exist even when the viewpoint or lighting conditions vary and then use these to locate faces. Different structural features are being used: facial local features, texture, and shape and skin color. Local features such as eyes, eyebrows, nose, and mouth are extracted using multi-resolution or derivative filters, edge detectors, morphological operations or thresholding. Statistical models are then built to describe their relationships and verify the existence of a face. Neural networks, graph matching, and decision trees were also proposed to verify face candidates. Skin color is another powerful cue for detection, because color scene segmentation is computationally fast, while being robust to changes in viewpoint, scale, shading, to partial occlusion and complex backgrounds. The color-based approach labels each pixel according to its similarity to skin color, and subsequently labels each sub-region as a face if it contains a large blob of skin color pixels. It is sensitive to illumination, existence of skin color regions, occlusion, and adjacent faces. There are also techniques that combine several features to improve the detection accuracy. Usually, they use features such as texture, shape and skin color to find face candidates and then use local facial features such as eyes, nose and mouth to verify the existence of a face. Template-based methods. To detect a face in a new image, first the head outline, which is fairly consistently roughly elliptical, is detected using filters, edge detectors, or silhouettes. Then the contours of local facial features are extracted in the same way, exploiting knowledge of face and feature geometry. Finally, the correlation between features extracted from the input image and predefined stored templates of face and facial features is computed to determine whether there is face present in the image. Template matching methods based on predefined templates are sensitive to scale, shape and pose variations. To cope with such variations, deformable template methods have been proposed, which model face geometry using elastic models that are allowed to translate, scale and rotate. Model parameters may include not only shape, but intensity information of facial features as well.
3. Appearance-based methods. While template-matching methods rely on a predefined template or model, appearance-based methods use large numbers of examples (images of faces and \ or facial features) depicting different variations



(face shape, skin color, eye color, open\closed mouth, etc). Face detection can be viewed as a pattern classification problem with two classes: "face" and "non-face". The "non-face" class contains images that may depict anything that is not a face, while the "face" class contains all face images. Statistical analysis and machine learning techniques are employed to discover the statistical properties or probability distribution function of the pixel brightness patterns of images belonging in the two classes. To detect a face in an input image, the whole image is scanned and image regions are identified as "face" or "non face" based on these probability functions. Well-known appearance-based methods used for face detection are eigenfaces , LDA , neural networks, support vector machines and hidden Markov models.

Face detection algorithm that can detect human face under the different lighting conditions with high speed and high detection ratio. Moreover, our method gives good performance if face with complex backgrounds, cartoon/human face discrimination, half-profile face, and some facial variations. By applying colour-based technique, we separate the skin regions and non-skin regions in YCbCr colour space. We adopt lighting compensation technique and nonlinear colour transformation to solve the problem of different lighting conditions. By applying feature-based technique, we extract the facial features according to human eyes, mouth, and height to width ratio of face.

Due to the digitization of data and advances in technology, it has become extremely easy to obtain and store the large quantities of data, particularly Multimedia data. Fields ranging from Commercial to Military need to analyze these data in an efficient and fast manner. Presently, tools for mining images are few and require human intervention. Feature selection and extraction is the pre-processing step of Image Mining. Obviously this is a critical step in the entire scenario of Image Mining. Our approach to mine from Images – to extract patterns and derive knowledge from large collections of images, deals mainly with identification and extraction of unique features for a particular domain. Though there are various features available, the aim is to identify the best features and thereby extract relevant information from the images. We have tried various methods for extraction; the features extracted and the techniques used are evaluated for their



contribution to solving the problem. Experimental results show that the features used are sufficient to identify the patterns from the Images.

The basic problem to be solved is to implement an algorithm for detection of faces in an image. At a first glance the task of face detection may not seem so overwhelming especially considering how easy it is solved by a human. However there is a stark contrast to how difficult it actually is to make a computer successfully solve this task. In order to ease the task Viola-Jones limits themselves to full view frontal upright faces. That is, in order to be detected the entire face must point towards the camera and it should not be tilted to any side.

### 3.1 Proposed Method for Face Detection

Due to colour and feature-based detections can fast and accurately find human, many researchers combine these two methods to obtain real human face in a picture. However, the traditional colour-based method is hard to detect the skin-colour for the case of different lighting condition, and the typical feature-based method has high computation complexity. In this section, we propose a new lighting compensation scheme to overcome the problem of colour-based method and simplify the feature-based detection. A system overview of our face detection algorithm is illustrated in Fig.11, and the details are explained as follows.

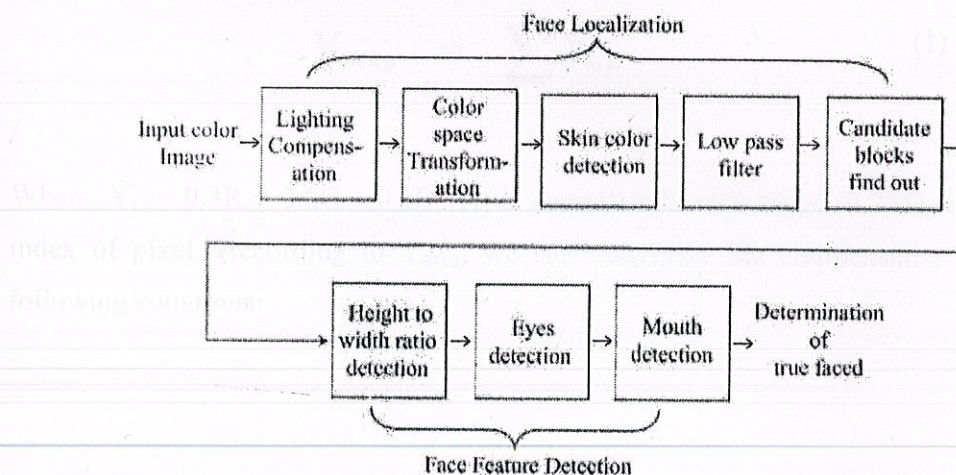


Figure 3.1. The flow of proposed method for face detection.



### 3.2 Face Detection Algorithm

Human face detection plays an important role in many applications such as video surveillance, face recognition, and faceimage database management. This paper describes a fast face detection algorithm with accurate results. We use lighting compensation to improve the performance of colour based scheme, and reduce the computation complexity of feature- based scheme. Our method is effective on facial variations such as dark/bright vision, close eyes, open moth, a half- profile face, and pseudo faces. It is worth stressing that our algorithm can also discriminate cartoon and human face correctly. The experimental results show that our approach can detect a frame in 111 msec with the 92.3% detection rate.

#### 3.2.1 Colour space transformation and lighting compensation

In order to apply to the real-time system, we adopt skin-colour detection as the first step of face detection. Due to YCbCr colour space transform is faster than other approaches, we select this transform to detect human skin. However, the luminance of every image is different. It results that every image has different colour distribution. Therefore, our lighting compensation is based on luminance to modulate the range of skin-colour distribution. First, we compute the average luminance  $Y_{avg}$  of input image.

$$Y_{avg} = \sum Y_{i,j} \quad (1)$$

Where,  $Y_{ij} = 0.3R + 0.6G + 0.1B$ ,  $Y_{ij}$  is normalized to the range (0,255), and  $i, j$  are the index of pixel. According to  $Y_{avg}$ , we can determine the compensated image  $C_{ij}$  by following equations:



$$R'_{ij} = (R_{ij})^{\tau} \quad (2)$$

$$G'_{ij} = (G_{ij})^{\tau} \quad (3)$$

$$C_{ij} = \{R'_{ij}, G'_{ij}, B_{ij}\}, \quad (4)$$

$$\tau = \begin{cases} 1.4, & Y_{avg} < 64 \\ 0.6, & Y_{avg} > 192 \\ 1, & \text{otherwise.} \end{cases}$$

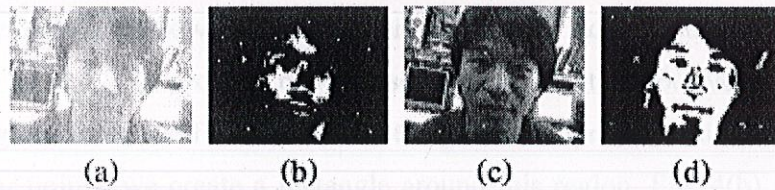
Note that we only compensate the colour of R and G to reduce computation. Due to chrominance (Cr) can represent human skin well, we only consider Cr factor for colour space transform to reduce the computation. Cr is defined as follow:

$$Cr = 0.5R' - 0.419G' - 0.081B \quad (5)$$

In Eq. (5) we can see that R' and G' are important factors due to their high weight. Thus, we only compensate R and G to reduce computation. According to Cr and experimental experience, we define the human skin by a binary matrix:

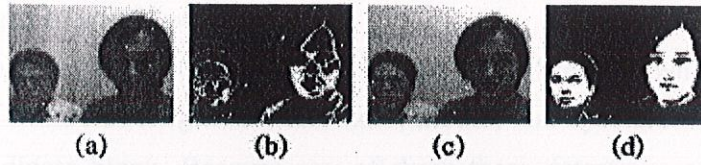
$$S_{ij} = \begin{cases} 0, & 10 < C_r < 45 \\ 1, & \text{otherwise} \end{cases} \quad (6)$$

Where, "0" is the white point, and "1" is black point. Fig. 2 and Fig. 3 show the compensation effect on bright and dark image respectively. We can see that result shown in Fig. 2(d) and Fig. 3(d) are good enough for skin-color detection.

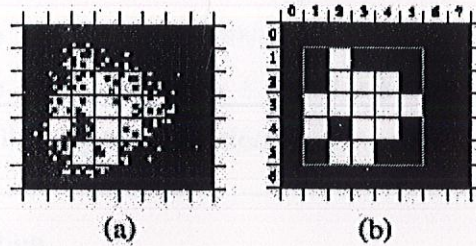


**Figure 3.2.** An example of bright image compensation: (a) original bright image (b) the  $S(i,j)$  without compensation (c) compensated image (d) the  $S(i,j)$  of compensated image.





**Figure 3.3.** An example of dark image compensation: (a) original dark image (b) the  $S(i,j)$  without compensation (c) compensated image (d) the  $S(i,j)$  of compensated image.



**Figure 3.4.** (a) An example of  $S(i,j)$  (b) remove noise by the  $5 \times 5$  low pass filter.

### 3.2.2 High frequency noisy removing

In order to remove high frequency noise fast, we implement a low pass filter by a  $5 \times 5$  mask. First, we segment  $S_{ij}$  into  $5 \times 5$  blocks, and calculate how many white points in a block. Then, every point of a  $5 \times 5$  block is set to white point when the number of white points is greater than half number of total points. On the other hand, if the number of black points is more than a half, this  $5 \times 5$  block is modified to a complete black block. Fig. 4(b) shows an example that we remove high frequency noise from Fig. 4(a). Although this fast filter will bring block effect, it can be disregarded due to that our target is to find where is the human skin.

### 3.2.3 Skin-Colour Blocks

After performing the low pass filter, there are several skin colour regions may be human face will be in  $S_{ij}$ . In order to mark these regions, we store fore vertices of rectangle for every region. First, we find the leftmost, rightmost, upmost, and down most points. By these four points, we create a rectangle around this region. Fig. 4(b) shows an example that store (1, 1), (1, 5), (5, 1), and (5, 5) to describe the candidate region. Thus, we can get several skin-colour blocks called candidate blocks to detect facial feature.



### 3.2.4 Height to width ratio detection

After the step of face localization, we can get several regions which may be human face. Then, the feature of height to width ratio, mouth, and eyes are detected sequentially for every candidate block. Because any of these three detections can reject the candidate blocks, low computation module has high priority to process. Height to width ratio is a very fast and simple detection. Let the size of candidate block is  $h \times w$ . We define that if the height to width ratio ( $h:w$ ) is out of range between 1.5 and 0.8, it should be not a face and this candidate block will be discarded. Note that the range is determined by experiments. If the ratio is between 1.5 and 0.8 may be a face, the block should be processed by the following two detections.

### 3.2.5 Mouth detection

After determining the height to width ratio for the candidate blocks, a more complex detection will be applied to find mouth feature. We use  $\theta$  to find the mouth pixels. The  $\theta$  value is calculated for all of the pixels in every candidate block. The  $\theta$  is defined as:

$$\theta = \cos^{-1} \left( \frac{0.5(2R' - G' - B)}{\sqrt{(R' - G')^2 + (R' - B)(G' - B)}} \right). \quad (7)$$

The pixel will be determined to be part of mouth by a binary matrix  $M$ :

$$M_{pq} = \begin{cases} 0, & \theta < 90 \\ 1, & \text{otherwise,} \end{cases} \quad (8)$$

Where, "0" means that pixel is mouth. Fig. 5(a) and (b) is an example for mouth pixel detection. In Fig. 5 (b), the mouth pixel is presented by white point. Then, we use vertical based histogram to determine whether or not it is a mouth in this block. We calculate how many mouth pixels are in the same y-coordinates, and use  $w_h$  to store the value of different y-coordinates. Fig. 5(c) illustrates an example of the histogram of Fig. 5(b). Note that the maximum value of  $w_h$  is denoted by  $w_{\max}$ , and the y-coordinate of  $w_{\max}$  is represented by  $h_m$ . Thus, we define if  $w_{\max}$  is less than 1/6 block width  $w$ , this block will



be rejected. For example, in Fig. 5(c)  $w_{\max}$  is more than  $(1/3)w$ , we can know that the mouth feature is embedded in this block.

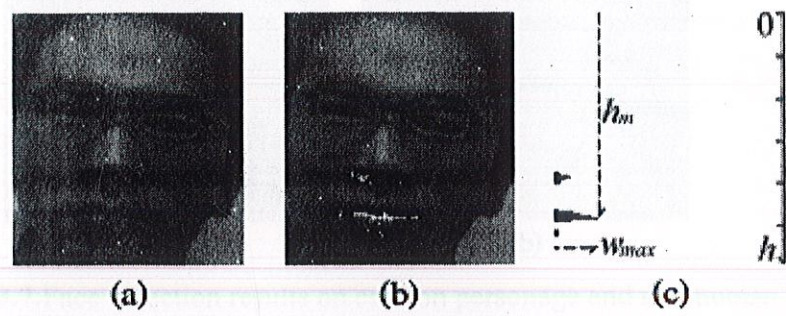
### 3.2.6 Eyes detection

After mouth detection stage, we know that the y-coordinate of mouth is  $h_m$  and the y-coordinate of eyes must smaller than  $h_m$  according to our definitions. This information let us to detect human eyes in the smaller region. The region is defined by the y-coordinate 0 to  $h_m - w_{\max}$ . Because the y-coordinate of mouth is must larger than eyes, the considered height of region must be less than  $h_m$ . An example of detecting region is shown in Fig. 6(a). Due to the deeper lineaments around human eyes, we can detect the existence of human eyes by the luminance which is slightly darker than average skin-colour. The pixels which around human eyes is defined by  $E_{hw}$ :

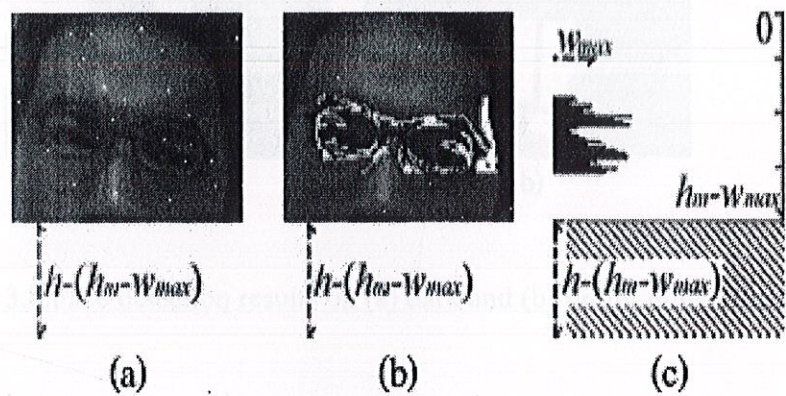
$$E_{hw} = \begin{cases} 0, & 65 < Y < 80 \\ 1, & \text{otherwise,} \end{cases} \quad (9)$$

Where,  $h = h_m - w_{\max}$ . Fig. 6(b) shows an example that we find out the pixels around eyes. Then, the vertical based histogram, illustrated in Fig. 6(c), shows the distribution of  $E_{hw}$ . In this histogram, we assume the candidate block has human eyes if there, exist a ' $\alpha$ ' value greater than a threshold  $\beta$ . Here we let  $\alpha = 0.5w_{\max}$  and  $\beta = w_{\max}$ . When we finish the eyes detection, we regard the blocks which pass three feature detections are human face.



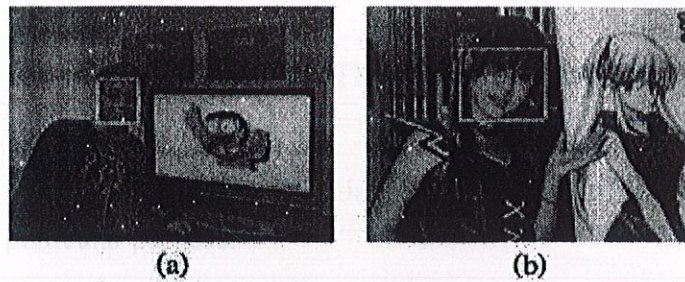


**Figure 3.5.** An example of mouth detection (a) original candidate block (b) mouth pixel detection vision (c)  $M(h, w)$  based vertical histogram.



**Figure 3.6.** An example of eyes detection (a) original candidate block (b) eyes pixel detection vision move (c)  $E(h, w)$  based vertical histogram





**Figure 3.7.**Face detection results on cartoon personage and real human.



**Figure 3.8.**Face detection results on (a) dark and (b) bright light vision.

Above are the constraints in a process of face detection that we applied in our project by making some modifications in the above discussed existing methods which enhance our system and make it more efficient:

#### 4.1 Idea behind Region of Interest analysis:



## CHAPTER IV

### EXTRACTING REGION OF INTEREST

A Region of Interest, abbreviated as **ROI**, is a selected subset of samples within a dataset identified for a particular purpose.

There are some examples of ROI in different object:

- on a waveform (1D dataset), a time or frequency interval
- on an image (2D dataset), the boundaries of an object
- in a volume (3D dataset), the contours or surfaces outlining an object
- in a time-volume (4D dataset), the outline of an object at or during a particular time interval

The concept of an ROI is commonly used in medical imaging. For example, the boundaries of a tumor may be defined on an image or in a volume, for the purpose of measuring its size. The endocardial border may be defined on an image, perhaps during different phases of the cardiac cycle, say end-systole and end-diastole, for the purpose of assessing cardiac function.

There are three fundamentally different means of encoding an ROI:

- burned in to the dataset, with a value that may or may not be outside the normal range of normally occurring values
- as separate purely graphic information, such as with vector or bitmap (rasterized) drawing elements, perhaps with some accompanying plain (unstructured) text annotation
- as separate structured semantic information (such as coded value types) with a set of spatial and/or temporal coordinates

#### 4.1 Idea behind Region of Interest analysis:

When subtracting two conditions in a group-analysis (paired-samples t-test), the significance of all voxels will be determined using a Bonferroni correction for the number of voxels. This might lead to non-significant results. It can be argued that the correction for the number of voxels (= the number of tests that is performed) is too strict, since only a few regions are of real interest to the researcher. Basically, one would like



to correct only for those voxels/tests that are of real interest. An additional advantage of selecting regions of interest (ROI's) is that the activation levels within these regions can be plotted against condition or group, so that the researcher gets an idea of activation patterns.

## 4.2 ROI selection

Careful selection of regions is an essential step in ROI analysis. There are some possible pitfalls:

- if you select ROI's based on the activation during one condition and compare these activation levels in several other conditions, it is very likely that the condition you have used to select the ROI's will show the highest activation.
- if you select the ROI's based on task activation Vs rest, then you might miss regions which are activated specifically in one condition and not in the others.
- Selecting ROI's should be based on regions which are activated in any of the conditions.
- When you have two groups (eg patients and controls) similar problems occur. Selecting ROI's from an average group t-map might lead to the missing of regions specifically activated in one of the groups.

## 4.3 How to select a region?

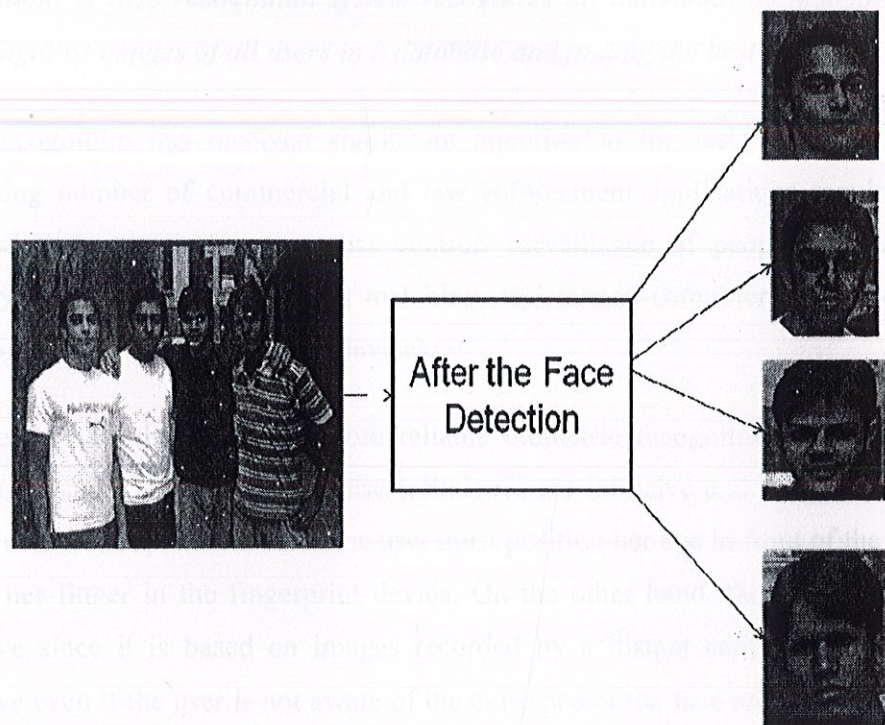
There are several strategies for selecting a ROI:

- Based on the actual activation (select region based on the voxels above some threshold)
- Drawing a sphere around some peak in the activation (SPM99)
- Based on anatomical labelling (requires a good brain mask or segmentation)

A combination of methods is also possible. For example, if the activation covers several interesting areas, you might want to separate them based on the anatomical regions areas.



In this module we extracted of region of interest from the input image. The detected faces in the image are our region of interest. Save that ROI in the memory so that we can again use that Face images to match with our query image.



**Figure 4.1.** Extracted faces from an input image

The above image shows us the ROI that we have taken from input image. These are obtained once the faces are detected. The ROI here is the rectangular box which contains the face part of the image. The techniques for detecting the faces are already discussed. Now these ROIs are stored in a database and are used for the face recognition during further enhancement.



### FACE RECOGNITION

**Definition:** *A face recognition system recognizes an individual by matching the input image against images of all users in a database and finding the best match.*

Face recognition has received significant attention in the last 15 years, due to the increasing number of commercial and law enforcement applications requiring reliable personal authentication (e.g. access control, surveillance of people in public places, security of transactions, mug shot matching, and human-computer interaction) and the availability of low-cost recording devices.

Despite the fact that there are more reliable biometric recognition techniques such as fingerprint and iris recognition, these techniques are intrusive and their success depends highly on user cooperation, since the user must position her eye in front of the iris scanner or put her finger in the fingerprint device. On the other hand, face recognition is non-intrusive since it is based on images recorded by a distant camera, and can be very effective even if the user is not aware of the existence of the face recognition system. The human face is undoubtedly the most common characteristic used by humans to recognize other people and this is why personal identification based on facial images is considered the friendliest among all biometrics.

Depending on the application, a face recognition system can be working either on identification or verification mode. In a face identification application, the system recognizes an individual by matching the input image against images of all users in a database and finding the best match. In face verification application the user claims an identity and the system accepts or rejects her claim by matching the input image against the image that corresponds to this specific identity, which can be stored either in a database or an identification card (e.g. smart card). In other words, face identification is a one-to-many comparison that answers the question "Who is the person in the input image? Is she someone in the database?", while face verification is a one-to-one comparison that answers the question "Is the person in the input image who she claims to



be?" In the sequel the term face recognition will be used for both identification and verification unless a distinction needs to be made.

Image matching usually involves three steps: 1. detection of the face in a complex background and localization of its exact position, 2. extraction of facial features such as eyes, nose, etc, followed by normalization to align the face with the stored face images, and 3. face classification or matching.

In addition, face recognition usually consists of the following four modules:

1. Sensor module, which captures face images of an individual. Depending on the sensor modality, the acquisition device maybe a black and white or color camera, a 3D sensor capturing range (depth) data, or an infrared camera capturing infrared images.
2. Face detection and feature extraction module. The acquired face images are first scanned to detect the presence of faces and find their exact location and size. The output of face detection is an image window containing only the face area. Irrelevant information, such as background, hair, neck and shoulders, ears, etc are discarded. The resulting face image is then further processed to extract a set of salient or discriminatory, local or global features, which will be used by the face classifier to identify or verify the identity of an unknown face. Such features maybe the measurements of local facial features (such as eyes, nose, mouth, etc) characteristics or global features such as transformation coefficients of global image decomposition (PCA, LDA, wavelets, etc). These features constitute the template or signature uniquely associated with the image.
3. Classification module, in which the template extracted during step 2 is compared against the stored templates in the database to generate matching scores, which reveal how identical the faces in the probe and gallery images are. Then, a decision-making module either confirms (verification) or establishes (identification) the user's identity based on the matching score. In case of face verification, the matching score is compared to a predefined threshold and based on the result of this comparison; the user is either accepted or rejected. In case of face identification, a set of matching scores between the extracted template and



the templates of enrolled users is calculated. If the template of user X produces the best score, then the unknown face is more similar to X, than any other person in the database. To ensure that the unknown face is actually X and not an impostor, the matching score is compared to a predefined threshold.

4. System database module, which is used to extract and store the templates of enrolled users. This module is also responsible for enrolling users in the face recognition system database. During the enrolment of an individual, the sensor module records images of her face. These images are called gallery images and they are used for training the classifier that will perform face recognition. Most commonly, several frontal neutral views of an individual are recorded, but often face images depicting different facial expressions (neutral, smile, laugh, anger, etc) and presence (or non-) of glasses are also acquired. Sometimes gallery images are recorded in more than one session. The time interval between different sessions may result in variations due to hairstyle, beard, make-up, etc being present in gallery images. The presence of such variations ensures a more robust face recognition performance. Given a user's set of acquired images, a set of features is extracted similarly to step 3 above, and a template that provides a compact and expressive representation of the user based on her images is generated. This is called training. The training algorithm depends on the face recognition method employed by the face recognition system. The aim of the training is to encode the most discriminative characteristics of a user based on the classifier chosen, and to determine the values of the different thresholds. Sometimes, more than one template per enrolled user is stored in the gallery database to account for different variations. Templates may also be updated over time, mainly to cope with variations due to aging.

A number of current face recognition algorithms use face representations found by unsupervised statistical methods. Typically these methods find a set of basis images and represent faces as a linear combination of those images. Principal component analysis (PCA) is a popular example of such methods. The basis images found by PCA depend only on pair wise relationships between pixels in the image database. In a task such as face recognition, in which important



information may be contained in the high-order relationships among pixels, it seems reasonable to expect that better basis images may be found by methods sensitive to these high-order statistics. Independent component analysis (ICA), a generalization of PCA, is one such method. We used a version of ICA derived from the principle of optimal information transfer through sigmoidal neurons. ICA was performed on face images in the FERET database under two different architectures, one which treated the images as random variables and the pixels as outcomes, and a second which treated the pixels as random variables and the images as outcomes. The first architecture found spatially local basis images for the faces. The second architecture produced a factorial face code. Both ICA representations were superior to representations based on PCA for recognizing faces across days and changes in expression. A classifier that combined the two ICA representations gave the best performance.

Redundancy in the sensory input contains structural information about the environment. Principal component analysis (PCA) is a popular unsupervised statistical method to find useful image representations. Consider a set of basis images each of which has pixels. A standard basis set consists of a single active pixel with intensity 1, where each basis image has a different active pixel. Any given image with pixels can be decomposed as a linear combination of the standard basis images. In fact, the pixel values of an image can then be seen as the coordinates of that image with respect to the standard basis. The goal in PCA is to find a "better" set of basis images so that in this new basis, the image coordinates (the PCA coefficients) are uncorrelated, i.e., they cannot be linearly predicted from each other. PCA can, thus, be seen as partially implementing Barlow's ideas: Dependencies that show up in the joint distribution of pixels are separated out into the marginal distributions of PCA coefficients. However, PCA can only separate pair wise linear dependencies between pixels. High-order dependencies will still show in the joint distribution of PCA coefficients, and, thus, will not be properly separated.

Some of the most successful representations for face recognition, such as eigenfaces, holons, and local feature analysis are based on PCA. In a task such as face recognition, much of the important information may be contained in the high-order relationships among the image pixels, and thus, it is important to investigate whether generalizations of



PCA which are sensitive to high-order relationships, not just second-order relationships, are advantageous. Independent component analysis (ICA) is one such generalization. A number of algorithms for performing ICA have been proposed. Here, we employ an algorithm from the point of view of optimal information transfer in neural networks with sigmoidal transfer functions. This algorithm has proven successful for separating randomly mixed auditory signals (the cocktail party problem), and for separating electroencephalogram (EEG) signals and functional magnetic resonance imaging (fMRI) signals.

There are a number of algorithms for performing ICA. We chose the info max algorithm proposed by Bell and Sejnowski, which was derived from the principle of optimal information transfer in neurons with sigmoidal transfer functions. The algorithm is motivated as follows:

Let  $\mathbf{x}$  be an  $N$ -dimensional ( $N$ -D) random vector representing a distribution of inputs in the environment. (Here, boldface capitals denote random variables, whereas plain text capitals denote matrices). Let  $\mathbf{W}$  be an invertible matrix, and an  $N$ -D random variable representing the outputs of  $N$ -neurons. Each component of  $\mathbf{y}$  is an invertible squashing function, mapping real numbers into the interval  $(0, 1)$ . Typically, the logistic function is used

1. The variables are linear combinations of inputs and can be interpreted as presynaptic activations of  $N$ -neurons. The variables can be interpreted as postsynaptic activation rates and are bounded by the interval  $(0, 1)$ . The goal in Bell and Sejnowski's algorithm is to maximize the mutual information between the environment and the output of the neural network. This is achieved by performing gradient ascent on the entropy of the output with respect to the weight matrix.

The gradient update rule for the weight matrix is as follows:

2. where  $\eta$ , the ratio between the second and first partial derivatives of the activation function, stands for transpose,  $\langle y \rangle$  for expected value,  $H(\mathbf{x})$  is the entropy of the random vector  $\mathbf{x}$ , and  $\nabla H(\mathbf{y})$  is the gradient of the entropy in matrix form, i.e., the cell in row  $i$ , column  $j$  of this matrix is the derivative of  $H(\mathbf{y})$  with respect to  $y_j$ . Computation of the matrix inverse can be avoided by employing the natural gradient, which amounts to multiplying the absolute gradient by, resulting in the following learning rule:

3. Where  $\mathbf{I}$  is the identity matrices the logistic transfer function (1) gives  $\sigma'(y) = \sigma(y)(1 - \sigma(y))$ . When there are multiple inputs and outputs, maximizing the joint entropy of the output encourages the



individual outputs to move toward statistical independence. A discussion of unsupervised learning for face recognition appears in the nonlinear transfer function is the same as the cumulative density functions of the underlying ICs (up to scaling and translation) it can be shown that maximizing the joint entropy of the outputs in also minimizes the mutual information between the individual outputs. In practice, the logistic transfer function has been found sufficient to separate mixtures of natural signals with sparse distributions including sound sources. The algorithm is speeded up by including a "sphering" step prior to learning. The row means of are subtracted, and then is passed through the whitening matrix, which is twice the inverse square root of the covariance matrix

4. This removes the first and the second-order statistics of the data; both the mean and covariance are set to zero and the variances are equalized. When the inputs to ICA are the "sphered" data, the full transform matrix is the product of the sphering matrix and the matrix learned by ICA

5. MacKay and Pearl mutter showed that the ICA algorithm converges to the maximum likelihood estimate of for the following generative model of the data:

6. Where is a vector of independent random variables, called the sources, with cumulative distributions equal to , in other words, using logistic activation functions corresponds to assuming logistic random sources and using the standard cumulative Gaussian distribution as activation functions corresponds to assuming Gaussian random sources. Thus, the inverse of the weight matrix in Bell and Sejnowski's algorithm can be interpreted as the source mixing matrix and the variables can be interpreted as the maximum-likelihood (ML) estimates of the sources that generated the data.

### 5.1 Problems and Considerations

Automatic face recognition is a particularly complex task that involves detection and location of faces in a cluttered background followed by normalization and recognition. The human face is a very challenging pattern to detect and recognize, because while its anatomy is rigid enough so that all faces have the same structure, at the same time there are a lot of environmental and personal factors affecting facial appearance. The main problem of face recognition is large variability of the recorded images due to pose, illumination conditions, facial expressions, use of cosmetics, different hairstyle, presence of glasses, beard, etc. Images of the same individual taken at different times, may



sometimes exhibit more variability due to the aforementioned factors (intrapersonal variability), than images of different individuals due to gender, race, age and individual variations (extra personal variability). One way of coping with intrapersonal variations is including in the training set images with such variations. And while this is a good practice for variations such as facial expressions, use of cosmetics and presence of glasses or beard, it may not be successful in case of illumination or pose variations. Another crucial parameter in face recognition is aging. A robust recognition system should be able to recognize an individual even after some years, especially in mug-shot matching forensic applications. This is a very challenging task, which has not been successfully addressed yet.

Recent public facial recognition benchmarks have shown that in general, the identification performance decreases linearly in the logarithm of number of people in the gallery database. Also, in a demographic point of view, it was found that the recognition rates for males were higher than for females, and that the recognition rates for older people were higher than for younger people. These tests also revealed that while the best recognition techniques were successful on large face databases recorded in well-controlled environments, their performance was seriously deteriorated in uncontrolled environments, mainly due to variations in illumination and head rotations. Such variations have proven to be one of the biggest problems of face recognition systems.

Several techniques have been proposed to recognize faces under varying pose. One approach is the automatic generation of novel views resembling the pose in the probe image. This is achieved either by using a face model (an active appearance model (AAM) or a deformable 3D model or by warping frontal images using the estimated optical flow between probe and gallery. Classification is subsequently based on the similarity between the probe image and the generated view. A different approach is based on building a pose varying eigenspace by recording several images of each person under varying pose. Representative techniques are the view-based subspace and the predictive characterized subspace. More recently, techniques that rely on 3D shape data have been proposed.

The problem of coping with illumination variations is increasingly appreciated by the scientific community and several techniques have been proposed that may be roughly



classified into two main categories. The first category contains techniques seeking illumination insensitive representations of face images. Several representations were seen to be relatively insensitive to illumination variability, e.g. the direction of the image gradient or the sum of gradient of ratios between probe and gallery images.

The second approach relies on the development of generative appearance models, able to reconstruct novel gallery images resembling the illumination in the probe images. Some of these techniques utilize a large number of example images of the same person under different illumination conditions to reconstruct novel images. Other approaches utilize a 3D range image and albedo map of the person's face to render novel images under arbitrary illumination, while others are based on a combination of the above. Finally, a third more recent approach is based on computer graphics techniques for relighting the probe image so that it resembles the illumination in gallery images.

**5.2 Eigenfaces:** are a set of eigenvectors used in the computer vision problem of human face recognition. The approach of using eigenfaces for recognition was developed by Sirovich and Kirby (1987) and used by Matthew Turk and Alex Pentland in face classification. It is considered the first successful example of facial recognition technology. These eigenvectors are derived from the covariance matrix of the probability distribution of the high-dimensional vector space of possible faces of human beings.

### 5.3 Eigenface generation

To generate a set of **eigenfaces**, a large set of digitized images of human faces, taken under the same lighting conditions, are normalized to line up the eyes and mouths. They are then all re-sampled at the same pixel resolution. Eigenfaces can be extracted out of the image data by means of a mathematical tool called principal component analysis (PCA).

The eigenfaces that are created will appear as light and dark areas that are arranged in a specific pattern. This pattern is how different features of a face are singled out to be evaluated and scored. There will be a pattern to evaluate symmetry, if there is any style of facial hair, where the hairline is, or evaluate the size of the nose or mouth. Other eigenfaces have patterns that are less simple to identify, and the image of the eigenface may look very little like a face.



The technique used in creating eigenfaces and using them for recognition is also used outside of facial recognition. This technique is also used for handwriting analysis, lip reading, voice recognition, sign language/hand gestures and medical imaging. Therefore, some do not use the term eigenface, but prefer to use 'eigenimage'. Research that applies similar eigen techniques to sign language images has also been made.

Informally, eigenfaces are a set of "standardized face ingredients", derived from statistical analysis of many pictures of faces. Any human face can be considered to be a combination of these standard faces. For example, your face might be composed of the average face plus 10% from eigenface 1, 55% from eigenface 2, and even -3% from eigenface 3. Remarkably, it does not take many eigenfaces summed together to give a fair likeness of most faces. Also, because a person's face is no longer recorded by a digital photograph, but instead as just a list of values (one value for each eigenface in the database used), much less space is taken for each person's face.

#### **5.4 Use in facial recognition**

Facial recognition was the source of motivation behind the creation of eigenfaces. For this use, eigenfaces have advantages over other techniques available, such as the system's speed and efficiency. Using eigenfaces is very fast, and able to functionally operate on lots of faces in very little time. Unfortunately, this type of facial recognition does have a drawback to consider: trouble recognizing faces when they are viewed with different levels of light or angles. For the system to work well, the faces need to be seen from a frontal view under similar lighting. Face recognition using eigenfaces has been shown to be quite accurate. By experimenting with the system to test it under variations of certain conditions, the following correct recognitions were found: an average of 96% with light variation, 85% with orientation variation, and 64% with size variation.

To complement eigenfaces, another approach has been developed called eigenfeatures. This combines facial metrics (measuring distance between facial features) with the eigenface approach. Another method, which is competing with the eigenface technique, uses 'fisherfaces'. This method for facial recognition is less sensitive to variation in lighting and pose of the face than the method using eigenfaces.



A more modern alternative to eigenfaces and fisherfaces is the active appearance model, which decouples the face's shape from its texture: it does an eigenface decomposition of the face after warping it to mean shape. This allows it to perform better on different projections of the face, and when the face is tilted.

#### Original Faces

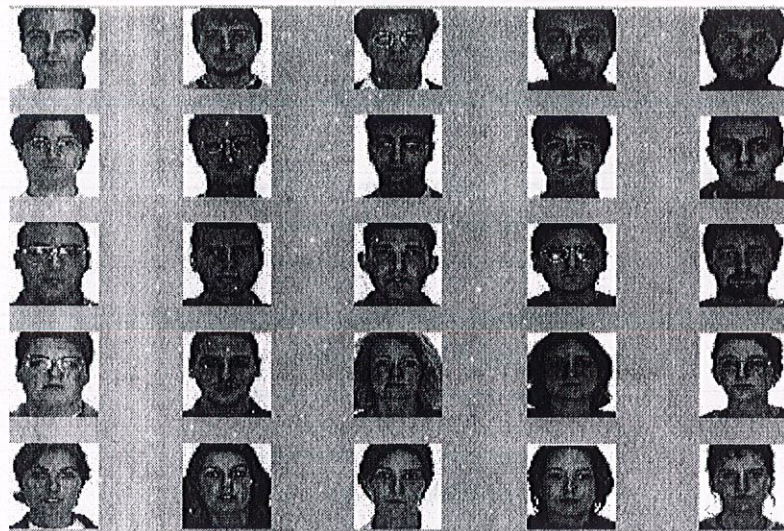


Figure 5.1. Original Faces

#### Eigenfaces

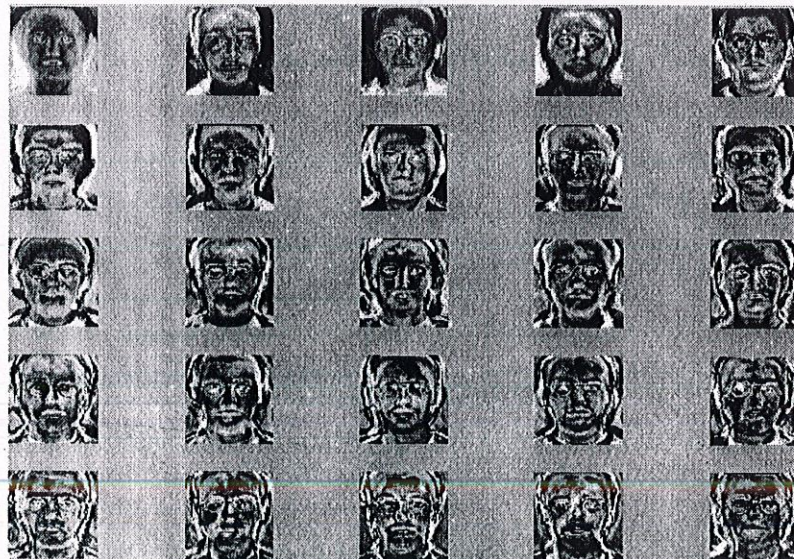


Figure 5.2. Eigen Faces



## 5.5 Principal Component Analysis (PCA)

### 5.5.1 Flow Chart:

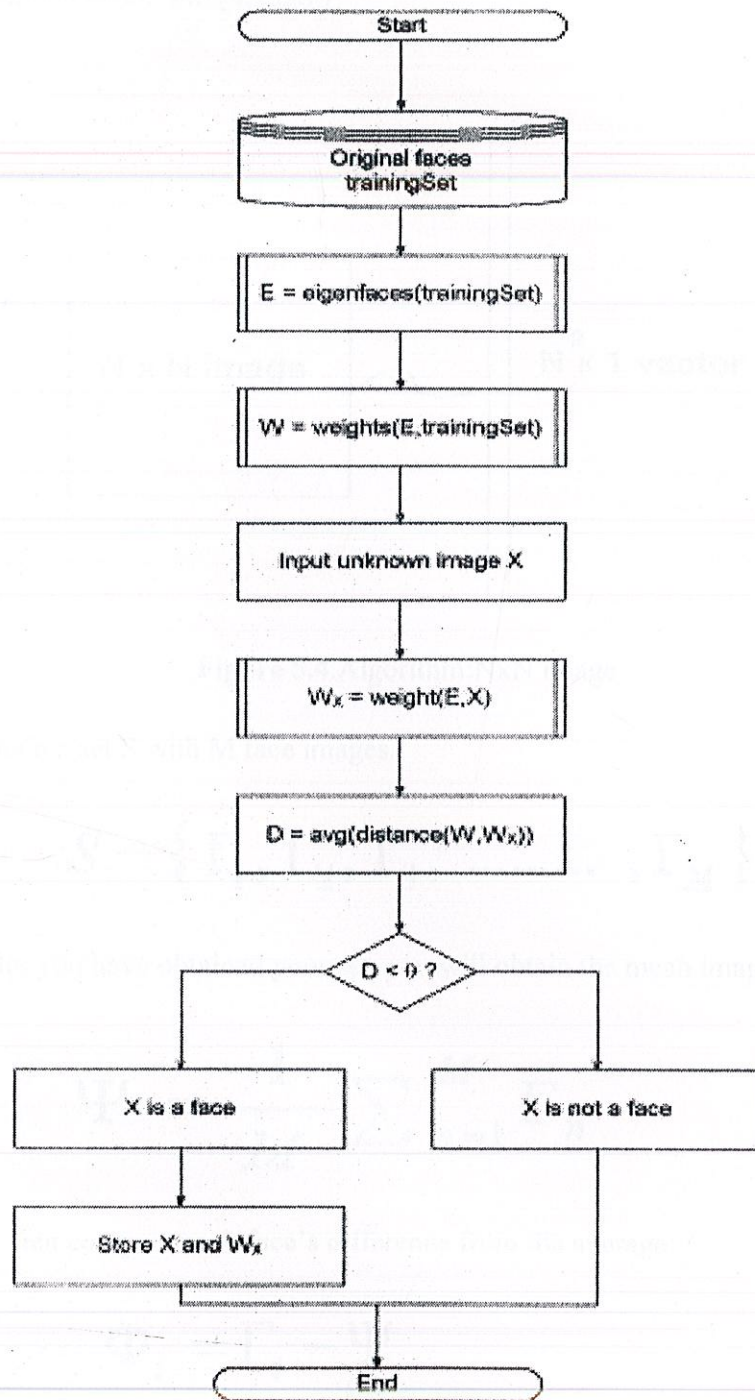


Figure 5.3. Flow Chart



### 5.5.2 Algorithm

- 1) Each face image is converted into a vector  $\Gamma_n$  of length N  
( $N = \text{imagewidth} * \text{imageheight}$ )

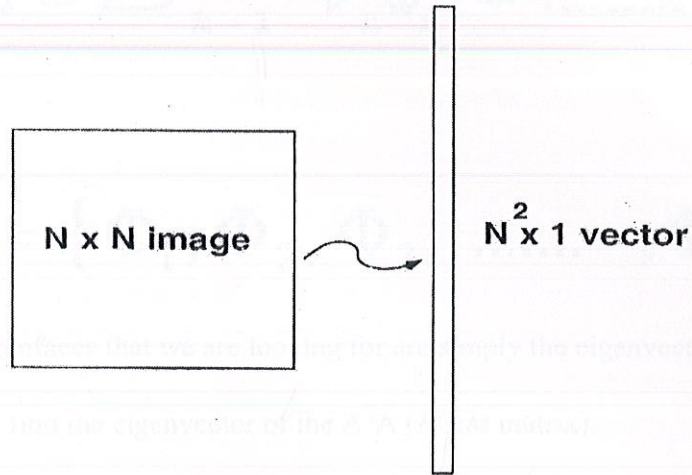


Figure 5.4. Algorithm: N x N image.

- 2) Obtain a set S with M face images.

$$S = \{ \Gamma_1, \Gamma_2, \Gamma_3, \dots, \Gamma_M \} \quad (10.a)$$

- 3) After you have obtained your set, you will obtain the mean image .

$$\Psi = \frac{1}{M} \sum_{n=1}^M \Gamma_n \quad (10.b)$$

- 4) We then compute each face's difference from the average:

$$\Phi_i = \Gamma_i - \Psi \quad (10.c)$$

- 5) We use these differences to compute a covariance matrix (C) for our dataset. The covariance between two sets of data reveals how much the sets correlate.



$$\begin{aligned}
 C &= \frac{1}{M} \sum_{n=1}^M \Phi_n \Phi_n^T = A A^T \\
 L &= A^T A \quad L_{m,m} = \Phi_m^T \Phi_m \\
 u_l &= \sum_{k=1}^M \frac{1}{\lambda_k} \Phi_k \quad l = 1, \dots, M
 \end{aligned} \tag{10.d}$$

6) Where

$$A = \{ \Phi_1, \Phi_2, \Phi_3, \dots, \Phi_n \} \tag{10.e}$$

The eigenfaces that we are looking for are simply the eigenvectors of C.

7) We will find the eigenvector of the  $A^T A$  ( $M \times M$  matrix).

8) We will have the M eigenfaces.

(Each face in the training set can be represented as a linear combination of the best K eigenvectors).

### 5.5.3. Representing Faces on the basis on Eigenfaces

Each face (minus the mean)  $\Phi_i$  in the training set can be represented as a linear combination of the best K eigenvectors.

$$\hat{\Phi}_i - mean = \sum_{j=1}^K w_j u_j, \quad (w_j = u_j^T \Phi_i) \tag{11.a}$$

Where,  $u_j$ 's are the eigenfaces.



Each normalized training face  $\Phi_i$  is represented in this basis by a vector:

$$\Omega_i = \begin{bmatrix} w_1^i \\ w_2^i \\ \dots \\ w_K^i \end{bmatrix}, \quad i = 1, 2, \dots, M \quad (11.b)$$

### 5.5.3 Face Recognition Using Eigenfaces

- Given an unknown face image  $\Gamma$  (centered and of the same size like the training faces) follow these steps:

Step 1: normalize  $\Gamma$ :  $\Phi = \Gamma - \Psi$

Step 2: project on the eigenspace

$$\hat{\Phi} = \sum_{i=1}^K w_i u_i \quad (w_i = u_i^T \Phi)$$

Step 3: represent  $\Phi$  as:  $\Omega = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_K \end{bmatrix}$

Step 4: find  $e_r = \min_l \|\Omega - \Omega^l\|$

$e_r$  will give the best possible match of input image.



## 5.6 Advantages of PCA

- The main advantages of PCA are its simple implementation, training, and very high recognition accuracy.
- It has a success rate of around 99% with just 1% chances of error.
- PCA is less affected by eye location errors.

## 5.7 Disadvantages of PCA

- Translation variant - Even if the images are shifted it won't recognize the face.
- Lighting variant- if the light intensity changes, the face won't be recognized that accurate.

For the face recognition eigenfaces are considered. The main idea that lies behind the eigenfaces is to get the eigenvector corresponding to each image known as training set which can be obtained by various algorithms like PCA. The steps for the purpose are as follows:

**Step1:** Obtain face image  $I_1, I_2, \dots, I_n$  (Training set)

**Step2:** Obtain face vector of each image

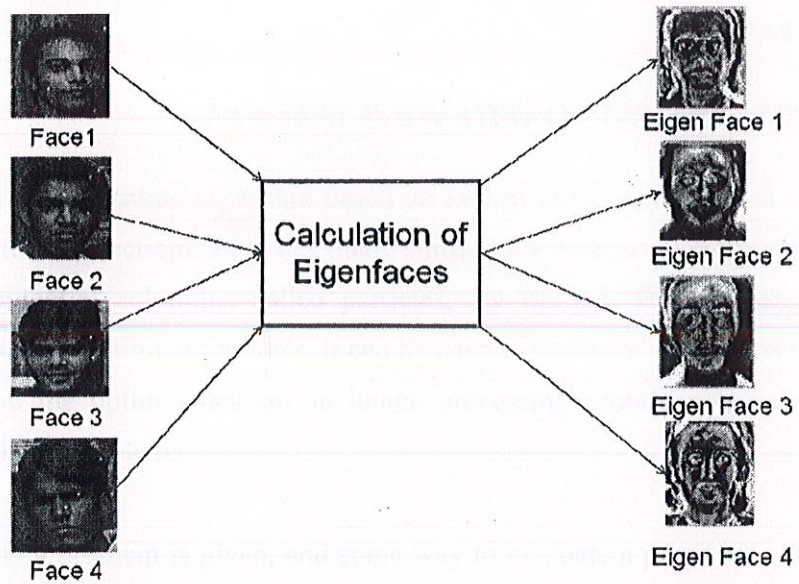
**Step3:** Obtain the mean face using face vector

**Step4:** Subtract the mean face from face vector

**Step5:** Compute the covariance matrix of face vector

**Step6:** Compute the eigenvectors (alias eigenfaces) with corresponding eigenvalues.





**Figure 5.5.**Eigenfaces obtained from the faces detected.



## PARTICLE SWARM OPTIMIZATION (PSO)

It is a stochastic optimization algorithm based on swarm intelligence. It is a concept for optimizing nonlinear functions. PSO has many similarities with genetic algorithms (GA). In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. It can be used to optimize parameters of a neural network, to find the optimal features in image processing problems and a variety of optimization related problems.

In the basic PSO a problem is given, and some way to evaluate a proposed solution to it exists in the form of a fitness function. A population of individuals called the *particles* is initialized. An iterative process to improve these candidate solutions is set in motion. The *particles* fly through the search space. The position of a particle is influenced by the best position visited by it and the position of the best particle in its neighbourhood. When the neighbourhood of a particle is the entire swarm, the best position in the neighbourhood is referred to as the global best particle, and the resulting algorithm is referred to as a *gbest* PSO. When smaller neighbourhoods are used, the algorithm is generally referred to as a *lbest* PSO. The performance of each particle (i.e. how close the particle is from the global optimum) is measured using a fitness function:

$x_i$ : The *current position* of the particle;

$v_i$ : The *current velocity* of the particle;

$y_i$ : The *personal best position* of the particle.

$\hat{y}_i$ : The *neighbourhood best position* of the particle.

The personal best position of particle  $i$  is the best position (i.e. the one resulting in the best fitness value) visited by particle  $i$  so far. Let  $f$  denote the objective function. Then the personal best of a particle at time step  $t$  is updated as

**The *gbest* model:** The position of the global best particle is given by,

$$\hat{y}(t) \in \{y_0, y_1, \dots, y_s\} = \min \{f(y_0(t)), f(y_1(t)), \dots, f(y_s(t))\} \quad (12)$$



Where,  $s$  denotes the size of the swarm. The velocity of particle  $i$  is updated using the following equation:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1r_{1,j}(t)(y_{i,j}(t) - x_{i,j}(t)) + c_2r_{2,j}(t)(\hat{y}_j(t) - x_{i,j}(t)) \quad (13)$$

Where  $w$  is the inertia weight,  $c_1$  and  $c_2$  are the acceleration constants, and  $r_{1,j}$  and  $r_{2,j}$  are factors lying between (0,1). The position of particle  $i$ ,  $x_i$ , is then updated using the following equation:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (14)$$

The basic algorithm for PSO is given below:

### 6.1 Algorithm PSO

Initialize population

Do

For particle  $i=1$  to Swarm size  $S$

if  $f(x_i) < f(p_i)$ , then  $p_i = x_i$

$p_g = \{ p_i \mid \min f(p_j) \ j=1,2,\dots,N \}$

For  $d=1$  to Dimension  $D$

Update  $v_{id}$  using equation 5.3

If  $v_i > v_{\max}$  then  $v_i = v_{\max}$

else if  $v_i < -v_{\max}$  then  $v_i = -v_{\max}$

Update  $x_{id}$  using equation 5.4

Next  $d$

Next  $i$

Until termination criteria is met.

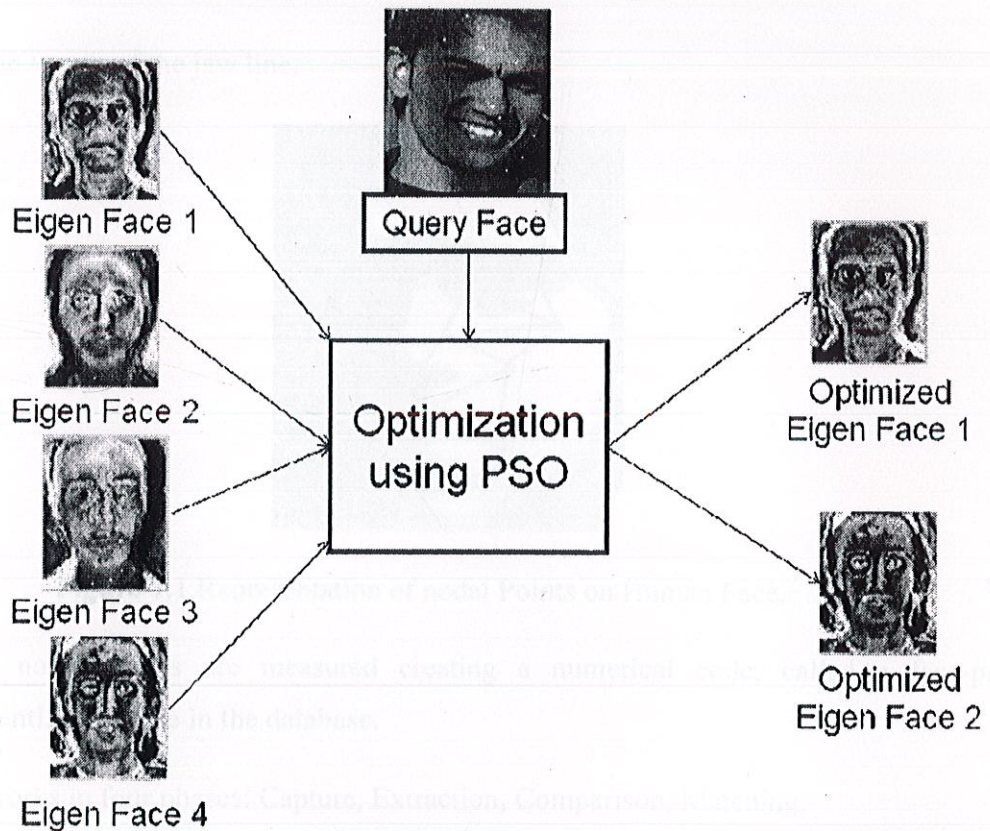
Above is the basic algorithm of PSO



## 6.2 Feature Optimization using PSO

For optimal feature selection query face is taken from the already existing pool of images and weights of the eigenfaces are calculated and optimized using PSO. Then the weights lying within the valid range are selected.

$$W_{eg} = \begin{cases} \text{Valid weight, } 0.4 < W_{eg} < 0.856 \\ \text{Invalid weight, otherwise} \end{cases} \quad (15)$$



**Figure 6.1.**Optimized faces after using PSO



## CHAPTER VII

### NODAL POINTS

Each human face has numerous, distinct landmarks, the different peaks and valleys that make up facial features, are known as nodal points. Each human face has approximately 80 nodal points. Some of these measured by the Facial Recognition Technology are:

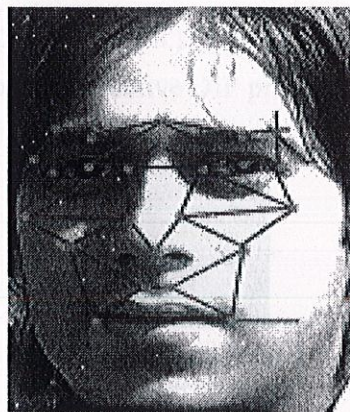
Distance between the eyes.

Width of the nose.

Depth of the eye sockets.

The shape of the cheekbones.

The length of the jaw line.



**Figure 7.1.**Representation of nodal Points on Human Face.

These nodal points are measured creating a numerical code, called a face-print, representing the face in the database.

This works in four phases: Capture, Extraction, Comparison, Matching.

**7.1 Capture:** a physical or behavioural sample is captured by the system during enrolment

**7.2 Extraction:** unique data is extracted from the sample and a template is created

**7.3 Comparison:** the template is then compared with a new sample

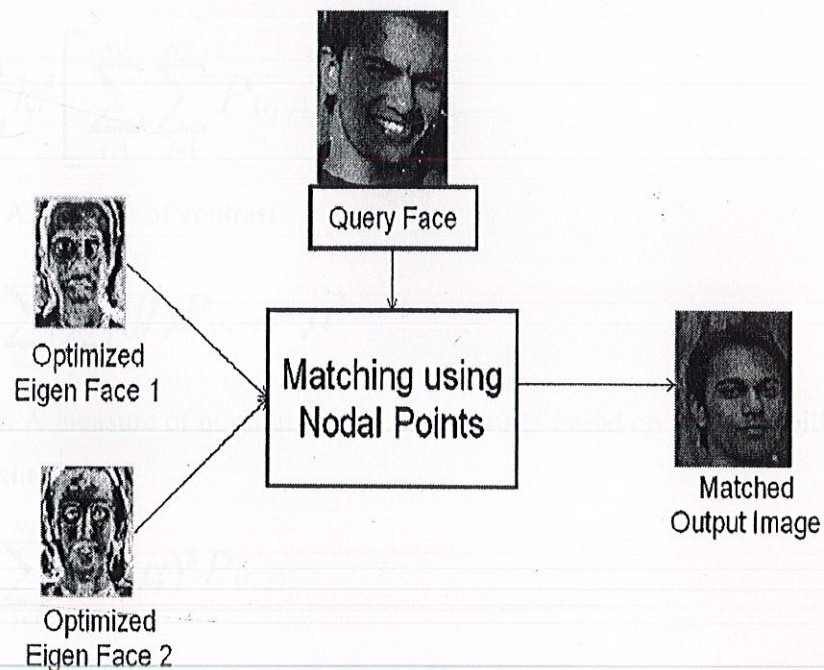


**7.4 Matching:** based on the measurements and the shapes of these nodal points the program is able to make a code of numbers for each nodal point and then decides if the features extracted from the new sample are matching or not

### 7.5 Face Matching using the Nodal Points

After optimizing the eigenfaces, matching of the faces is done using the nodal point. For this we have to construct the face matrix which can be obtained by two different methods  
1) Using the shape modal analysis 2) Using the finite method and modal analysis. The features discussed in section 3.1 help to refine the images on the basis of nodal points and thus give more efficient results differentiating the twin images.

Since the number of points of objects to be matched is different, with "one to one" restriction all the points don't get matched, so some fictitious points get added with some enhanced elements thus changing the points to "one to many". Each one of the excess nodes is fitted between the nearest matched points (of similar objects). As in the optimization phase, these matched nodal points also play a vital role for face recognition process ultimately results to very effective DIP principles.



**Figure 7.2.**Final matched image after Nodal point analysis



## FEATURES OF FACE

**8.1 Colour:** Egeria occurs in 2 colours – pink (rusty rose) and black. The picture elements can be compared to these defined spectra.

**8.2 Edge:** Edge is simply a “large change in frequency”. This is particularly creates difference between the dark Egeria and the lighter water bodies.

**8.3 Texture:** It refers to the properties held and sensations caused by the external surface of objects received through the sense of touch. **Haralick’s textural features** were considered for this work, which included the following:

**8.3.1 Energy:** A measure of homogeneity

$$f_1 = \sum_{i=1}^{Ng} \sum_{j=1}^{Ng} P_{(i,j)}^2 \quad (16)$$

**8.3.2 Correlation:** A measure of linear dependency of brightness.

$$f_2 = \sum_{k=0}^{Ng-1} k^2 \left[ \sum_{i=1}^{Ng} \sum_{j=1}^{Ng} P_{(i,j)} \right] \quad (17)$$

**8.3.3 Inertia:** A measure of contrast

$$f_3 = \frac{1}{\sigma^2} \sum_{i=1}^{Ng} \sum_{j=1}^{Ng} (ij) P_{(i,j)} - \mu^2 \quad (18)$$

**8.3.4 Entropy:** A measure of non-uniformity in the image based on the probability of co-occurrence values.

$$f_4 = \sum_{i=1}^{Ng} \sum_{j=1}^{Ng} (i - \mu)^2 P_{(i,j)} \quad (19)$$

**8.3.5 Inverse Difference Moment:** A measure of local homogeneity

$$f_5 = \sum_{i=1}^{Ng} \sum_{j=1}^{Ng} \frac{P_{(i,j)}}{1 + (i - j)^2} \quad (20)$$



## SNAPSHOTS OF IMPLEMENTATION

## 9.1 Snapshots of FDRS

## 9.1.1 Interface of FDRS

The starting user interface is as shown below in fig9.1. This is the starting interface when we run the code of FDRS. Initially, all the fields are empty except for the buttons of input image, query image, match image and exit.

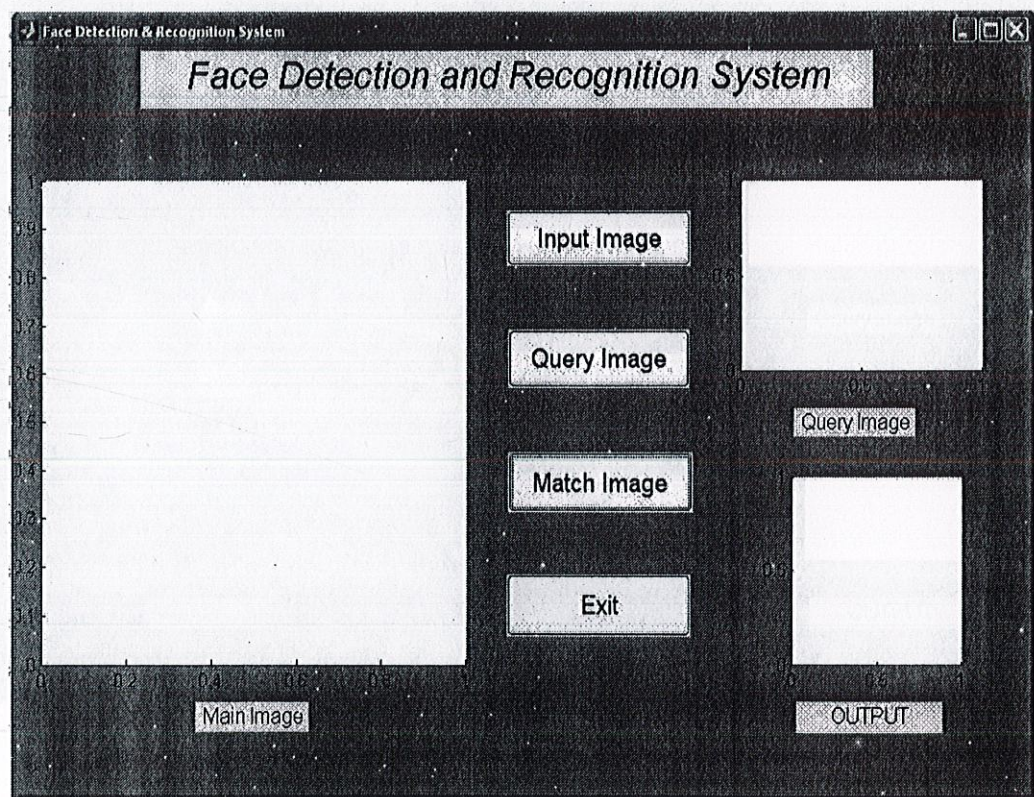


Figure 9.1.Interface of FDRS



### 9.1.2 Loading of Input Image

After starting the system, now we have to load the input image at which we have to apply the face detection and then with respect to the query image the face recognition. The image can be chosen from the existing database or simply from the computer hard disk. This can be done by clicking on the input image button of the GUI.

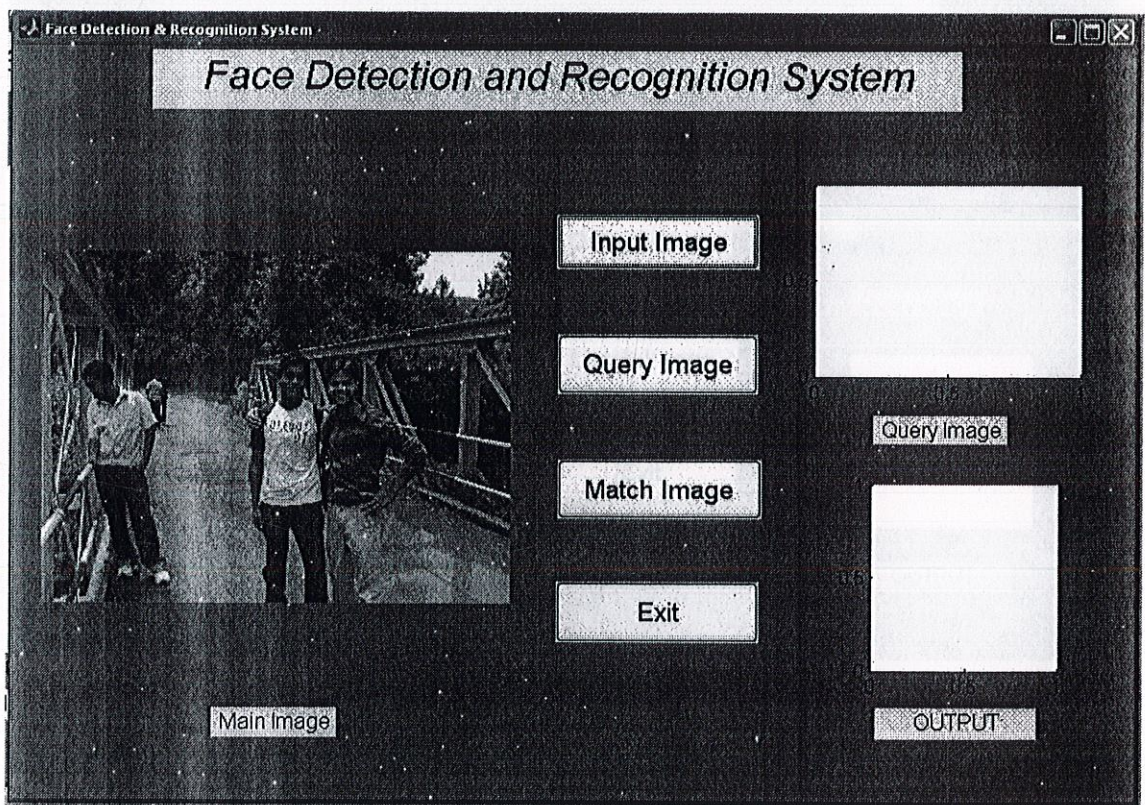


Figure 9.2.Loading of Input Image



### 9.1.3 Fetching Query Image

After loading image, now the query image is fetched into the system, which recognizes that whether the face for which we searching in the image is present or not. This can be done by simply clicking on the query image button of GUI. After fetching query image the further processing is done for matching the image.

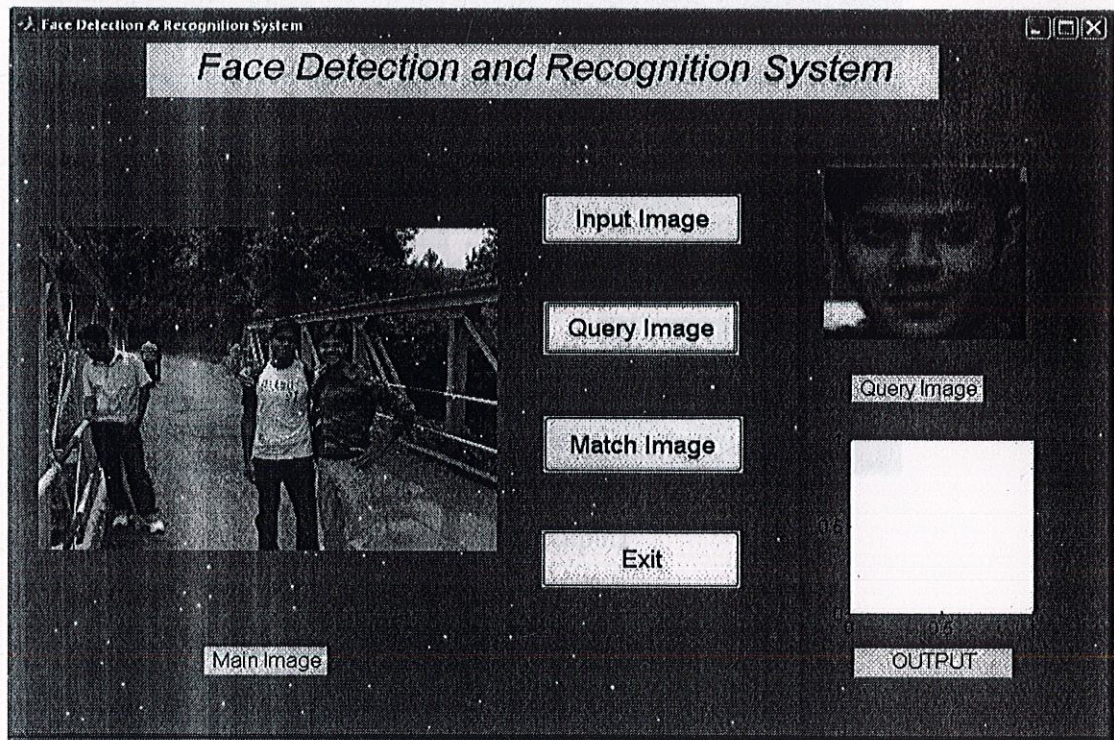


Figure 9.3.Fetching of query image



#### 9.1.4 Observed Outputs

Now the final results are observed by the match image button. We can see that output1 and output2 are perfectly matched outputs with respect to different sample images.

##### Output 1

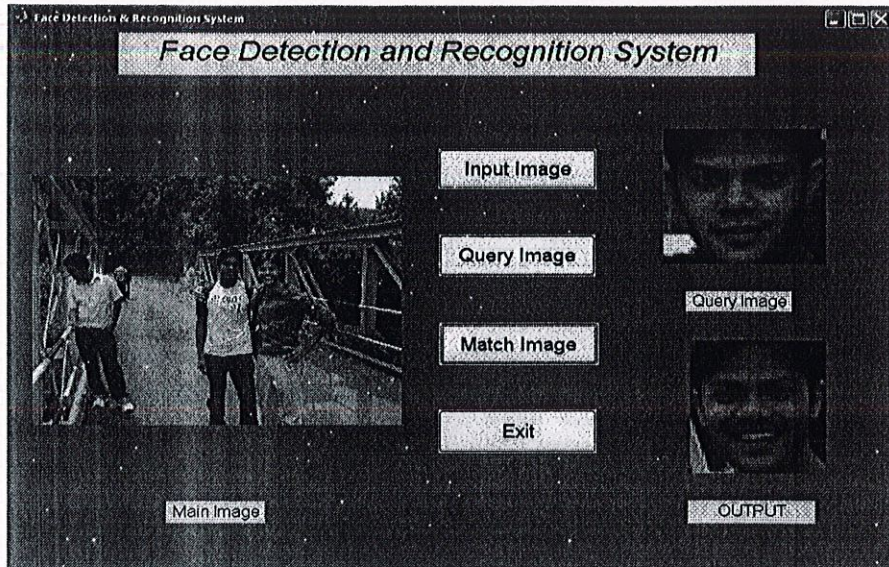


Figure 9.4.Observed Output1

##### Output 2

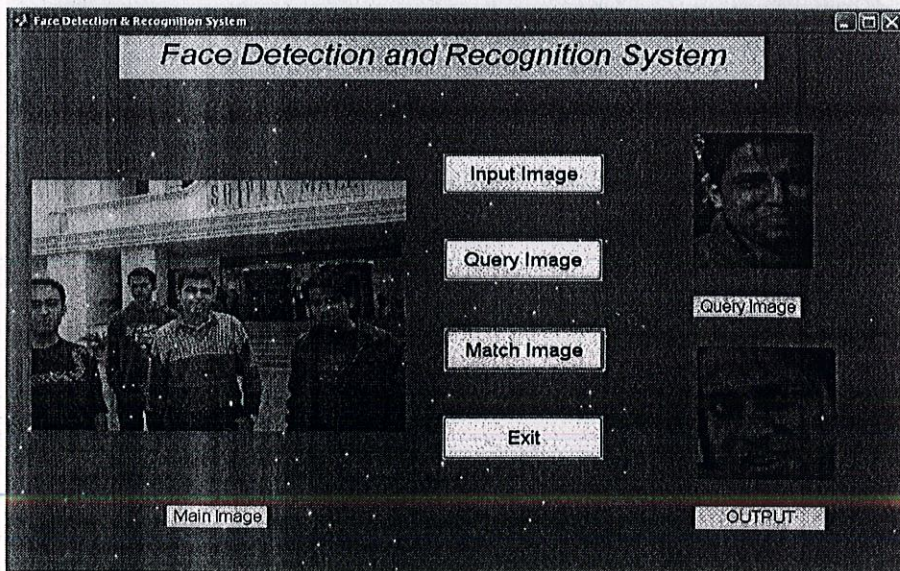


Figure 9.5.Observed Output2



### Output 3 (False Negative)

More results are obtained by checking with more images. The following result tells us that the person (face image) for which we are searching, is not present in the input image. (i.e. false negative result is verified which means the result holds good here too.)

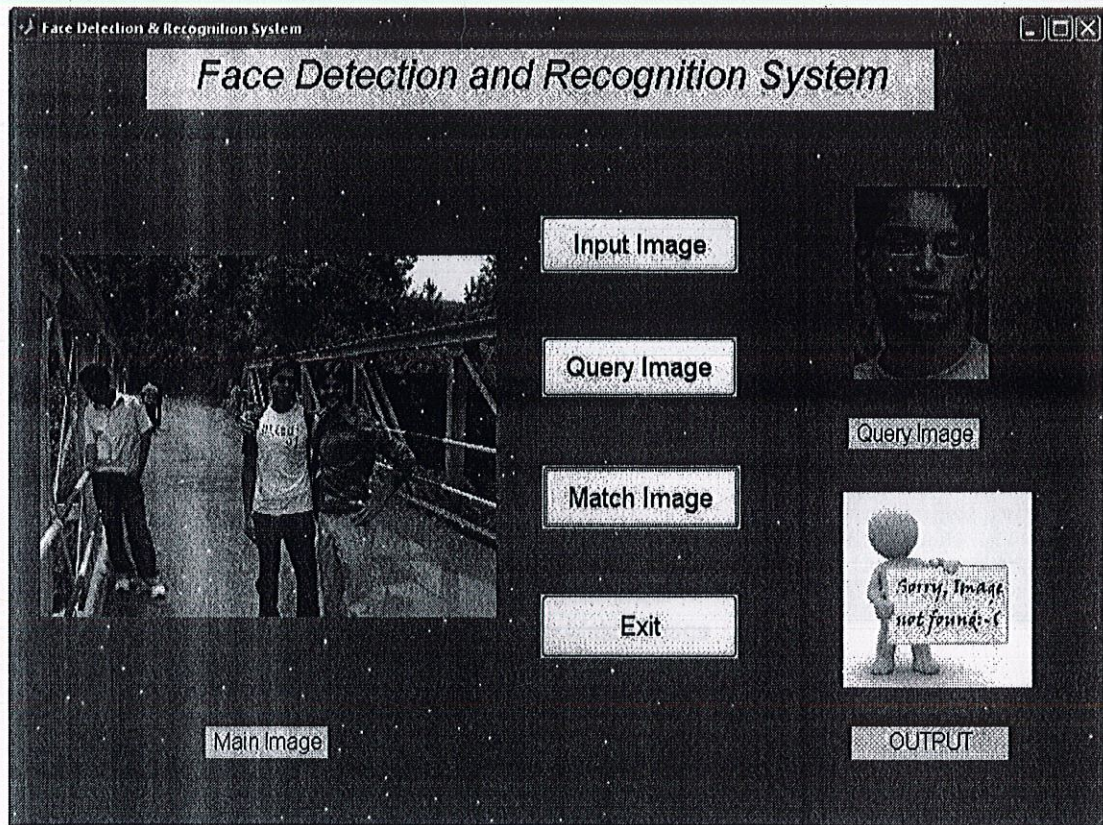
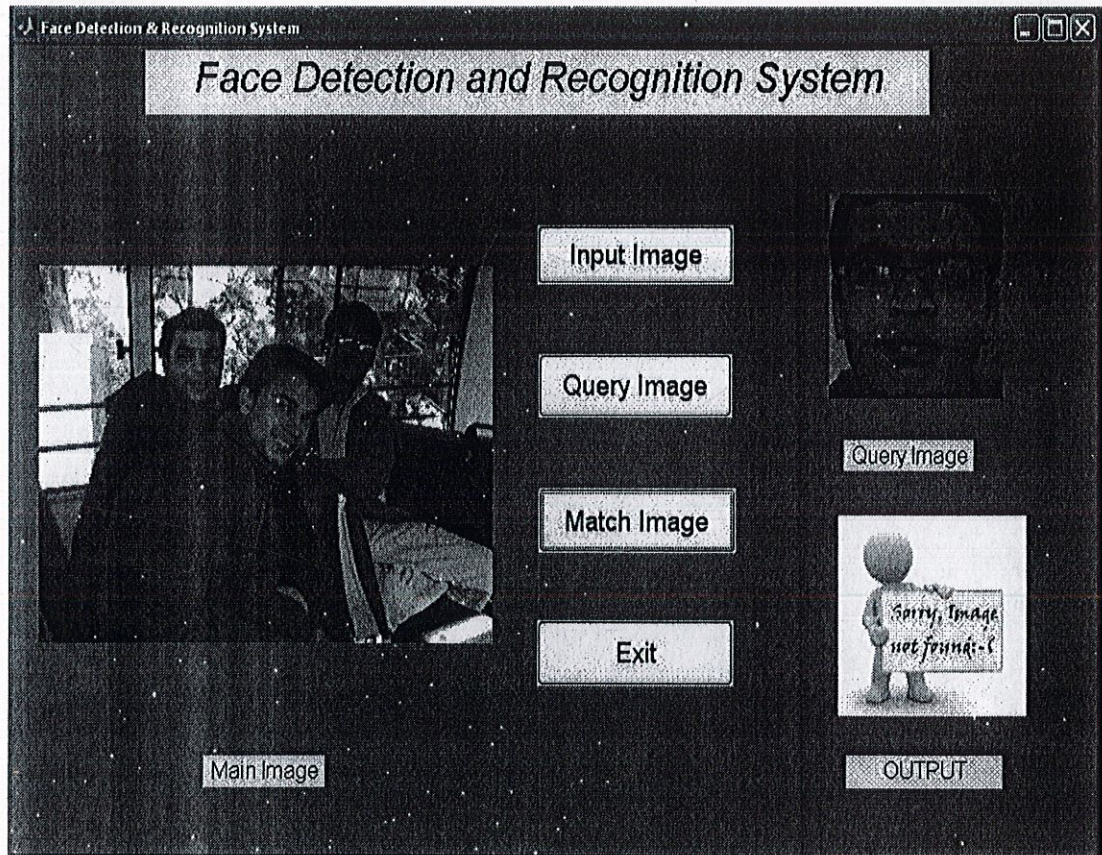


Figure 9.6.Observed Output3 (False Negative)



#### Output 4(True Negative)

Now, another sample image has been taken over here and it also undergoes the same process and the result is obtained. But the result obtained over here is true negative. This is the shortcoming of the system. The face image of the person in input image exists with respect to query image but the result obtained is negative. (i.e. it represents a true negative result.)

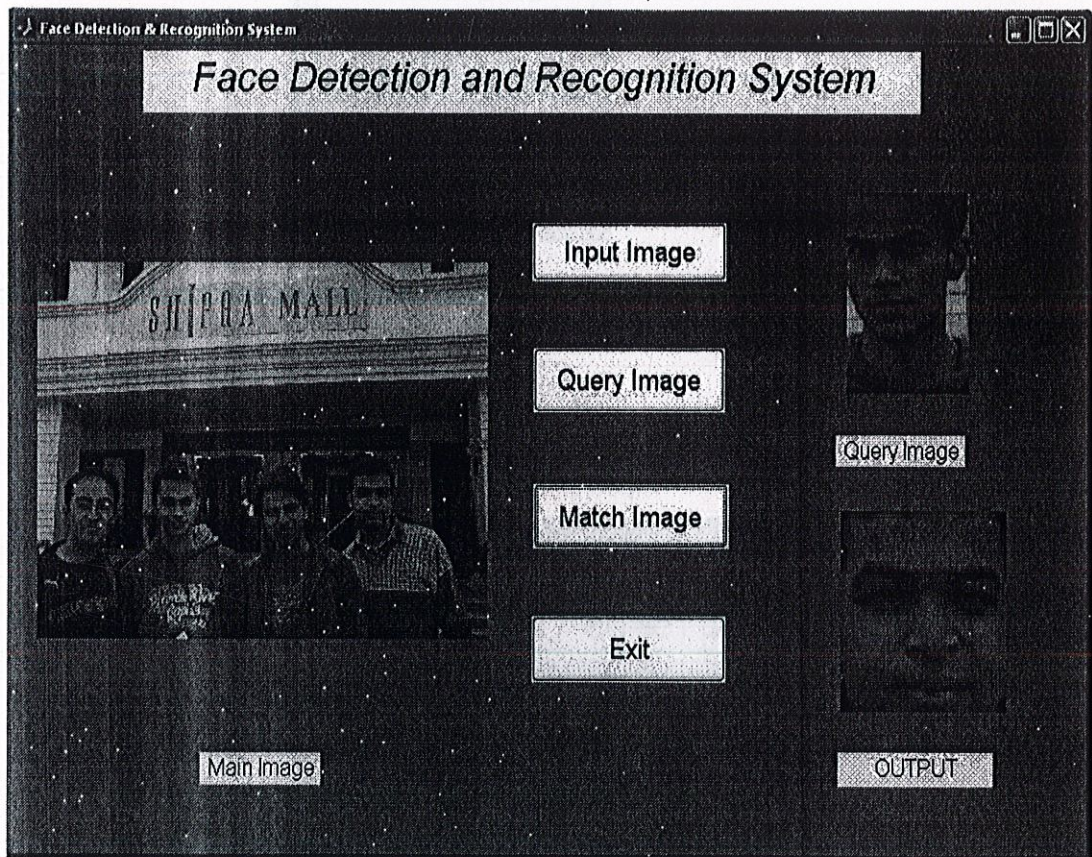


**Figure 9.7.**Observed Output4 (True Negative)



### Output 5 (False Positive)

This represents, the another shortcoming of the FDRS. Here the image with respect to the query image exists, but still system shows another image as a matched one. The matched image shown is also present in the input image but here we are not concerned about that image. (i.e. this snapshot represents a false positive result.)



**Figure 9.8.**Observed Output5 (False Positive)



## 9.2 Snapshot of Face Recognition Implementation

### 9.2.1 Interface



Figure 9.9.Face Recognition Interface

### 9.2.2 Input Image

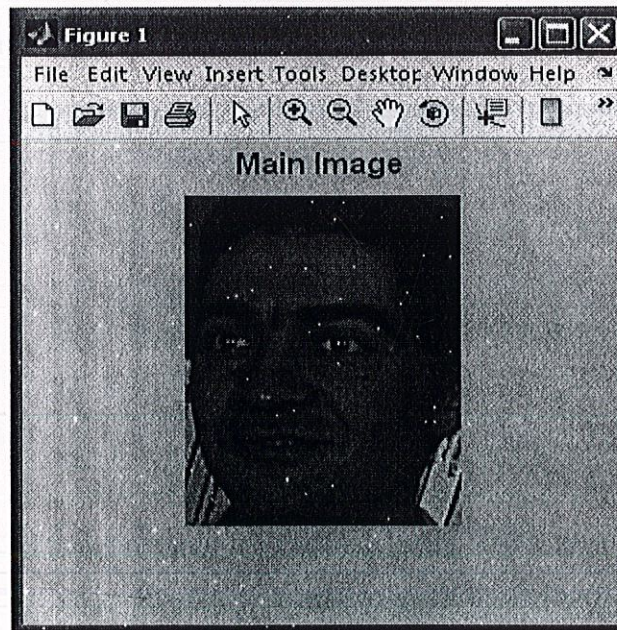


Figure 9.10.Face Recognition: Input image



### 9.2.3 Testing Images

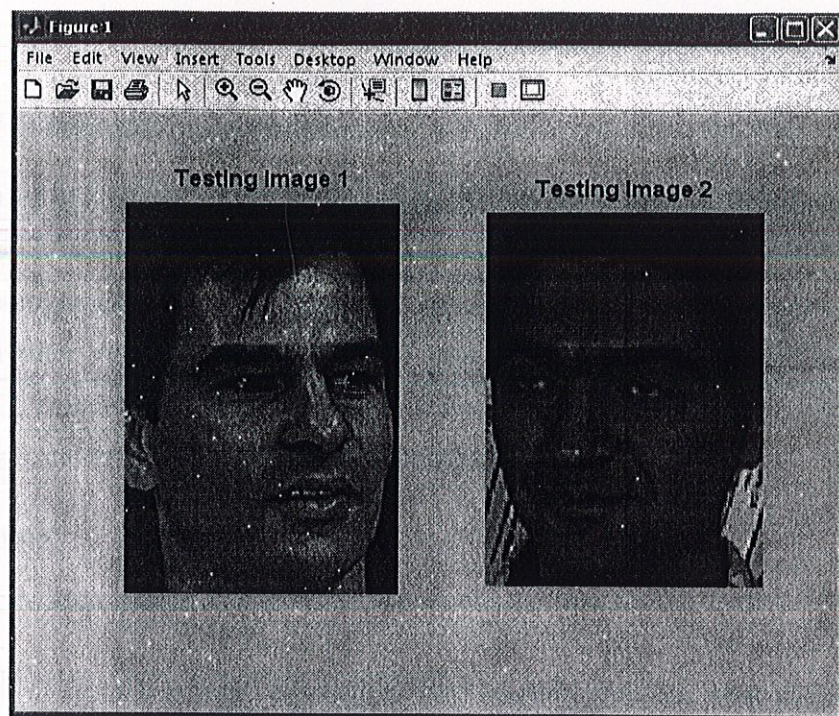


Figure 9.11. Testing images

### 9.2.4 Matched O/P

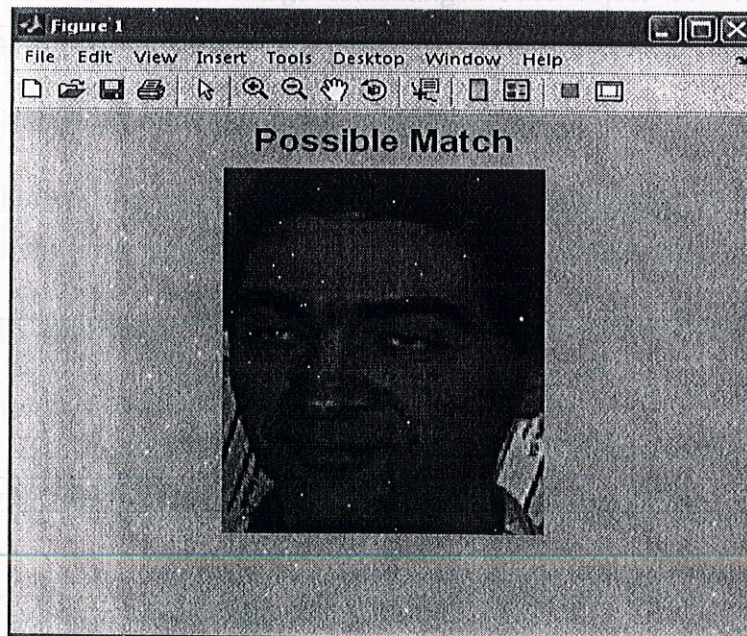


Figure 9.12. Matched o/p image



### TECHNOLOGY & APPLICATIONS

#### 10.1 About Technology

The system uses MATLAB 7.1 IPT 5.0.2(Image Processing Toolbox) toolbox to perform various Image Processing techniques on image. The Image Processing Toolbox (IPT) provides a comprehensive set of functions for image manipulation, analysis, digital imaging, computer vision, and digital image processing. The IPT capabilities include image file I/O, colour space transformations, linear filtering, mathematical morphology, texture analysis, pattern recognition, image statistics and others.

#### 10.2 Applications

##### 10.2.1 Image search engine

With the birth of Google it was very easy to find relevant data but what about images. What normally search engines provide is text based search, which helps users to find relevant textual data, but he is not able to do image based search. At max what can be done is to type in a text and find the images associated to it. Search can be based on Image. The input will be an image containing a Face/Object now this ROI can be extracted from the input image and can be used to search in the image Database.

##### 10.2.2 Face Detection and Recognition on facebook/orkut

The application will be very similar to the image search engines. This will enable the users to search the other uses on the social networking sites, using their images or snaps.

##### 10.2.3 Security application

Applications for surveillance for example Bank surveillance.

##### 10.2.4 Person Recognition

This application is very similar to face recognition, but in this application input need not to be the close-up snap of a person.



### 10.3 How to make it better?

Because of the disadvantage of PCA algorithm (i.e Lighting variant- if the light intensity changes, the face won't be recognized that accurate) there are some problems present related to the accuracy of the system. There are alternate technologies available to remove this flaw from the system. Instead of PCA algorithm we can use other Face Recognition algorithms which are more accurate and efficient. Other Face recognition algorithms like ICA (Independent Component Analysis) & LDA can be used.

### 10.4 Alternate Technology

Alternate technology available is Open CV .Open CV has itself played a role in the growth of Computer Vision by enabling people to do more productive works in vision. Face Detection in Open CV is based on Haar classifiers which builds a boosted rejection cascade.

Open CV implements a version of face detection technique developed by Paul Viola and Michael Jones ....commonly known as Viola-Jones Detector.



## CHAPTER XI

### FUTURE ASPECTS

#### 11.1 Future of Face Detection

##### 11.1.1 Explanation

The face detector above uses the responses to a series of simple filters to classify regions of an image as either a face or not a face. The filters are called Haar filters and are calculated by taking the sum of pixels within a number of rectangles, multiplying each sum by a weight and adding the results. Example filters are shown below:

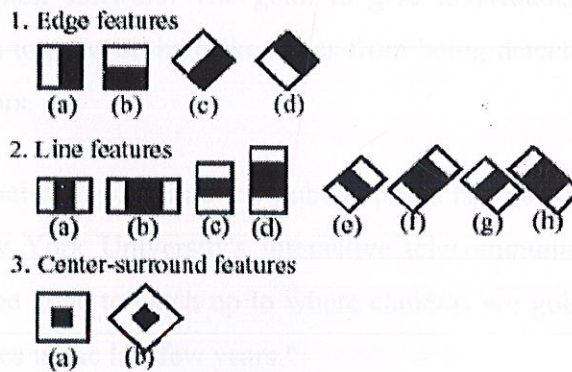


Figure 11.1. Image Geometric Features

After the application of the filter those image regions least likely to be a face are rejected, those that might represent a face are passed on to the next stage, as shown below:

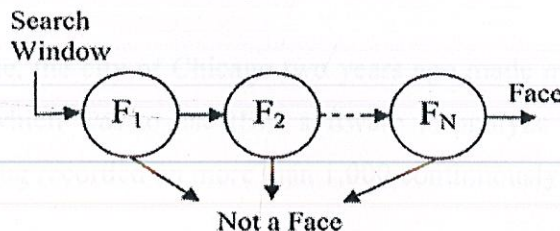


Figure 11.2. Window Search



You can display the search scan, testing each image region using the algorithm. Displaying the search is a lot slower, as you will see if you turn off the search animation. You can experiment with a search range with a large larger rectangle and search step size and then animate the result. At the end overlapping rectangles are combined into a single face box.

### **11.1.2 Reverse-engineering artist busts face detection tech**

Concerned about the proliferation of face recognition systems in public places, a grad student in New York is developing privacy-enhancing hacks designed to thwart the futuristic surveillance technology.

Using off-the-shelf makeup and accessories such as glasses, veils, and artificial hair, Adam Harvey's master's thesis combines hipster fashion aesthetics with hardcore reverse engineering of face detection software. The goal: to give individuals a low-cost and visually stimulating means to prevent their likenesses from being detected and cataloged by face-recognition monitors.

"The number of sensors that are going into the public spaces has been increasing," said Harvey, a student in New York University's interactive telecommunications program. "There's a lot of work to be done to catch up to where cameras are going because there have been so many advances in the last few years."

Although still in its adolescence, face recognition technology is quickly being adopted by governments and corporations to identify individuals whose images are captured by surveillance cameras. At the 2001 Super Bowl, for instance, officials digitized the faces of everyone entering Raymond James Stadium in Tampa, Florida and compared the results against photographic lists of known malefactors.

In another example, the city of Chicago two years ago made much fanfare of "Operation Virtual Shield," which was to use IBM software to analyze in real time thousands of hours of video being recorded on more than 1,000 continuously running cameras.

As a starting point, Harvey's research involves the reverse engineering of OpenCV, which



its creators describe as an open-source "library of programming functions for real-time computer vision." From that work, he developed an understanding of the algorithm used to tell if an image captured by a camera is, say, a car, a building or a human face.

Based on the so-called Viola-Jones method (pdf), the algorithm examines the spatial relationships of an object captured in an image and looks for features commonly found in faces. Most faces have a dark region just above the eyes, while the cheek bones and nose-bridge will appear lighter. When the algorithm detects enough such attributes, it guesses the object is a face. The method is generally regarded as effective. Errors are in favor of false positives, making it hard for unobstructed faces to escape notice when they aren't captured at an angle.

Once a face is detected, other technologies, such as face recognition, can be used to compare the face against a database in an attempt to identify the person it belongs to.

But Harvey has discovered that face detection can often be thrown off by using makeup to alter the contrasts the technology looks for. For example, dark patterns applied around eyes and cheek bones, as in the image below, are one such possibility.



**Figure 11.3.** Viola Jones Detection

These faces aren't detected by Viola-Jones algorithms

"There's a lot of trial and error," Harvey said. "The common thread is throwing off the symmetry" the algorithm looks for. "It's a lot more difficult than applying a bunch of makeup and hoping it works or putting on your 3D glasses left over from *Avatar*."



Technology that detects or recognizes faces still hasn't gone main stream, said Ralph Gross, an expert in the field and a scientist at Hyperactive Technologies. For now, it's mostly limited to specialized applications, such as the Super Bowl outings or Las Vegas casinos, but he said he expects that to change.

"The technology is getting better all the time," he said. "Along with computing power being greater, it becomes more of an option. I think we're heading to the point where we as a society need to think about what we are comfortable with."

At the same time, Gross said a pair of dark sunglasses or a simple veil would probably prove just as effective at thwarting face detection as anything Harvey is recommending.

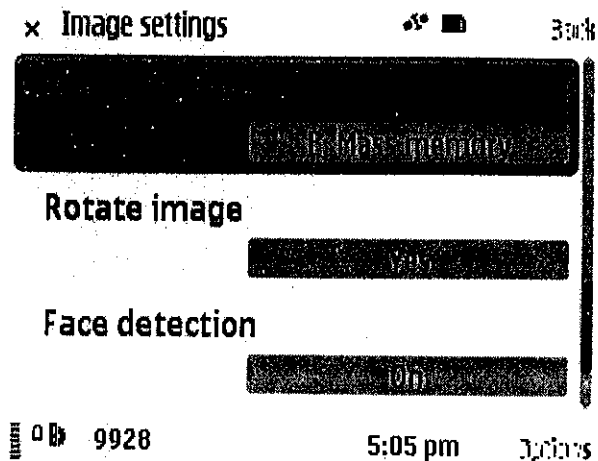
But, Harvey said the point of his project from the beginning has been to create disguises that do more than simply hide a person's face.

"The combination of hair, makeup and accessories gives you the potential to do an infinite number of creative new looks that have some futuristic value to them with anti-face-detection functionality," he said. "Maybe you could go to a privacy hair stylist in the future."

#### **11.1.3 Nokia N86 to have face detection in future firmware update**

AAS is reporting that Nokia's camera flagship is to have the face detection feature in its next firmware upgrade. This feature, if it is implemented, will enable the N86, as the name suggests, detect a face and increase focus over it. Face Detection has long been available to samsung devices and now for the first time on a Nokia device. Have a look at the screenshot below for further proof.





**Figure 11.4.N-86 Face Detection GUI**

The screenshot on the left is from a yet to be released firmware update.

Other notable changes are improvements in camera quality.

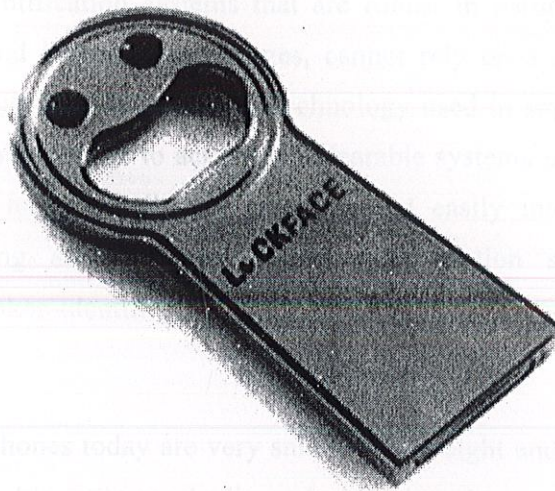
#### **11.1.4 Secure USB Drive relies on recognizing faces:**

These works as bottle opener too.

Portable data security has stepped up a notch following one manufacturer's decision to pair a USB Flash drive with facial recognition technology.

The first time when the Lockface USB drive is inserted into the PC, it will ask to "register" a face, so that in future the device can authenticate you (say your pearly white grin and baby blue eyes) with ease.





**Figure 11.5.** Lock Face Opener

Lock face opener will work on the basis of face recognition.

The software only requires to register one's face is contained on the 4GB drive itself, but its facial recognition feature only works with PCs connected to a webcam.

Plug the keyhole-shaped drive into a friend's camless machine and it will ask for a password before being granted access to Lockface's data files, according to a report by the Nikkei.

Should a would-be data thief fail miserably at passing your friend's face off as yours or at guessing your password, they could always break into your beer supply using Lockface's handy integrated bottle opener.

### **11.2 Future of Face Recognition Technology**

Face recognition systems used today work very well under constrained conditions, although all systems work much better with frontal mug-shot images and constant lighting. All current face recognition algorithms fail under the vastly varying conditions under which humans need to and are able to identify other people. Next generation person recognition systems will need to recognize people in real-time and in much less constrained situations.



We believe that identification systems that are robust in natural environments, in the presence of noise and illumination changes, cannot rely on a single modality, so that fusion with other modalities is essential. Technology used in smart environments has to be unobtrusive and allow users to act freely. Wearable systems in particular require their sensing technology to be small, low powered and easily integrable with the user's clothing. Considering all the requirements, identification systems that use face recognition and speaker identification seem to us to have the most potential for wide-spread application.

Cameras and microphones today are very small, light-weight and have been successfully integrated with wearable systems. Audio and video based recognition systems have the critical advantage that they use the modalities humans use for recognition. Finally, researchers are beginning to demonstrate that unobtrusive audio-and-video based person identification systems can achieve high recognition rates without requiring the user to be in highly controlled environments;

**11.2.1 Mobile authentication:** Application in mobile phone

**11.2.2 IR-based technology:** To achieve excellent accuracy

**11.2.3 3D face recognition:** Under research

After the improvement in face detection and face recognition technologies our system (FDRS) can be used very efficiently and effectively for the very high security purpose and can also be used on the social networking sites for the help of people. If anyone has its friend's image then he can easily find him/her by applying FDRS.



## CHAPTER XII

### CONCLUSION & RESULTS

In order to verify the correctness of the proposed model four different image sets were taken consisting of 4, 6, 8 and 10 faces respectively. The algorithm was run 10 times for each input image set and the query images were selected randomly. The result of matching has been given in table. It is evident from the table, that as the number of faces increases in the input image, the accuracy of the model goes down

Set	No. of Faces	Correctly Matched Faces			Accuracy (%)
		Face1	Face2	Face3	
I	4	4	4	3	91.67
II	6	5	4	6	83.33
III	8	7	6	6	79.17
IV	10	6	8	8	73.33

**Figure 12.1** Result of 10 runs on four image sets



## PROJECT CODING

## 13.1 GUI Code

```

function varargout = GUI2(varargin)
clc
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @GUI2_OpeningFcn, ...
                  'gui_OutputFcn',  @GUI2_OutputFcn, ...
                  'gui_LayoutFcn',   [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before GUI2 is made visible.
function GUI2_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to GUI2 (see VARARGIN)

% Choose default command line output for GUI2
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes GUI2 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = GUI2_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

```



```
% --- Executes on button press in pushbutton1.
function im1 = pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[file_name file_path] = uigetfile ('*.jpg');
im1 = imread ([file_path,file_name]);
axes(handles.axes1);
imshow(im1);
```

```
% --- Executes on button press in pushbutton2.
function im2 = pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[file_name file_path] = uigetfile ('*.jpg');
im2 = imread ([file_path,file_name]);
axes(handles.axes2);
imshow(im2);
```

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
im1 = pushbutton1_Callback(hObject, eventdata, handles);
im2 = pushbutton2_Callback(hObject, eventdata, handles);
axes(handles.axes3);
OI = matchface(im1,im2);
% imshow(OI);
```

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
clear all;
clc;
close all;
```

```
% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```



### 13.2 Matching Function Code

```
function OI = matchface(MI,QI)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Reshape the Query Image %%%%%%%%%%
    try
        QI = rgb2gray(QI);
    end
    QI = imresize(QI,[112 92]);
    r = reshape(QI,112*92,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Sending Main Image For Face Detection%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    I = double(MI);
    face = [];
    % taking the size and different component of rgb image
    H=size(I,1);
    W=size(I,2);
    R=I(:,:,1);
    G=I(:,:,2);
    B=I(:,:,3);

    % converting the rgb image into YCbCr color map image
    YCbCr=rgb2ycbcr(I);
    Y=YCbCr(:,:,1);

    % normalization of image
    minY=min(min(Y));
    maxY=max(max(Y));
    Y=255.0*(Y-minY)/(maxY-minY);
    YEye=Y;
    Yavg=sum(sum(Y))/(W*H);
    T=1;
    if (Yavg<64)
        T=1.4;
    elseif (Yavg>192)
        T=0.6;
    end

    if (T~=1)
        RI=R.^T;
        GI=G.^T;
    else
        RI=R;
        GI=G;
    end

    C=zeros(H,W,3);
    C(:,:,1)=RI;
    C(:,:,2)=GI;
    C(:,:,3)=B;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Skin Color Extration%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    YCbCr=rgb2ycbcr(C);
    Cr=YCbCr(:,:,3);
    S=zeros(H,W);
```



```

[SkinIndexRow, SkinIndexCol] = find(10 < Cr & Cr < 45);
for i=1:length(SkinIndexRow)
    S(SkinIndexRow(i), SkinIndexCol(i))=1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Skin Color Block Extraction %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

L = bwlabel(S,8); % give the matrix of same size with connected label
of 8

BB = regioncrops(L, 'BoundingBox'); %
bboxes= cat(1, BB.BoundingBox);
widths=bboxes(:,3);
wid = size(widths,1)
heights=bboxes(:,4);
hByW=heights./widths;
lenRegions = size(bboxes,1);
foundFaces = zeros(1,lenRegions);
rgb=label2rgb(L);
j=0;

% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Check Face Criteria %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i=1:lenRegions
    % 1st criteria: height to width ratio, computed above.
    if (hByW(i)>1.75 || hByW(i)<0.75)
        continue;
    end
    % implemented by me: Impose a min face dimension constrain
    if (heights(i)<50 && widths(i)<50)
        continue;
    end

    % get current region's bounding box

    CurBB=bboxes(i,:);
    XStart=CurBB(1);
    YStart=CurBB(2);
    WCur=CurBB(3);
    HCur=CurBB(4);

    % crop current region

    rangeY=int32(YStart):int32(YStart+HCur-1);
    rangeX= int32(XStart):int32(XStart+WCur-1);
    RIC=RI(rangeY, rangeX);
    GIC=GI(rangeY, rangeX);
    imo = GIC;
    imo = uint8(imo);
    if(size(GIC,1) >100 && size(GIC,2) >100 )
        if(size(GIC,1) <300)
            r1 = imresize(imo,[112 92]);
            j = j+1;
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Eye Detection Check %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    M=zeros(HCur, WCur);
    theta=acos( 0.5.*(2.*RIC-GIC-BC) ./ sqrt( (RIC-GIC).*(RIC-GIC) +
    (RIC- BC).*(GIC-BC) ) );

```



```

theta(isnan(theta))=0;
thetaMean=mean2(theta);

%%%%%%%%%%%% Mouth Detection Check %%%%%%%%%%%%%%
[MouthIndexRow,MouthIndexCol] =find(theta<thetaMean/4);
for j=1:length(MouthIndexRow)
    M(MouthIndexRow(j),MouthIndexCol(j))=1;
End
% now compute vertical mouth histogram
Hist=zeros(1, HCur);

for j=1:HCur
    Hist(j)=length(find(M(j,:)==1));
end

wMax=find(Hist==max(Hist));
wMax=wMax(1); % just take one of them.

if (wMax < WCur/6)
    %reject due to not existing mouth
    continue;
end

figure, imshow(M);
face(:,j) = reshape(r1,112*92,1);
imwrite(imo,['k',int2str(j),'.jpg']);
else
    disp('cant display');
end
end
v = face;

%%%%%%%%%%%% Face Matching %%%%%%%%%%%%%%
v = uint8(v);
x = size(v,2);
O=uint8(ones(1,size(v,2))); % size(O)
m=uint8(mean(v,2)); % m is the mean images.
size(uint8(single(m)*single(O)))
vzm=v-uint8(single(m)*single(O)); %difference of individual
images wd mean image
N=x; % define the vector for highest eigenvalue
L=single(vzm)'*single(vzm); %A'*A
[V,D]=eig(L); %eigen vector calculation
% k1 = size(V) % 4 4
V=single(vzm)*V;
size(V); %10304 2
V=V(:,end:-1:end-(N-1));
% k= size(V) % 10304 2
cv=zeros(size(v,2),N);
for i=1:size(v,2);
    cv(i,:)=single(vzm(:,i))'*V; % Each row in cv is the
signature for one image.
end
size(cv); % 2 2
% subplot(122);
p=r-m; %10304 1 % Subtract the mean
s=single(p)'*V; % sy = size(s) % 1 2

```



```

z=[]; % c3=size(v)
for i=1:x
    z=[z,norm(cv(i,:)-s,2)];

end
[a,i]= min(z)
[b,j]= max(z)
if(a== 2.1608e+006)
    i=3
end

if(b==5668314||b==6.1369e+006)
    j=2;
end
if(b==13824115 ||
b==12345485||b==5668314||b==9472335||b==6.1369e+006)
    OI = reshape(v(:,j),112,92);
    OI = imresize(OI,[60 55]);
    imshow(OI);
    return
end

if (2000 <a)
    if(a <= 2.60957e+006)
        OI = reshape(v(:,i),112,92);
        OI = imresize(OI,[60 55]);
        imshow(OI);
    else
        imshow(imread('notfound image.jpg'));
    end
else
    imshow(imread('notfound image.jpg'));
end
end

```

### 13.3 Regioncrops Function Code

```

function outstats = regionprops(varargin)
[L,I,requestedStats,officialStats] = ParseInputs(varargin{:});

if ndims(L) > 2
    % Remove stats that aren't supported for N-D input and issue
    % warning messages as appropriate.
    requestedStats = PreprocessRequestedStats(requestedStats);
end

if isempty(requestedStats)
    eid = sprintf('Images:%s:noPropertiesWereSelected',mfilename);
    error(eid, 'No properties were selected.');
```

```

end

if (isempty(L))
    numObjs = 0;
else
    numObjs = round(double(max(L(:)))));
end

```



```

% Initialize the stats structure array.
tempStats = {'PerimeterCornerPixelList'};
allStats = [officialStats; tempStats];
numStats = length(allStats);
empties = cell(numStats, numObjs);
stats = cell2struct(empties, allStats, 1);
% Initialize the statsAlreadyComputed structure array. Need to avoid
% multiple calculations of the same property for performance reasons.
zz = cell(numStats, 1);
for k = 1:numStats
    zz{k} = 0;
end
statsAlreadyComputed = cell2struct(zz, allStats, 1);

% Calculate PixelIdxList
[stats, statsAlreadyComputed] = ...
    ComputePixelIdxList(L, stats, statsAlreadyComputed, numObjs);

% Compute statistics.
numRequestedStats = length(requestedStats);
for k = 1 : numRequestedStats
    switch requestedStats{k}

        case 'Area'
            [stats, statsAlreadyComputed] = ...
                ComputeArea(stats, statsAlreadyComputed);

        case 'FilledImage'
            [stats, statsAlreadyComputed] = ...
                ComputeFilledImage(L, stats, statsAlreadyComputed);

        case 'FilledArea'
            [stats, statsAlreadyComputed] = ...
                ComputeFilledArea(L, stats, statsAlreadyComputed);

        case 'ConvexArea'
            [stats, statsAlreadyComputed] = ...
                ComputeConvexArea(L, stats, statsAlreadyComputed);

        case 'Centroid'
            [stats, statsAlreadyComputed] = ...
                ComputeCentroid(L, stats, statsAlreadyComputed);

        case 'EulerNumber'
            [stats, statsAlreadyComputed] = ...
                ComputeEulerNumber(L, stats, statsAlreadyComputed);

        case 'EquivDiameter'
            [stats, statsAlreadyComputed] = ...
                ComputeEquivDiameter(L, stats, statsAlreadyComputed);

        case 'Extrema'
            [stats, statsAlreadyComputed] = ...
                ComputeExtrema(L, stats, statsAlreadyComputed);

        case 'BoundingBox'
            [stats, statsAlreadyComputed] = ...

```



```

        ComputeBoundingBox(L, stats, statsAlreadyComputed);

    case 'SubarrayIdx'
        [stats, statsAlreadyComputed] = ...
            ComputeSubarrayIdx(L, stats, statsAlreadyComputed);

    case {'MajorAxisLength', 'MinorAxisLength', 'Orientation',
        'Eccentricity'}
        [stats, statsAlreadyComputed] = ...
            ComputeEllipseParams(L, stats, statsAlreadyComputed);

    case 'Solidity'
        [stats, statsAlreadyComputed] = ...
            ComputeSolidity(L, stats, statsAlreadyComputed);

    case 'Extent'
        [stats, statsAlreadyComputed] = ...
            ComputeExtent(L, stats, statsAlreadyComputed);

    case 'ConvexImage'
        [stats, statsAlreadyComputed] = ...
            ComputeConvexImage(L, stats, statsAlreadyComputed);

    case 'ConvexHull'
        [stats, statsAlreadyComputed] = ...
            ComputeConvexHull(L, stats, statsAlreadyComputed);

    case 'Image'
        [stats, statsAlreadyComputed] = ...
            ComputeImage(L, stats, statsAlreadyComputed);

    case 'PixelList'
        [stats, statsAlreadyComputed] = ...
            ComputePixelList(L, stats, statsAlreadyComputed);

    case 'Perimeter'
        [stats, statsAlreadyComputed] = ...
            ComputePerimeter(L, stats, statsAlreadyComputed);

    case 'PixelValues'
        [stats, statsAlreadyComputed] = ...
            ComputePixelValues(I, stats, statsAlreadyComputed);

    case 'WeightedCentroid'
        [stats, statsAlreadyComputed] = ...
            ComputeWeightedCentroid(L, I, stats, statsAlreadyComputed);

    case 'MeanIntensity'
        [stats, statsAlreadyComputed] = ...
            ComputeMeanIntensity(I, stats, statsAlreadyComputed);

    case 'MinIntensity'
        [stats, statsAlreadyComputed] = ...
            ComputeMinIntensity(I, stats, statsAlreadyComputed);

    case 'MaxIntensity'
        [stats, statsAlreadyComputed] = ...

```



```

        ComputeMaxIntensity(I,stats,statsAlreadyComputed);
    end
end

% Initialize the output stats structure array.
empties = cell(numRequestedStats, numObjs);
outstats = cell2struct(empties, requestedStats, 1); %#ok<NASGU>

fnames = fieldnames(stats);
deleteStats = fnames(~ismember(fnames,requestedStats));
outstats = rmfield(stats,deleteStats);

%%%
%%% ComputePixelIdxList
%%%
function [stats, statsAlreadyComputed] = ...
    ComputePixelIdxList(L, stats,statsAlreadyComputed,numobj)
%   A P-by-1 matrix, where P is the number of pixels belonging to
%   the region. Each element contains the linear index of the
%   corresponding pixel.

statsAlreadyComputed.PixelIdxList = 1;

if ~isempty(L) && numobj ~= 0
    idxList = regionpropsmex(L, numobj);
    [stats.PixelIdxList] = deal(idxList{:});
end

%%%
%%% ComputeArea
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeArea(stats, statsAlreadyComputed)
%   The area is defined to be the number of pixels belonging to
%   the region.

if ~statsAlreadyComputed.Area
    statsAlreadyComputed.Area = 1;

    for k = 1:length(stats)
        stats(k).Area = size(stats(k).PixelIdxList, 1);
    end
end

%%%
%%% ComputeEquivDiameter
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeEquivDiameter(L, stats, statsAlreadyComputed)
%   Computes the diameter of the circle that has the same area as
%   the region.
%   Ref: Russ, The Image Processing Handbook, 2nd ed, 1994, page
%   511.

if ~statsAlreadyComputed.EquivDiameter
    statsAlreadyComputed.EquivDiameter = 1;
end

```



```

    if ndims(L) > 2
        NoNDSupport('EquivDiameter');
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputeArea(stats, statsAlreadyComputed);

    factor = 2/sqrt(pi);
    for k = 1:length(stats)
        stats(k).EquivDiameter = factor * sqrt(stats(k).Area);
    end
end

%%%
%%% ComputeFilledImage
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeFilledImage(L, stats, statsAlreadyComputed)
% Uses imfill to fill holes in the region.

if ~statsAlreadyComputed.FilledImage
    statsAlreadyComputed.FilledImage = 1;

    [stats, statsAlreadyComputed] = ...
        ComputeImage(L, stats, statsAlreadyComputed);

    conn = conndef(ndims(L), 'minimal');

    for k = 1:length(stats)
        stats(k).FilledImage = imfill(stats(k).Image, conn, 'holes');
    end
end

%%%
%%% ComputeConvexArea
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeConvexArea(L, stats, statsAlreadyComputed)
% Computes the number of "on" pixels in ConvexImage.

if ~statsAlreadyComputed.ConvexArea
    statsAlreadyComputed.ConvexArea = 1;

    if ndims(L) > 2
        NoNDSupport('ConvexArea');
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputeConvexImage(L, stats, statsAlreadyComputed);

    for k = 1:length(stats)
        stats(k).ConvexArea = sum(stats(k).ConvexImage(:));
    end
end
end

```



```

%%
%% ComputeFilledArea
%%
function [stats, statsAlreadyComputed] = ...
    ComputeFilledArea(L,stats,statsAlreadyComputed)
% Computes the number of "on" pixels in FilledImage.

if ~statsAlreadyComputed.FilledArea
    statsAlreadyComputed.FilledArea = 1;

    [stats, statsAlreadyComputed] = ...
        ComputeFilledImage(L,stats,statsAlreadyComputed);

    for k = 1:length(stats)
        stats(k).FilledArea = sum(stats(k).FilledImage(:));
    end
end
S
%%
%% ComputeConvexImage
%%
function [stats, statsAlreadyComputed] = ...
    ComputeConvexImage(L,stats,statsAlreadyComputed)
% Uses ROIPLY to fill in the convex hull.

if ~statsAlreadyComputed.ConvexImage
    statsAlreadyComputed.ConvexImage = 1;

    if ndims(L) > 2
        NoNDSupport('ConvexImage');
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputeConvexHull(L,stats,statsAlreadyComputed);
    [stats, statsAlreadyComputed] = ...
        ComputeBoundingBox(L,stats,statsAlreadyComputed);

    for k = 1:length(stats)
        M = stats(k).BoundingBox(4);
        N = stats(k).BoundingBox(3);
        hull = stats(k).ConvexHull;
        if (isempty(hull))
            stats(k).ConvexImage = false(M,N);
        else
            firstRow = stats(k).BoundingBox(2) + 0.5;
            firstCol = stats(k).BoundingBox(1) + 0.5;
            r = hull(:,2) - firstRow + 1;
            c = hull(:,1) - firstCol + 1;
            stats(k).ConvexImage = roipoly(M, N, c, r);
        end
    end
end

%%
%% ComputeCentroid
%%
function [stats, statsAlreadyComputed] = ...

```



```

        ComputeCentroid(L, stats, statsAlreadyComputed)
    % [mean(r) mean(c)]

    if ~statsAlreadyComputed.Centroid
        statsAlreadyComputed.Centroid = 1;

        [stats, statsAlreadyComputed] = ...
            ComputePixelList(L, stats, statsAlreadyComputed);

        for k = 1:length(stats)
            stats(k).Centroid = mean(stats(k).PixelList,1);
        end

    end

end

%%%
%%% ComputeEulerNumber
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeEulerNumber(L, stats, statsAlreadyComputed)
% Calls BWEULER on 'Image' using 8-connectivity

if ~statsAlreadyComputed.EulerNumber
    statsAlreadyComputed.EulerNumber = 1;

    if ndims(L) > 2
        NoNDSupport('EulerNumber');
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputeImage(L, stats, statsAlreadyComputed);

    for k = 1:length(stats)
        stats(k).EulerNumber = bweuler(stats(k).Image, 8);
    end
end

end

%%%
%%% ComputeExtrema
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeExtrema(L, stats, statsAlreadyComputed)

% A 8-by-2 array; each row contains the x and y spatial

if ~statsAlreadyComputed.Extrema
    statsAlreadyComputed.Extrema = 1;

    if ndims(L) > 2
        NoNDSupport('Extrema');
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputePixelList(L, stats, statsAlreadyComputed);

```



```

for k = 1:length(stats)
    pixelList = stats(k).PixelList;
    if (isempty(pixelList))
        stats(k).Extrema = zeros(8,2) + 0.5;
    else
        r = pixelList(:,2);
        c = pixelList(:,1);

        minR = min(r);
        maxR = max(r);
        minC = min(c);
        maxC = max(c);

        minRSet = r == minR;
        maxRSet = r == maxR;
        minCSet = c == minC;
        maxCSet = c == maxC;

        % Points 1 and 2 are on the top row.
        r1 = minR;
        r2 = minR;
        % Find the minimum and maximum column coordinates for
        % top-row pixels.
        tmp = c(minRSet);
        c1 = min(tmp);
        c2 = max(tmp);

        % Points 3 and 4 are on the right column.
        % Find the minimum and maximum row coordinates for
        % right-column pixels.
        tmp = r(maxCSet);
        r3 = min(tmp);
        r4 = max(tmp);
        c3 = maxC;
        c4 = maxC;

        % Points 5 and 6 are on the bottom row.
        r5 = maxR;
        r6 = maxR;
        % Find the minimum and maximum column coordinates for
        % bottom-row pixels.
        tmp = c(maxRSet);
        c5 = max(tmp);
        c6 = min(tmp);

        % Points 7 and 8 are on the left column.
        % Find the minimum and maximum row coordinates for
        % left-column pixels.
        tmp = r(minCSet);
        r7 = max(tmp);
        r8 = min(tmp);
        c7 = minC;
        c8 = minC;

        stats(k).Extrema = [c1-0.5 r1-0.5
                           c2+0.5 r2-0.5
                           c3+0.5 r3-0.5
                           c4+0.5 r4+0.5
                           c5-0.5 r5+0.5
                           c6-0.5 r6+0.5
                           c7-0.5 r7+0.5
                           c8+0.5 r8+0.5];
    end
end

```



```

        c5+0.5 r5+0.5
        c6-0.5 r6+0.5
        c7-0.5 r7+0.5
        c8-0.5 r8-0.5];
    end
end

end

%%%
%%% ComputeBoundingBox
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeBoundingBox(L, stats, statsAlreadyComputed)
% [minC minR width height]; minC and minR end in .5.

if ~statsAlreadyComputed.BoundingBox
    statsAlreadyComputed.BoundingBox = 1;

    [stats, statsAlreadyComputed] = ...
        ComputePixelList(L, stats, statsAlreadyComputed);

    num_dims = ndims(L);

    for k = 1:length(stats)
        list = stats(k).PixelList;
        if (isempty(list))
            stats(k).BoundingBox = [0.5*ones(1,num_dims)
zeros(1,num_dims)];
        else
            min_corner = min(list,[],1) - 0.5;
            max_corner = max(list,[],1) + 0.5;
            stats(k).BoundingBox = [min_corner (max_corner -
min_corner)];
        end
    end
end

%%%
%%% ComputeSubarrayIdx
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeSubarrayIdx(L, stats, statsAlreadyComputed)
% Find a cell-array containing indices so that L(idx(:)) extracts the
% elements of L inside the bounding box.

if ~statsAlreadyComputed.SubarrayIdx
    statsAlreadyComputed.SubarrayIdx = 1;

    [stats, statsAlreadyComputed] = ...
        ComputeBoundingBox(L, stats, statsAlreadyComputed);
    num_dims = ndims(L);
    idx = cell(1,num_dims);
    for k = 1:length(stats)
        boundingBox = stats(k).BoundingBox;
        left = boundingBox(1:(end/2));
        right = boundingBox((1+end/2):end);
        left = left(1,[2 1 3:end]);

```



```

        right = right(1,[2 1 3:end]);
        for p = 1:num_dims
            first = left(p) + 0.5;
            last = first + right(p) - 1;
            idx{p} = first:last;
        end
        stats(k).SubarrayIdx = idx;
    end
end

%%%
%%% ComputeEllipseParams
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeEllipseParams(L,stats,statsAlreadyComputed)

region. Compute the axes lengths, orientation, and eccentricity of the
% ellipse. Ref: Haralick and Shapiro, Computer and Robot Vision vol
I,
% Addison-Wesley 1992, Appendix A.

if ~(statsAlreadyComputed.MajorAxisLength && ...
    statsAlreadyComputed.MinorAxisLength && ...
    statsAlreadyComputed.Orientation && ...
    statsAlreadyComputed.Eccentricity)
    statsAlreadyComputed.MajorAxisLength = 1;
    statsAlreadyComputed.MinorAxisLength = 1;
    statsAlreadyComputed.Eccentricity = 1;
    statsAlreadyComputed.Orientation = 1;

    if ndims(L) > 2
        NoNDSupport({'MajorAxisLength', 'MinorAxisLength', ...
            'Eccentricity', 'Orientation'});
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputePixelList(L,stats,statsAlreadyComputed);
    [stats, statsAlreadyComputed] = ...
        ComputeCentroid(L,stats,statsAlreadyComputed);

    for k = 1:length(stats)
        list = stats(k).PixelList;
        if (isempty(list))
            stats(k).MajorAxisLength = 0;
            stats(k).MinorAxisLength = 0;
            stats(k).Eccentricity = 0;
            stats(k).Orientation = 0;
        else
            % Assign X and Y variables so that we're measuring
orientation
            % counterclockwise from the horizontal axis.

            xbar = stats(k).Centroid(1);
            ybar = stats(k).Centroid(2);

```



```

    x = list(:,1) - xbar;
    y = -(list(:,2) - ybar); % This is negative for the
    % orientation calculation (measured in the
    % counter-clockwise direction).

    N = length(x);

    % Calculate normalized second central moments for the
    region. 1/12 is
    % the normalized second central moment of a pixel with unit
    length.
    uxx = sum(x.^2)/N + 1/12;
    uyy = sum(y.^2)/N + 1/12;
    uxy = sum(x.*y)/N;

    % Calculate major axis length, minor axis length, and
    eccentricity.
    common = sqrt((uxx - uyy)^2 + 4*uxy^2);
    stats(k).MajorAxisLength = 2*sqrt(2)*sqrt(uxx + uyy +
    common);
    stats(k).MinorAxisLength = 2*sqrt(2)*sqrt(uxx + uyy -
    common);
    stats(k).Eccentricity =
    2*sqrt((stats(k).MajorAxisLength/2)^2 - ...
    (stats(k).MinorAxisLength/2)^2) / ...
    stats(k).MajorAxisLength;

    % Calculate orientation.
    if (uyy > uxx)
        num = uyy - uxx + sqrt((uyy - uxx)^2 + 4*uxy^2);
        den = 2*uxy;
    else
        num = 2*uxy;
        den = uxx - uyy + sqrt((uxx - uyy)^2 + 4*uxy^2);
    end
    if (num == 0) && (den == 0)
        stats(k).Orientation = 0;
    else
        stats(k).Orientation = (180/pi) * atan(num/den);
    end
end
end

end

%%%
%%% ComputeSolidity
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeSolidity(L,stats,statsAlreadyComputed)
% Area / ConvexArea

if ~statsAlreadyComputed.Solidity
    statsAlreadyComputed.Solidity = 1;

    if ndims(L) > 2
        NoNDSupport('Solidity');
        return
    end
end

```



```

end

[stats, statsAlreadyComputed] = ...
    ComputeArea(stats, statsAlreadyComputed);
[stats, statsAlreadyComputed] = ...
    ComputeConvexArea(L, stats, statsAlreadyComputed);

for k = 1:length(stats)
    if (stats(k).ConvexArea == 0)
        stats(k).Solidity = NaN;
    else
        stats(k).Solidity = stats(k).Area / stats(k).ConvexArea;
    end
end
end

end

%%%
%%% ComputeExtent
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeExtent(L, stats, statsAlreadyComputed)
% Area / (BoundingBox(3) * BoundingBox(4))

if ~statsAlreadyComputed.Extent
    statsAlreadyComputed.Extent = 1;

    if ndims(L) > 2
        NoNDSupport('Extent');
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputeArea(stats, statsAlreadyComputed);
    [stats, statsAlreadyComputed] = ...
        ComputeBoundingBox(L, stats, statsAlreadyComputed);

    for k = 1:length(stats)
        if (stats(k).Area == 0)
            stats(k).Extent = NaN;
        else
            stats(k).Extent = stats(k).Area /
prod(stats(k).BoundingBox(3:4));
        end
    end
end

end

%%%
%%% ComputeImage
%%%
function [stats, statsAlreadyComputed] = ...
    ComputeImage(L, stats, statsAlreadyComputed)
% Binary image containing "on" pixels corresponding to pixels
% belonging to the region. The size of the image corresponds
% to the size of the bounding box for each region.

if ~statsAlreadyComputed.Image
    statsAlreadyComputed.Image = 1;

```



```

[stats, statsAlreadyComputed] = ...
    ComputeSubarrayIdx(L, stats, statsAlreadyComputed);

for k = 1:length(stats)
    subarray = L(stats(k).SubarrayIdx{:});
    if ~isempty(subarray)
        stats(k).Image = (subarray == k);
    else
        stats(k).Image = logical(subarray);
    end
end
end

function [stats, statsAlreadyComputed] = ...
    ComputePixelList(L, stats, statsAlreadyComputed)

if ~statsAlreadyComputed.PixelList
    statsAlreadyComputed.PixelList = 1;

    % Convert the linear indices to subscripts and store

    In = cell(1, ndims(L));
    for k = 1:length(stats)
        if ~isempty(stats(k).PixelIdxList)
            [In{:}] = ind2sub(size(L), stats(k).PixelIdxList);
            stats(k).PixelList = [In{:}];
            stats(k).PixelList = stats(k).PixelList(:, [2 1 3:end]);
        else
            stats(k).PixelList = zeros(0, ndims(L));
        end
    end
end

%%% ComputePerimeterCornerPixelList

function [stats, statsAlreadyComputed] = ...
    ComputePerimeterCornerPixelList(L, stats, statsAlreadyComputed)
% Find the pixels on the perimeter of the region; make a list
% of the coordinates of their corners; sort and remove
% duplicates.

if ~statsAlreadyComputed.PerimeterCornerPixelList
    statsAlreadyComputed.PerimeterCornerPixelList = 1;

    if ndims(L) > 2
        NoNDSupport('PerimeterCornerPixelList');
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputeImage(L, stats, statsAlreadyComputed);
    [stats, statsAlreadyComputed] = ...
        ComputeBoundingBox(L, stats, statsAlreadyComputed);

    for k = 1:length(stats)
        perimImage = bwmorph(stats(k).Image, 'perim8');
        firstRow = stats(k).BoundingBox(2) + 0.5;

```



```

        firstCol = stats(k).BoundingBox(1) + 0.5;
        [r,c] = find(perimImage);
        % Force rectangular empties.
        r = r(:) + firstRow - 1;
        c = c(:) + firstCol - 1;
        rr = [r-.5 ; r      ; r+.5 ; r      ];
        cc = [c      ; c+.5 ; c      ; c-.5];
        stats(k).PerimeterCornerPixelList = [cc rr];
    end

end

%%% ComputeConvexHull
function [stats, statsAlreadyComputed] = ...
    ComputeConvexHull(L,stats,statsAlreadyComputed)
%   A P-by-2 array representing the convex hull of the region.
%   The first column contains row coordinates; the second column
%   contains column coordinates. The resulting polygon goes
%   through pixel corners, not pixel centers.

if ~statsAlreadyComputed.ConvexHull
    statsAlreadyComputed.ConvexHull = 1;

    if ndims(L) > 2
        NoNDSupport('ConvexHull');
        return
    end

    [stats, statsAlreadyComputed] = ...
        ComputePerimeterCornerPixelList(L,stats,statsAlreadyComputed);
    [stats, statsAlreadyComputed] = ...
        ComputeBoundingBox(L,stats,statsAlreadyComputed);

    for k = 1:length(stats)
        list = stats(k).PerimeterCornerPixelList;
        if (isempty(list))
            stats(k).ConvexHull = zeros(0,2);
        else
            rr = list(:,2);
            cc = list(:,1);
            hullIdx = convhull(rr, cc);
            stats(k).ConvexHull = list(hullIdx,:);
        end
    end
end

end

%%%
%%% ComputePerimeter
%%%
function [stats, statsAlreadyComputed] = ...
    ComputePerimeter(L, stats, statsAlreadyComputed)

if ~statsAlreadyComputed.Perimeter
    statsAlreadyComputed.Perimeter = 1;

    if ndims(L) > 2
        NoNDSupport('ComputePerimeter');
        return
    end
end

```



```

end

B = regionboundariesmex(double(L),8);

for i = 1:length(B)
    boundary = B{i};
    delta = diff(boundary).^2;
    stats(i).Perimeter = sum(sqrt(sum(delta,2)));
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [stats, statsAlreadyComputed] = ...
    ComputePixelValues(I,stats,statsAlreadyComputed)

if ~statsAlreadyComputed.PixelValues
    statsAlreadyComputed.PixelValues = 1;

    for k = 1:length(stats)
        stats(k).PixelValues = I(stats(k).PixelIdxList);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [stats, statsAlreadyComputed] = ...
    ComputeWeightedCentroid(L,I,stats,statsAlreadyComputed)

if ~statsAlreadyComputed.WeightedCentroid
    statsAlreadyComputed.WeightedCentroid = 1;

    [stats, statsAlreadyComputed] = ...
        ComputePixelList(L,stats,statsAlreadyComputed);

    for k = 1:length(stats)
        sumIntensity = sum(I(stats(k).PixelIdxList));
        numDims = size(stats(k).PixelList,2);
        wc = zeros(1,numDims);
        for n = 1 : numDims
            M = sum(stats(k).PixelList(:,n) .* ...
                double(I(stats(k).PixelIdxList)));
            wc(n) = M / sumIntensity;
        end
        stats(k).WeightedCentroid = wc;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [stats, statsAlreadyComputed] = ...
    ComputeMeanIntensity(I,stats,statsAlreadyComputed)

if ~statsAlreadyComputed.MeanIntensity
    statsAlreadyComputed.MeanIntensity = 1;

    [stats, statsAlreadyComputed] = ...

```



```

        ComputePixelValues(I, stats, statsAlreadyComputed);

    for k = 1:length(stats)
        stats(k).MeanIntensity = mean(stats(k).PixelValues);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [stats, statsAlreadyComputed] = ...
    ComputeMinIntensity(I, stats, statsAlreadyComputed)

if ~statsAlreadyComputed.MinIntensity
    statsAlreadyComputed.MinIntensity = 1;

    [stats, statsAlreadyComputed] = ...
        ComputePixelValues(I, stats, statsAlreadyComputed);

    for k = 1:length(stats)
        stats(k).MinIntensity = min(stats(k).PixelValues);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [stats, statsAlreadyComputed] = ...
    ComputeMaxIntensity(I, stats, statsAlreadyComputed)

if ~statsAlreadyComputed.MaxIntensity
    statsAlreadyComputed.MaxIntensity = 1;

    [stats, statsAlreadyComputed] = ...
        ComputePixelValues(I, stats, statsAlreadyComputed);

    for k = 1:length(stats)
        stats(k).MaxIntensity = max(stats(k).PixelValues);
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [L I reqStats officialStats] = ParseInputs(varargin)

shapeStats = {
    'Area'
    'Centroid'
    'BoundingBox'
    'SubarrayIdx'
    'MajorAxisLength'
    'MinorAxisLength'
};

pixelValueStats = {
    'PixelValues'
    'WeightedCentroid'
    'MeanIntensity'
    'MinIntensity'
};

```



```

        'MaxIntensity');

iptchecknargin(1, inf, nargin, mfilename);

L = varargin{1};
if islogical(L)
    eid = 'Images:regionprops:binaryInput';
    error(eid, ...
        'Use bwlabel(BW) or double(BW) convert binary image to \n%s',
        ...
        'a label matrix before calling regionprops.');
```

```

end
iptcheckinput(L, {'numeric'}, ...
    {'real', 'integer', 'nonnegative', 'nonsparse'}, ...
    mfilename, 'L', 1);

basicStats = {'Area'
    'Centroid'
    'BoundingBox'};

I = [];
officialStats = shapeStats;
if nargin == 1
    %REGIONPROPS(L)
    reqStats = basicStats;
    return;
elseif isnumeric(varargin{2}) || islogical(varargin{2})
    %REGIONPROPS(L,I) or REGIONPROPS(L,I,PROPERTIES)
    I = varargin{2};
    iptcheckinput(I, {'numeric', 'logical'}, {}, mfilename, 'I', 2);
    assert(isequal(size(L), size(I)), ...
        'Images:regionprops:sizeMismatch', ...
        'I must have the same size as L.');
```

```

    officialStats = [shapeStats; pixelValueStats];
    if nargin == 2
        %REGIONPROPS(L,I)
        reqStats = basicStats;
        return;
    else
        %REGIONPROPS(L,I,PROPERTIES)
        startIdxForProp = 3;
        reqStats = getPropsFromInput(startIdxForProp, ...
            officialStats, pixelValueStats, basicStats, varargin{:});
    end
else
    %REGIONPROPS(L,PROPERTIES)
    startIdxForProp = 2;
    reqStats = getPropsFromInput(startIdxForProp, ...
        officialStats, pixelValueStats, basicStats, varargin{:});
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
function [reqStats, officialStats] = getPropsFromInput(startIdx,
    officialStats, ...
        pixelValueStats, basicStats, varargin)
```



```

if iscell(varargin{startIdx})
    %REGIONPROPS(...,PROPERTIES)
    propList = varargin{startIdx};
elseif strcmpi(varargin{startIdx}, 'all')
    %REGIONPROPS(...,'all')
    reqStats = officialStats;
    return;
elseif strcmpi(varargin{startIdx}, 'basic')
    %REGIONPROPS(...,'basic')
    reqStats = basicStats;
    return;
else
    %REGIONPROPS(...,PROP1,PROP2,...)
    propList = varargin{startIdx:end};
end

numProps = length(propList);
reqStats = cell(1, numProps);
for k = 1 : numProps
    if ischar(propList{k})
        noGrayscaleImageAsInput = startIdx == 2;
        if noGrayscaleImageAsInput
            tempStats = [officialStats; pixelValueStats];
            prop = iptcheckstrs(propList{k}, tempStats, mfilename, ...
                'PROPERTIES', k);
            if any(strcmp(prop,pixelValueStats))
                eid = sprintf('Images:%s:needsGrayscaleImage',
mfilename);
                error(eid, 'REGIONPROPS needs I as an %s ''%s''.', ...
                    'input to calculate', prop);
            end
        else
            prop = iptcheckstrs(propList{k}, officialStats, mfilename,
...
                'PROPERTIES', k);
            end
            reqStats{k} = prop;
        else
            eid = sprintf('Images:%s:invalidType', mfilename);
            error(eid, 'PROPERTY must be a string.');
```

```

end

function NoNDSupport(str)
wid = sprintf('Images:%s:measurementNotForN-D',mfilename);

if iscell(str)
    warn_str = sprintf('%s: %s ', ...
        'These measurements are not supported if ndims(L) > 2.', ...
        sprintf('%s ', str{:}));
else
    warn_str = sprintf('%s: %s', ...
        'This measurement is not supported if ndims(L) > 2.', ...
        str);
end
```



```
warning(wid, '%s', warn_str);
```

```
%%%
```

```
%% PreprocessRequestedStats
```

```
%%%
```

```
function requestedStats = PreprocessRequestedStats(requestedStats)
% Remove any requested stats that are not supported for N-D input
% and issue an appropriate warning.
```

```
no_nd_measurements = {'MajorAxisLength'
    'MinorAxisLength'
    'Eccentricity'
    'Orientation'
    'ConvexHull'
    'ConvexImage'
    'ConvexArea'
    'EulerNumber'
    'Extrema'
    'Extent'
    'Perimeter'};
```

```
bad_stats = find(ismember(requestedStats, no_nd_measurements));
```

```
if ~isempty(bad_stats)
```

```
    NoNDSupport(requestedStats(bad_stats));
```

```
end
```

```
requestedStats(bad_stats) = [];
```



## BIBLIOGRAPHY

- [1] Guangzheng Yang, Thomas S Huang, Human face detection in a complex background, *Pattern Recognition*, Volume 27, Issue 1, Pages 53-63, January 1994.
- [2] M. Turk and A. Pentland, "Eigenfaces for Recognition", *Journal of Cognitive Neuroscience*, vol.3, no. 1, pp. 71-86, 1991
- [3] M. Turk and A. Pentland, "Face recognition using eigenfaces", *Proc. IEEE conference on Computer Vision and Pattern Recognition*, pp. 586-591, 1991.
- [4] Juwei Lu, Kostantinos N. Plataniotis, and Anastasios N. Venetsanopoulos, "Face Recognition Using LDA", *IEEE Transactions on Neural Networks*, Vol. 14, No. 1, Jan 2003.
- [5] O. Deniz, M. Castrillon, M. Hernandez, "Face recognition using independent component analysis and support vector machines", *Pattern Recognition Letters, Audio- and Video-based Biometric Person Authentication (AVBPA 2001)*, Volume 24, Issue 13, Pages 2153-2157, September 2003.
- [6] Rabab M. Ramadan and Rehab F. Abdel - Kader, "Face Recognition using Particle Swarm Optimization-Based Selected Features" *International Journal of Signal Processing, Image Processing and Pattern Recognition* Vol. 2, No. 2, June 2009.
- [7] Paul Viola, Michael J. Jones, "Robust Real-Time Face Detection", *International Journal of Computer Vision*, pages 137-154, July 2003.
- [8] Wei, G. and Sethi, I. K. "Omni-face detection for video/image content description", *ACM Workshops on Multimedia*, pp. 185-189, October 2000.
- [9] Q. Yuan, W. Gao, and H. Yao, "Robust frontal face detection in complex environment," in *16th International Conference on Pattern Recognition, 2002. Proceedings*, Aug. 2002, vol. 1, pp. 25-28.
- [9] R.-L. Hsu, M. Abdel-Mottaleb, and A. K. Jain, "Face detection in color images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 696-706, May 2002.
- [10] J. Kovac, P. Peer, and F. Solina, "Illumination independent color-based face detection," in *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis*, Sept. 2003, vol. 1, pp. 510-515.



- [11] P. Belhumeur, J. Hespanha, and D. Kriegman (july 1997). "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection". IEEE Transactions on pattern analysis and machine intelligence
- [12] Eigenfaces for Face Detection/Recognition ,M. Turk and A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol.3, no. 1, pp. 71-86, 1991.
- [13] Rafael C. Gonzalez and Richard E. Woods,Digital Image Processing Third edition .
- [14] Rafael C. Gonzalez and Richard E. Woods,Digital Image Processing using Matlab.
- [15] <http://www.mathworks.com>
- [16] <http://en.wikipedia.org>
- [17] <http://vismod.media.mit.edu>
- [18] [www.ieeexplore.com](http://www.ieeexplore.com)
- [19] <http://www.theregister.co.uk>
- [20]<http://www.fonearena.com>
- [21] <http://www.facedetection.com>
- [22] [www.face-rec.org](http://www.face-rec.org)