ABSTRACT

COMPUTER AND INFORMATION SCIENCE DEPARTMENT

ARAFA, AMR B.S. ARAB ACADEMY FOR SCIENCE AND TECHNOLOGY, 2003

FACE RECOGNITION/CLUSTERING, PERFORMANCE IMPROVEMENTS

Advisor: Professor Roy George

Thesis dated May 2007

This thesis will introduce Face Recognition as an important and crucially needed type of biometrics. The existing and most widely used Face Recognition algorithms have been tested and the results will be presented. Additionally, the limitations of the existing FF methods will be discussed, focusing chiefly on the future of Face Recognition and the reasons such relatively poor results were achieved in comparison with results from other Biometrics. Finally, a novel system that enhances the performance of the face matching for existing FF algorithms (High-speed k-means Image Clustering using the Discrete Cosine Transform and its comparison with existing methods) will be discussed. Appendix A focuses on the results obtained from the Haar face detection algorithm, utilizing different face databases. Appendix B is dedicated to the Matlab code.

FACE RECOGNITION/CLUSTERING - PERFORMANCE IMPROVEMENTS

A THESIS

SUBMITTED TO THE FACULTY OF CLARK ATLANTA UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER'S OF COMPUTER SCIENCE

BY

AMR ARAFA

DEPARTMENT OF COMPUTER SCIENCE

ATLANTA, GEORGIA

MAY 2007

ACKNOWLEDGEMENTS

First and foremost, I thank my parents for their continuous love and support. I hope I have made them proud. I thank my supervisor, Dr. Roy George for his support through out my master's period and for the tolerance, patience and trust he has always given me. I specially thank him for allowing me to work in this exciting and very interesting area of pattern recognition and image analysis.

I thank Dr. Samir Moghazy from the Physics Department for encouraging me to join the Computer Science Master's program and for his truly caring parental support, which I shall never forget. Dr. Samir, this thesis is dedicated to you. I also thank Dr. Khalil Shujaee for his support and guidance throughout my period of study at Clark Atlanta University. I thank Clark Atlanta University's Computer and Information Science professors Dr. Hsin-Chu Chen, Dr. Gerry Howe, and Dr. Zhichenq Wang who have been very helpful and inspiring to me on both personal and academic levels. I thank the Clark Atlanta University staff for always assuring that everything ran smoothly, and for their willingness to help me whenever I needed them.

TABLE OF CONTENTS

TABLE OF CONTENTS

(continued)

Chapter Page

TABLE OF CONTENTS

(continued)

TABLE OF CONTENTS

(continued)

Chapter Page

LIST OF FIGURES

LIST OF FIGURES

(continued)

LIST OF TABLES

Table Page

CHAPTER 1

INTRODUCTION

1.0 Introduction

The human ability of recognizing faces is quite impressive. Although face recognition studies date back twenty years, a machine that could imitate the human brain's ability to recognize faces under different natural challenges like pose, illumination and aging, did not exist yet. In the past decade, biometrics has shown a great deal of success in many applications; however, the need of a contact free authentication system has become much more of a necessity, while at the same time, it provides one of the best advantages offered by face recognition. The amount of time spent on matching the entered face with the entire database is both processor intense and time wasting. To address this issue, in this thesis, a clustering system that works as a pre-process to face recognition was developed. The idea is to cluster faces that look similar together so that an entered face would be compared to the most similar cluster first. A comparison was also made of some important face recognition algorithms over different databases. In Appendix A the Haar face detection algorithm was also tested over different databases.

1.1 Overview

In this project, a method for automated classification and recognition of face images from a random set of images is presented using *k-means clustering* and compared

with general clustering methods in face clustering. The method is speeded up considerably, using the *Discrete Cosine Transform* (DCT) for dimensionality reduction of image data. The method results in the obtainment of one or more *central* images, each representing a different person's face found in the face database. Recognition can then be implemented by the nearest neighbor technique, or any other suitable technique.

The service includes entering the database location in the memory into the interface and automatically grouping them into a number of clusters entered by the user, and displaying the central images for each cluster. The next phase of the project is to allow the user to save the new clusters to allow the third phase, the recognition. The recognition phase involves matching an entered face with the central images created earlier by the system. All this is done in a step-by-step fashion and a user-friendly interface to widen the range of possible users to the system.

1.2 Project Objectives

The primary goal of this thesis is to design and implement a system that is capable of successful face clustering for different face databases. The main objectives of the face Recognition System is:

- To provide a user-friendly interface.
- To reduce the consumption of the storage devices.
- To speed up the clustering process by applying DCT (Discrete Cosine Transform) technique

1.3 General Description

Face database clustering is a system that clusters faces with similar features in a one database to ease the subsequent task of recognition. The following diagram shows the system's flow chart.

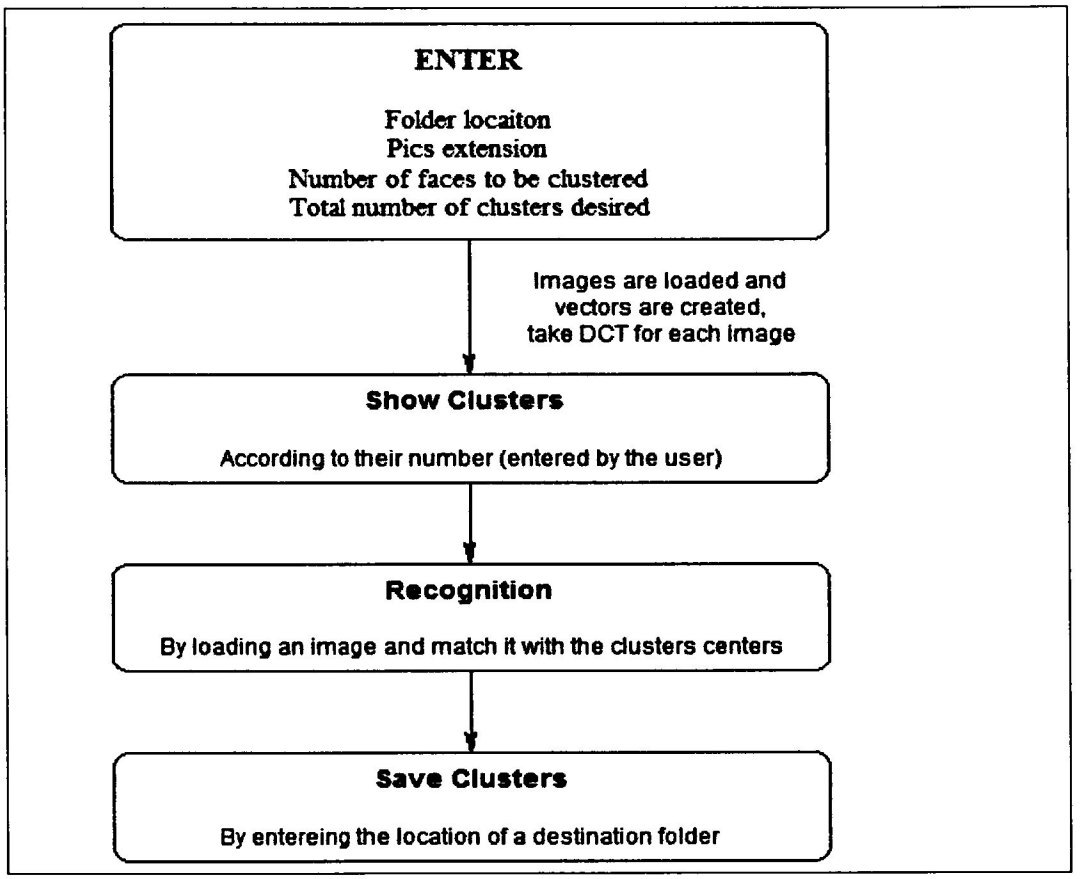


Fig. 1. The system model

1.4 System Characteristics

Other than knowing the basic functions of a computer, no other skills are needed. Any user will find the system user-friendly and easy to understand. The system does not take into consideration any assumption or dependency.

1.5 System Functionality

The system functionalities are explained in the following headings.

- User interface.
- Load location of images, number if images to be clustered and their extension and number of clusters desired.
- Convert images into vectors.
- Apply DCT into the images.
- Display images and central images.
- Display nearest central image (after matching).
- Save clusters.

1.6 Interface Requirements

The interfaces required are categorized in the following phases. A Graphical User Interface (GUI) was developed to make it easier for the user to effectively use the system. From that interface, the user will enter the desired information and the system will then produce the respective results. The proposed system will work on any computer that has the ability to run Matlab7 and will run on Windows98/XP or its compatible operating systems.

1.7 Motivation

The process of clustering is an important function in information processing and machine learning. Clustering refers to grouping several sets of data into one or more distinct groups based on certain pre-defined, or automatically determined, data

characteristics. In particular, clustering has been extremely useful in automated classification of data, as in automated face recognition and helps to speed up the process of image matching.

1.8 Applications

Face recognition can be used and has been attempted in the following areas [1]:

- Entertainment: In entertainment it has been used for these specific applications:
    i. Virtual reality
    ii. Human Robot Interaction
    iii. Human Computer Interaction
- Smart Cards: Currently, people carry a variety of smart cards and each of them has access control. Some of the applications of face recognition are:
    i. National ID
    ii. Passport
    iii. Driving License
    iv. Voter Registration
- Information Security: Some of the applications in information security are:
    i. TV Parent Control
    ii. Personal Device Logon
    iii. Desktop Logon
- Law Enforcement and Surveillance: It is one of the areas where it is widely accepted because of the threat of disruptive forces to civilization.

i. CCTV control

ii. Theft control

iii. Terrorist operation and investigation

1.9 System Specifications

In order to measure the performance of the Face Clustering/Recognition system developed, a different standard face databases was used: AT&T, Gtech, FERET, etc. The minimum system requirements to run Face Clustering/Recognition system so that it works efficiently, are listed below. These requirements are part of the industry-based standards on which it is developed for Windows XP, Pentium(R) M processor 1.73GB with 0.99GB of RAM.

1.10 Organization of Report

Chapter 2 contains a literature review of Face recognition. Chapter 3 is a literature review of image clustering. Chapter 4 contains the face database clustering system. Chapter 5 is the user manual. Appendix A contains testing of the Haar face detection software and Appendix B is the Matlab code of our system.

CHAPTER 2

LITERATURE REVIEW ON FACE RECOGNITION

2.0 Biometrics

"Any automatically measurable, robust and distinctive physical characteristic or personal trait that can be used to identify an individual or verify the claimed identity of an individual can be considered a biometric" [1]. Here is the definition of the terms used in the previous paragraph:

- Measurable: Easy to be measured. The characteristic can be easily presented to the sensor.
- Robust: Does not change over time.
- Distinctive: The differences in the biometric among the population.

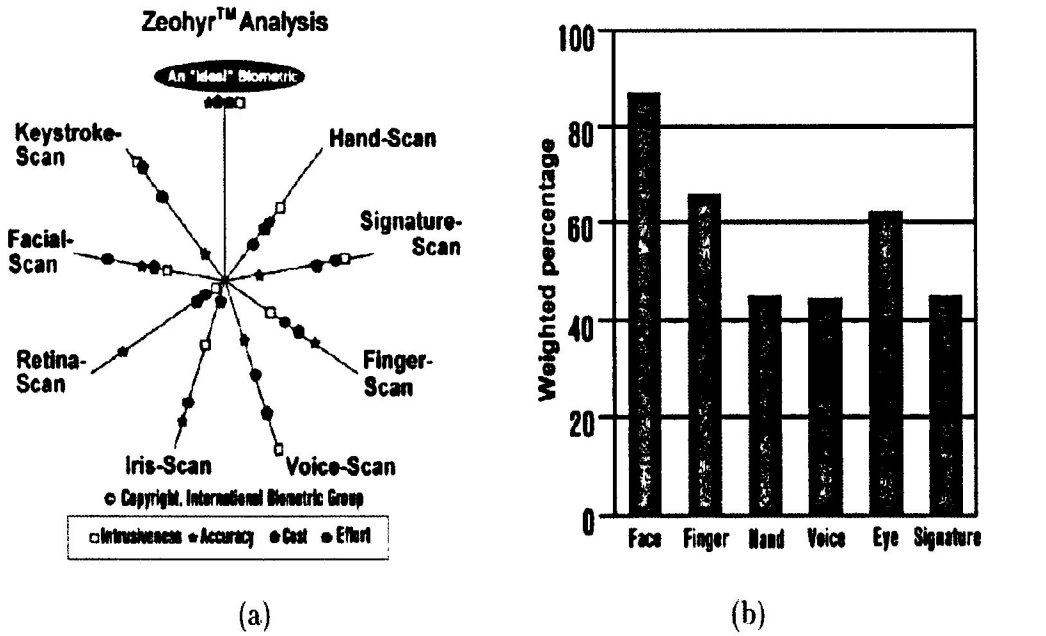
In today's world there are many types of available biometrics that include, but are not limited to: Iris scan, Retinal scan, Voice recognition, Finger print, Ear geometry, Face recognition, Hand geometry, Dynamic Signature verification, Keystroke dynamics, DNA analysis. In fact, any human feature can be considered as a biometric; however, there are some attributes that should be validated before a feature can be considered as one.

2.0.1 Biometrics Attributes

These attributes include, as discussed in the National Center for State Courts:

- Verify: Whether or not the Biometric is capable of verification. Verification is the process by which an input is compared to specific data previously recorded from the user, to find out if the person is who they claim to be.
- ID: Whether or not the Biometric is capable of identification. Identification is the process by which an input is compared to a large data set previously recorded from many people, to determine which person the user is.
- Accuracy: How well the Biometric is able to tell individuals apart. This is partially determined by the amount of information gathered, as well as the number of possible different data results.
- Reliability: How dependable the Biometric is for recognition purposes.
- Error Rate: This is calculated as the crossing point, when graphed of false positives and false negatives created using this Biometric.
- Errors: Typical causes of errors for this Biometric.
- False Pos.: How easy it is to create a false positive reading with this biometric, for example, someone is able to impersonate someone else.
- False Neg.: How easy it is to create a false negative reading with this biometric, for example, someone is able to avoid identification as oneself.
- Security Level: The highest level of security at which this Biometric is capable of working.
- Long-term Stability: How well this Biometric continues to work without data updates over long periods of time.
- User Acceptance: How willing the public is to use this Biometric.

- Intrusiveness: How much the Biometric is considered to invade one's privacy or require interaction by the user.
- Ease of Use: How easy this Biometric is for both the user and the personnel involved.
- Low Cost: Whether or not there is a low-cost option for this Biometric to be used.
- Hardware: Type and cost of hardware required to use this Biometric.
- Standards: Whether or not standards exist for this Biometric.



(a) (b)

(a) Based on Zephyr Analysis (b) Based on MRTD

Fig. 2. Comparison of Various Biometric Features

2.1 Face Recognition

2.1.0 Need for Face Recognition

Recognition means the action of acknowledging the existence of a given input. In this case, the existence of a given face within a database of known faces is being acknowledged. The necessity of this process is increasing in today's world as more and more activities are becoming automated and as a result, people are required to remember several passwords to access these systems. Biometric systems simplify the task by using human features to access the systems. In face recognition, the human face is used for granting access to an individual; therefore, the advantage of face recognition over other types of biometrics can be summarized as follows:

- Uses public information (FACES).
- Involves Contact-Free Process.
- Uses legacy systems.
- Integrates with existing systems.
- No special hardware, other than a camera is needed

For many reasons we can understand how face recognition is a more pleasant method of identification that uses a contact free process to allow one to have his face recognized many times, in many databases, without even realizing it (video-based face recognition). Also it uses public information (face databases), legacy systems, and can also integrate with the existing systems. Nevertheless, little has been done to further the advancement of face recognition; otherwise we would be able to enter restricted access locations every day, without needing to show identification. Video-based face recognition has especially had relatively poor testing and research performed, so far.

2.1.1 Past Work

One of the foremost works on classifying faces can be traced back to 1888 when Francis Galton published his paper titled, "Personal Identification and description", in *Nature*, June 21, 1888, pp 173-177. Some of the work in this field can be traced back to the 1950's in psychology when LS Bruner and R. Tagiuri published their work titled, "The perception of People" in *Handbook of Social Psychology*, Vol-2 pp.634-654, Reading MA: Addison-Wesley, 1954. But the real work on face recognition began in the 1970's after T. Kanade [3] and M. D. Kelly [4] published their works on computer recognition of human faces and visual identification of people by computer. In the past, the face recognition problem was considered merely as a 2-D pattern recognition problem and as a result, many of the earlier techniques were based on the distance matching between two points in a face. Current approaches manage this problem by identifying 3-D objects from 2-D images. During the past few years, availability of hardware and opportunities in commercial fields have given a thrust to this area. Therefore people have adapted different techniques to solve this problem. The most widely used approaches are illustrated in the different techniques of face recognition in the following table [2].

Table 1: Comparison of The Face Recognition Techniques

Appearance Based	**Feature Based**	**Hybrid Based**
This technique uses whole face region	It uses geometrical features.	It uses both whole face and geometrical features
It is applied to specific regions of face or as a whole	It establishes geometric features relationships	

2.1.2 Algorithms Used

Based on the techniques used, there are different methods for recognition. The following table illustrates some of them.

Table 2: Different Recognition Methods

Holistic Approach	**Feature Based Approach**	**Hybrid Based**
Eigenfaces	Pure Geometry	Modular EigenFaces
Probabilistic Eigenfaces	Dynamic Link Architecture	Hybrid Local feature
Fisherfaces, ICA	Hidden Markov Model	Shape-normalized
Nearest feature Lines		Component Based

Algorithms used in face recognition can be classified to image-based face recognition and video-based face recognition.

1. Image Based Face Recognition
    - PCA (Principal Component Analysis)
    - ICA (Independent Component Analysis)
    - LDA (Linear Discriminant Analysis)
    - EBGM (Elastic Bunch Graph Matching)
    - HMM (Hybrid Hidden Markov Model)
    - Bayesian Framework
    - 3-D Morphable Model
    - Kernel Methods
    - Fourier Transform

2. Video Based Face Recognition

Basically, video-based face recognition is involved with streams rather than images; however, it splits the video sequence into number of images, and then does the detection. In this way, it is also involved with images. As a result, the same methods can be used for the video-based face recognition; however, it is a much harder task because of the smaller quality of images extracted from any given surveillance camera. Hence, the image size is smaller, therefore a fewer number of pixels are available for a given feature. Moreover, the orientation of the image will not be consistent, which makes the takes even more difficult, since the poses of the captured images are likely to be different from the ones in the databases.

2.1.3 Face Recognition Difficulties

Although face recognition happens to be a very useful and powerful biometric, as discussed earlier, there are many difficulties that challenge the face recognition process. These are:

- 3D head pose.

- Illumination.

- Facial expression.

- Occlusion due to other objects or accessories.

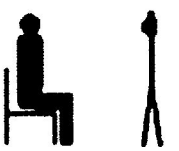
- Facial hair

- Aging

2.1.4 The Five Facial Recognition Steps

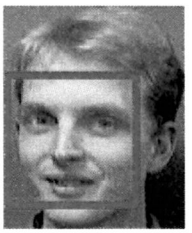
Generally there are five steps for any complete facial recognition system, starting from capturing the image, up to declaring the similarity. Our system involved comparing templates (step 4), which will be described later on in this thesis.

1. Capture Image.



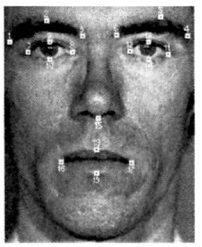
The task of capturing the image, whether by a video camera or a still camera and, whether the person being captured knows it or not, is based on the application. For example, many airports now use a video surveillance system to capture faces and perform facial recognition on them later.

2. Find The Face Area.



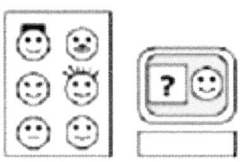
This is the task of "detecting" the face area in the image captured from step one. In most scientific papers it is referred to as "face detection,"

3. Extract Features (To generate template)



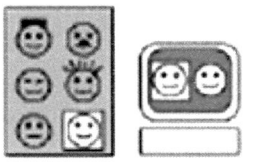
This is the most important step in the face recognition process. It differs from one face recognition algorithm to another and sometimes it is done manually.

4. Compare Templates.



The task of comparing the given face with the existing databases is the most time consuming step in the face recognition process. Therefore, an algorithm to enhance the matching performance (High-speed k-means Image Clustering using the Discrete Cosine Transform) was developed, which will be discussed later in this document with the experiments/results. Comparing the distance can be calculated using different distance measurement methods such as Euclidean distance, Mahalanobis distance, Hamming distance, Levenshtein distance, etc.

5. Declare Matches.



The task of declaring the matches according to their similarity with the one that was input, is considered as the "output" of a facial recognition system.

2.1.5 The Current Facial Recognition Vendors

Here's a list of some existing facial recognition vendors.

- Images Technologies Inc.
- Neurotechnologija
- C-VIS Computer Vision und Automation GmbH
- DreamMirh Co., Ltd.
- Neven Vision, Inc.
- Animetrics Inc.

- Geometrix, Inc.
- Cognitec Systems GmbH (my heritage)
- Cybula Ltd.
- Iconquest
- Takumi Vision Technologies, Inc.
- Viisage
- VisionSphere Technologies Inc.
- A4Vision, Inc.
- AcSys Biometrics Corp.
- Identix Inc.

2.1.6 Case Study Algorithms

In this project, five different face recognition systems have been studied and implemented and they have been compared based on the results. The following is a brief description of them, followed by their results:

- Face Recognition by PCA

PCA stands for Principal Component analysis. It extracts the available objects on the face such as eyes, nose, and mouth, and the relative distance between them. These features are characteristic to a particular face and known as eigenfaces or principal components. PCA converts each original image in the training set to eigenfaces. Each eigenface represents certain features of the face. Therefore, reconstruction of the original image from eigenface is possible by adding eigenface in training set in the right

proportion. Reconstructed image represents an approximation of the original image; therefore, in this method, each face is a linear combination of the eigenfaces. Distance measure methods help in identifying the test image from the eigenface.

- Face Recognition by Gabor Filters

In this method, the face from the original image is projected onto an elastic grid. Gabor filter bank response for each grid node is calculated. . These nodes are known as fiducial points, which represents center of the eyes, top of the nose etc. Face recognition is performed by measuring the similarity of the filter response at each node.

- Face Recognition by Fisherfaces

This method is based on PCA. It uses FLD (Fisher Linear Discriminant) as a classification algorithm. It gives emphasis on feature sets with variations such as illumination condition, facial expression etc. It takes more time in offline learning compared to on-line learning. It is more robust and measures individual characteristics accurately. It uses Euclidean distance for recognition

- Face Recognition by Fourier Spectra

Fourier spectra of an input original image split into sine and cosine components and the output represents the image in frequency domain. In Fourier spectra, each point represents a particular frequency contained in the spatial image. The amplitude at each frequency point represents the energy contained in that point. DFT or FFT only contains a set of samples, which describes the input image. In face recognition, this property is utilized because energy contained in each person's face is not uniform. Therefore,

frequency domain correlation between test image and images in the database is used as a measure for recognition.

- Face Recognition by Fourier-Bessel Transform

Fourier-Bessel transform is widely used in analyzing patterns in circular domain because it represents radial and angular components in an image. In square image, it represents the distance from the center of the region to one of its corners. For every image FB coefficients are extracted. Given a test image, its Fourier Bessel coefficients are compared with the stored coefficients in the database. Distance measure is used for recognition and the stored image with least distance to test image is confirmed as detected image.

2.1.7 Our Testing System

A testing system was designed to test the five algorithms described above on the AT&T face database. A GUI that combines all the above algorithms, and another for testing them were also developed.

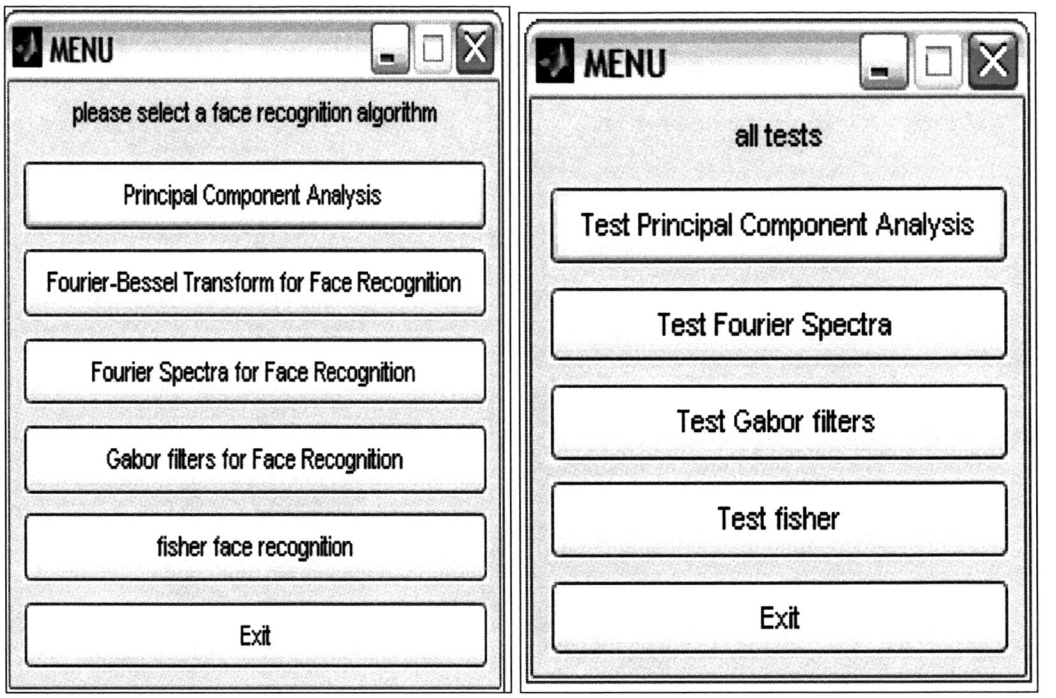


Fig. 3a. The System Graphical User Interfaces

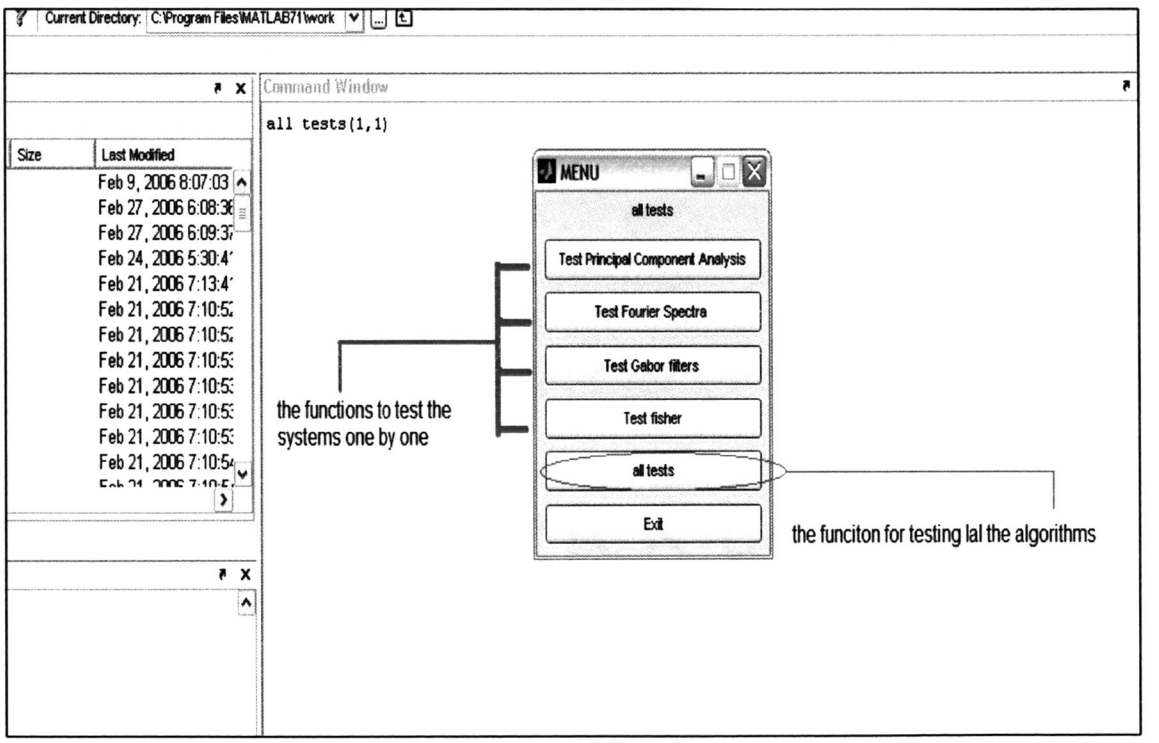


Fig. 3b. The GUI for the Algorithms and the Testing

2.1.8 The Testing Routine

The testing routine reads all images and stores them in a variable. Each time random images are selected (known faces) we can chose the number of known images, chosen randomly for each person, (the first input of the function) then the recognition step is done. If the "recognized" face does not correspond to the "true face" a variable is incremented by one; one error in recognition, and so on.

2.1.9 Our results

The following figure is a snapshot from the results screen. It illustrates the percentage of a successful recognition for each algorithm. They were later added to a chart in Figure 5.

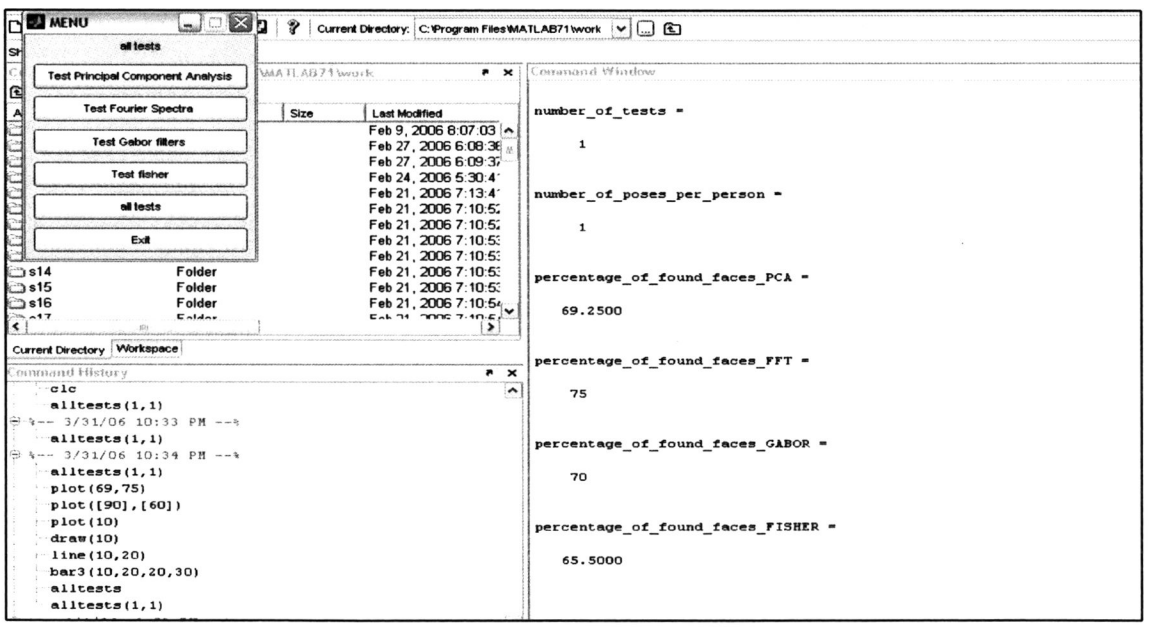


Fig. 4. The results in the command window

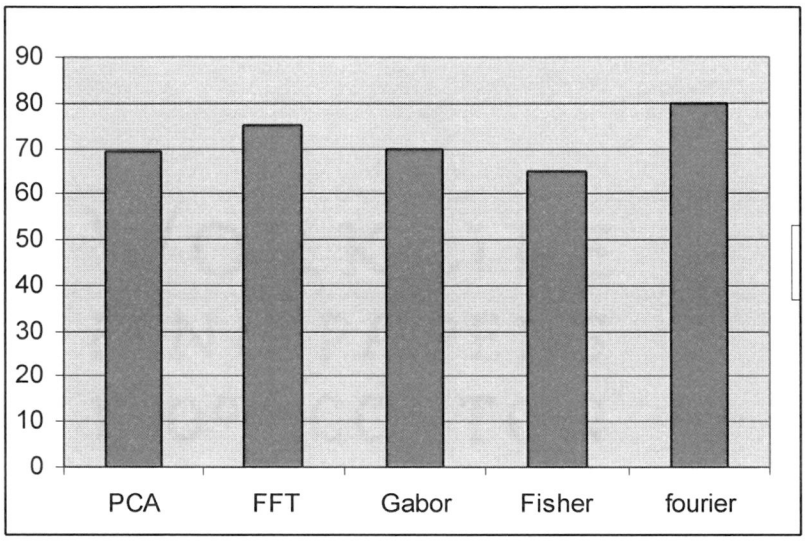


Fig. 5. The results in a chart

2.1.10 Using the Face Recognition Algorithms

Using the face recognition as a reliable system is not yet a good idea. It is enough to look at the Palm Beach Airport experience to realize how the facial recognition systems are still in their developing phase. At Palm Beach Airport, there were 15 volunteers with a total number of 250 images. The success rate was less than 50% and there were 50 False alarms per 5000 passengers, which makes two-three false alarms/hour per check point. The system was sensitive to eyeglasses, poses, facial hair and lighting.

2.1.11 Why The Bad Results?

The problem is that many systems still treat images as numbers of pixels and perform mathematical calculations to find the differences. For example, observe a

sample from the Max Planck Institute for Biological Cybernetics Face Database, which is a database with the following characteristics:

No. of Subjects	Conditions		Image Resolution
200	Modality Pose	2 7	256 × 256
http://faces.kyb.tuebingen.mpg.de/			

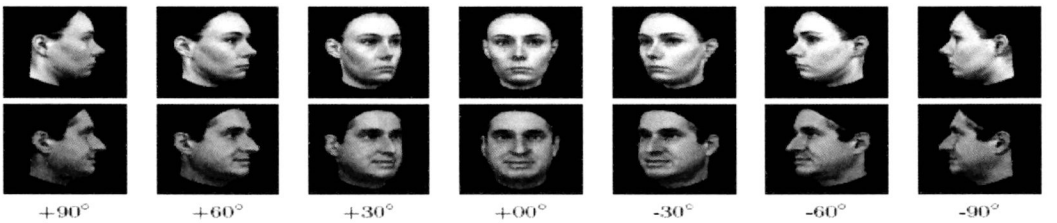


Fig. 6a. Max Planck Institute for Biological Cybernetics Face Database

If the following example is taken from this database:

Fig. 6b. A Typical Recognition Mistake

Most conventional measures of image similarity would declare images one and two to be more similar than images one and three, although one and three are images of

the same person! Obviously, similarity needs to be computed over attributes more complex than raw pixel values. This example illustates how the facial recognition task needs to be looked at from a different angle [5].

2.1.12 Future of Face Recognition

As discussed earlier in this chapter all the existing facial recognition systems treat face images as a number of pixels. This limited vision of dealing with the face images has shown failure. The future of face recognition lies in viewing faces from different perspectives and attempting to find the important cues in the face that provide understanding of the differences. It was suggested by a group at MIT [5] that the communication between the human vision researchers and the computer vision researchers should be increased because:

- The only system that does seem to work well, even under severely degraded viewing conditions is the human face recognition [5].
- Together, they can design an automated system that can match, and eventually exceed the current level of performance.

The following questions are important to determine the future of face recognition. We must know:

1. What are the limits of human face recognition abilities?

It's impressive how our human face recognition allows us to recognize faces under severely degraded viewing conditions. Perhaps our human face recognition worked as a pre-alarm and is the reason for our very survival [5].

Prince Charles Woody Allen Bill Clinton Saddam Hussein Richard Nixon Princess Diana.

Fig. 7. Faces that we can still recognize

2. What are some important cues that the human visual system rely upon for judgments of identity?

We are still trying to find out the important cues on which our human face recognition rely. We think that understanding the work of caricaturists, minimalist portrait artists, can help us understand the most important cue in a face [5].

3. Is the human FR innate or learned?

A group of researchers attempted to study the newborn's behavior towards faces to help understand whether we were born with a smart face recognition system or it if is learned over time, and the results were encouraging. Evidence revealed that newborns could recognize their mother's picture and that they showed a preference for face images over non-face images. Such performance can only be achieved if newborns already possess at least a rudimentary facial processing mechanism [5].

CHAPTER 3

IMAGE CLUSTERING

LITERATURE REVIEW

3.0 Introduction

Image clustering, also known as image classification or categorization, is a means for high-level description of image content. In recent years there has been a growing interest in developing effective methods for searching large image databases based on image content. There is more than one way of achieving this. For example, one might wish to cluster the images based on scene content; forest, agricultural, or urban scenes. Or, perhaps one might wish to cluster all images into groups with the same lighting or with the same pose, such as faces used in face recognition software. This is a very difficult thing to achieve because this categorization is based on the identity of the 3-D objects that the images represent, but where the observer's viewpoint has varied between images. Another way to group similar images together is by their actual chromatic content.

The two main approaches for image database interface, applied to existing systems are: "search-by-query" and "browsing." Most approaches to image database management have focused on the "search-by-query"" method. In content-based search, the goal is to retrieve the most similar images to a query image introduced to the system. The images belonging to the query-image category are the images we wish to retrieve

first and characterize each category's unifying characteristics. A second approach to categorization of image information is to view it as a clustering task, unsupervised or supervised, in which the goal is to find relationships among the images and the best way to characterize the content within a given image archive.

3.1 History

Today there is a large variety of image clustering algorithms and more than one method used for improving image search performances. By the level of the autoimmunization of the search method we have two cases:

A) Supervised - where the user usually specifies some other parameters for the search, other than the image itself.

B) Unsupervised - where the search is completely automatic, and no user input is necessary, other than the image reference. By the algorithm itself, we can have:

- Heuristic-based. Like the Pattern Matrix, a prototype-based algorithm (k-means, k-medoid) or the Proximity Matrix. There are two variations of the Proximity Matrix: the Linkage methods (single-link, clustering, min-cut);
- Model-based. Two variations of this model are used: Spatial clustering and the Mixture Model (Gaussian mixture, Latent class);
- Density-based. We have the variations: Kernel-based (DENCLUE) and Mode-seeking (mean-shift)

The most popular partitional clustering algorithm, k-means, has been proposed several times in the literature by Steinhaus in 1955, Lloyd in 1957, and MacQueen in

1967. The ISODATA algorithm by Ball and Hall in 1965 can be regarded as an adaptive version of k-means that adjusts the number of clusters. The historical account of vector quantization also presents the history of some of the partitional clustering algorithms. In 1971, Zahn proposed a graph-theoretic clustering method [6], which is closely related to single-link clustering. The EM algorithm, which is the standard algorithm for estimating a nite mixture model for mixture-based clustering, is attributed to Dempster et al. in 1977 [7]. Interest in mean-shift clustering was revived in 1995 by Cheng [8], and Comaniciu and Meer further popularized it [9]. Ho man and Buhmann considered the use of deterministic annealing for pairwise clustering [10], and Fischer and Buhmann modified the connectedness idea in single-link clustering that led to path-based clustering [11]. The normalized cut algorithm by Shi and Malik [12] in 1997 is often regarded as the first spectral clustering algorithm, though similar ideas were considered by spectral graph theorists earlier. A summary of the important results in spectral graph theory can be found in the 1997 book by Chung [13]. The emergence of data mining leads to a new line of clustering research that emphasizes efficiency when dealing with huge databases. DBSCAN by Ester et al [14] for density-based clustering and CLIQUE by Agrawal et al [15] for subspace clustering are two well-known algorithms in this community. The current literature on cluster analysis is vast, and hundreds of clustering algorithms have been proposed in the literature. It will require a tremendous effort to list and summarize all the major clustering algorithms.

3.2 Clustering Algorithms

The k-means algorithm and the EM algorithm are clustering algorithms. Other clustering algorithms that are used regularly in pattern recognition include the mean-shift algorithm, pair wise clustering, path-based clustering, and spectral clustering.

3.2.0 k-means Algorithm

The *k*-means algorithm is probably the best-known clustering algorithm. In this algorithm, the j-th cluster is represented by the "cluster prototype" μ_j in R^d. Clustering is done by finding z_i and μ_j that minimize the following cost function:

$$J_{k\text{-}means} = \sum^{n}_{i=1} ||y_i - \mu_j||^2 = \sum^{n}_{i=1} \sum^{k}_{j=1} I(z_i=j) ||y_i - \mu_j||^2. \quad (1)$$

Here, $I(z_i=j)$ denotes the indicator function, which is one, if the condition $z_i = j$ is true, and zero, otherwise. To optimize $J_{k\text{-}means}$, we first assume that all μ_j| are specified. The values of z_i that minimize $J_{k\text{-}means}$ are given by

$$z_i = arg\ min_j\ ||yi - \mu_j||^2. \quad (2)$$

On the other hand, if z_i is fixed, the optimal μ_j can be found by differentiating $J_{k\text{-}means}$ with respect to μ_j and setting the derivatives to zero, leading to:

$$\mu_j = (\sum^{k}_{j=1} I(z_i=j)\ \mu_j)/(\sum^{k}_{j=1} I(z_i=j)\ ) = (\sum^{n}_{i=1} z_i=j^{\mu j})/(number\ of\ i\ with\ z_i=j) \quad (3)$$

Starting from an initial guess on μ_j , the k-means algorithm iterates between Equations (2) and (3), which is guaranteed to decrease the k-means objective function until a local minimum is reached. In this case, μ_j and z_i remain unchanged after the iteration, and the *k*-means algorithm is said to have converged. The resulting z_i and μ_j constitute the clustering solution. In practice, one can stop if the change in successive values of $J_{k\text{-}means}$ is less than a threshold.

The *k*-means algorithm is easy to understand and is also easy to implement. However, *k*-means has problems in discovering clusters that are not spherical in shape. It also encounters some difficulties when different clusters have a significantly different number of points. *K*-means also requires a good initialization to avoid getting trapped in a poor local minimum. In many cases, the user does not know the number of clusters in advance, which is required by *k*-means. The problem of determining the value of *k* automatically, still does not have a very satisfactory solution. Because the *k*-means algorithm alternates between the two conditions of optimality, it is an example of alternating optimization. The *k*-means clustering result can be interpreted as a solution to vector quantization, with a codebook of size *k* and a square error loss function. Each μ_j is a codeword in this case.

The *k*-means algorithm can also be viewed as a special case of fitting a Gaussian mixture, with covariance matrices of all the mixture components fixed to be $\sigma^2 I$ and σ tends to zero (for the "hard" cluster assignment). The *k*-medoid algorithm is similar to *k*-means, except that μ_j is restricted to be one of the given patterns y_i. There is also an online version of *k*-means.

When the i-th data point y_i is observed, the cluster center that is the nearest to y_i is found. μ_j is then updated by

$$\mu_j^{new} = \mu_j + \alpha(y_i - \mu_j) \quad (4)$$

where α is the learning rate. This learning rule is an example of "winner-take-all" in competitive learning, because only the cluster that "wins" the data point can learn from it.

3.2.1 Clustering by Fitting Finite Mixture Model

The k-means algorithm is an example of "hard" clustering, where a data point is assigned to only one cluster. In many cases, it is beneficial to consider "soft" clustering, where a point is assigned to different clusters with different degrees of certainties. This can be done either by fuzzy clustering or by mixture-based clustering. The latter is preferred because it has a more rigorous foundation. In mixture-based clustering, a finite mixture model is fitted to the data. Let Y and Z be the random variables for a data point and a cluster label, respectively. Each cluster is represented by the component distribution $p(Y_j|\theta_j)$, where θ_j denotes the parameter for the j-th cluster. Data points from the j-th cluster are assumed to follow this distribution, i.e., $p(Y|Z = j) = p(Y_j|\theta_j)$. The component distribution $p(Y_j|\theta_j)$ is often assumed to be a Gaussian when Y is continuous, and the corresponding mixture model is called "a mixture of Gaussians". If Y is categorical, multinomial distribution can be used for $p(Y_j|\theta_j)$. Let $\alpha_j = P(Z = j)$ be the prior probability for the j-th cluster. The key idea of a mixture model is

$$p(Y|\Theta)=\sum_{j=1}^{k}P(Y|Z=j)P(Z=j)=\sum_{j=1}^{k}\alpha_j p(Y_j|\theta_j) \quad (5)$$

where $\Theta = \{\theta_1; : : : ; \theta_k; \alpha_1; : : : ; \alpha_k\}$ contains all the model parameters. The mixture model can be understood as a two-stage data generation process. First, the hidden cluster label Z is sampled from a multinomial distribution with parameters $(\alpha_1; : : : ; \alpha_k)$. The data point Y is then generated according to the mixture distribution determined by Z, i.e., Y is sampled from $p(Y_j|\theta_j)$ if $Z = j$. The degree of membership of yi to the j-th cluster is determined by the posterior probability of Z equals to j given y_i, i.e.,

$$p(Z=j|Y=y_i)=p(Z=j,Y=y_i) / p(Y=y_i)= (\alpha_j p(Y_j|\theta_j)) / \sum_{j=1}^{k}\alpha_j p(Y_j|\theta_j) \quad (6)$$

If a "hard" clustering is needed, y_i can be assigned to the cluster with the highest posterior probability $p(Z=j|Y=y_i)$. The parameter Θ can be determined using the maximum likelihood principle. We seek Θ that minimizes the negative log-likelihood:

$$J_{mixture} = - \sum_{i=1}^{n}\log\sum_{j=1}^{k}\alpha_j p(y_j|\theta_j) \quad (7)$$

For brevity of notation, we write $p(y_j|\theta_j)$ to denote $p(Y=y_j|\theta_j)$

The EM algorithm can be used to optimize J mixture. EM is a powerful technique for parameter estimation when some of the data is missing. In the context of a finite mixture model, the missing data are the cluster labels. Starting with an initial guess of the parameters, the EM algorithm alternates between the "E-step" and the "M-step". Let r_{ij} =

$P(Z=j|Y=y_i,\Theta^{old})$, where Θ^{old} is the current parameter estimate. In the E-step, we compute the expected complete data log-likelihood, also known as the Q-function:

$$Q(\Theta\,||\Theta^{old}) = E[\sum^{n}_{i=1} \log p(y_i,z_i)] = \sum^{n}_{i=1}\sum^{k}_{i=1} r_{ij}(\log \alpha_j + \log p(y_j|\theta_j)) \qquad (8)$$

Note that the expectation is done with respect to the old parameter value via r_{ij}. Computationally, E-step requires calculation of rij . In the M-step, Θ that maximizes $Q(\Theta\,||\Theta^{old})$ is found:

$$\Theta^{new} = \arg\max_{\Theta} Q(\Theta\,||\Theta^{old}) \qquad (9)$$

The M-step is guaranteed to decrease $J_{mixture}$. By repeating the E-step and the M-step, the negative log-likelihood continues to decrease until a local minimum is reached.

Convergence proofs on the EM algorithm. In this section, we shall state the well-known proof in the literature that the M-step indeed decreases $J_{mixture}$, thereby showing that the EM algorithm does converge to a local minimum of $J_{mixture}$. We consider the correctness of the EM algorithm in a more general setting, where Y and Z are redefined to mean "observed data" and "missing data" respectively. Note that the data points and the missing labels are examples of observed data and missing data, respectively.

In this general setting, $Q(\Theta\,||\Theta^{old})$ can be written as

$$Q(\Theta\,||\Theta^{old}) = \sum_{Z} p(Z|Y\,\Theta^{old}) \log p(Y,Z|\,\Theta) \qquad (10)$$

Our first proof is based on the concavity of the logarithm function. Because M-step maximizes $Q(\Theta)$, $Q(\Theta^{new}) - Q(\Theta^{old}) >= 0$. Observe that:

$$Q(\Theta^{new}) - Q(\Theta^{old}) = \sum_Z p(Z|Y,\Theta^{old})(\log p(Y,Z|\Theta^{new}) - \log p(Y,Z|\Theta^{old}))$$

$$= \log p(Y|\Theta^{new}) - \log p(Y|\Theta^{old}).$$

The inequality is due to the concavity of logarithm, and the fact that $p(Z|Y,\Theta^{old})$ can be viewed as "weights" because they are non-negative and $\sum_Z p(Z|Y,\Theta^{old}) = 1$. Since $QQ(\Theta^{new}) - Q(\Theta^{old}) >= 0$, the above implies $\log p(Y|\Theta^{new}) - \log p(Y|\Theta^{old}) >= 0$. So, the update of parameter from Θ^{old} to Θ^{new} indeed improves the log-likelihood of the observed data. When $\Theta^{old} = \Theta^{new}$, the inequality becomes an equality, and we reach a local minimum of $\log p(Y|\Theta)$. Note that the above argument holds as long as $Q(\Theta^{new}) - Q(\Theta^{old}) >= 0$. Thus it suffices to increase, instead of maximize the expected complete log-likelihood in the M-step. The resulting algorithm that only increases the expected complete log-likelihood is known as the generalized EM algorithm. It is interesting to note a variant of the EM algorithm used Bayesian parameter estimation. The goal is to find Θ that maximizes $\log p(Y|\Theta)$. The E-step computes $\int p(\Theta|Z,Y) \log \text{plog } p(Y|\Theta^{old},Y)$ dZ, and the M-step solves $\Theta^{new} = \arg\max_\Theta \int p(Z|\Theta^{old},Y) \log p(\Theta|Z,Y)\, dZ$. The correctness of this variant of the EM algorithm can be seen by the following:

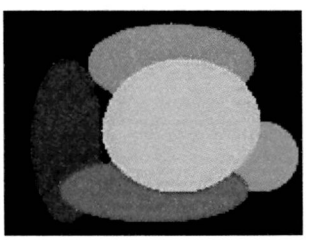
$$\int p(Z|\Theta^{old},Y)\log p(\Theta^{new}|Z,Y)dZ - \int p(Z|\Theta^{old},Y)\log p(\Theta^{old}|Z,Y)dZ$$

$$= \int p(Z|\Theta^{old},Y)\,(\log p(\Theta^{new}|Y) + \log p(Z|\Theta^{new},Y) - \log P(Z|Y) - \log p(\Theta^{old}|Y) - \log p(Z|\Theta^{old},Y) + \log P(Z|Y)dZ <= \log p(\Theta^{new}|Y) - \log p(\Theta^{old}|Y)$$

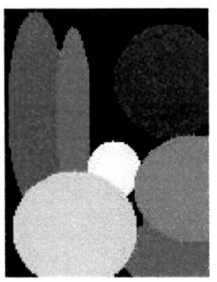
Note that $p(\Theta|Z,Y) = p(\Theta|Y)p(p(Z|\Theta,Y)/p(Z|Y)$.

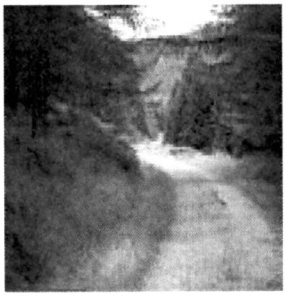
3.2.2 GMM Gaussian mixture models

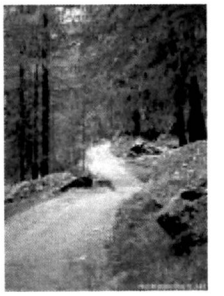
Utilizing the Information Bottleneck method, the image models are grouped into coherent clusters that can be used for various archive operations. The system is based on global image representations including global color, texture and edge histograms. Histograms are the classical means of representing image content. A histogram is a discrete representation of the continuous feature space. The feature space partition is determined by the features chosen, for example, the color space representation, by the quantization scheme chosen, such as uniform or vector quantization, as well as by computational and storage considerations. Color histograms advantages and disadvantages are well studied and many variations of this algorithm exist. A graphical representation of the result of this algorithm is shown below:



A second example is shown below:







This method incorporates region-based approaches for the image representation. Since image regions are the basic building blocks in forming the visual content of an image, they have great potential in representing the image content as well as the category content. Images in the database are divided into rectangular regions and represented by a set of normalized histograms corresponding to these rectangular regions. The size of the regions is an important parameter. Regions should be small enough to emphasize the local color and large enough to offer statistically valid histograms. The similarity measure between two images is expressed as the sum of similarities between histograms of the corresponding rectangular regions.

The main steps of this algorithm are:

- Image Representation

The raw pixel representation of an input image is shifted to a mid-level representation, in which the image is represented as a set of coherent regions in feature space. This work focuses on the color feature. In particular each image is modeled as a mixture of Gaussians in the color feature space. It should be noted that the representation model is a general one, and can incorporate any desired feature space (such as texture, shape, etc) or combination thereof.

- Feature Extraction

Color features are extracted by representing each pixel with a three-dimensional color descriptor in a selected color space. The pixels are then grouped into homogeneous regions based on their assumed features. Used in a database search engine, this algorithm can be a very effective tool, because some of the most irrelevant results of the search can

be eliminated in the first phase of the image processing, and then a more refined search will be performed.

3.2.3 Image Clustering with Metric, Local Linear Structure and Affine Symmetry

This is a totally different algorithm of the ones already described up to this point, because it tries to group the images by the 3-D object that they represent more than the 2-D image. Schematically, this algorithm is similar to the other clustering algorithms described previously. It, as a plus has defined affinity measures between all pairs of images. These affinity measures are represented in a symmetric $n \times n$ matrix $A = (aij)$, i.e., the affinity matrix and a straightforward application of any standard spectral clustering method then yields our clustering result. Here is how it works: First, the image-clustering problem is defined. The input of the problem is a collection of unlabelled images $\{I1, \cdots, In\}$ and the number of clusters N. We assume that all images have the same number of pixels s, and by rasterizing the images, we obtain a collection of corresponding sample points $\{x1, \cdots, xn\}$ in IRs. The algorithm outputs a cluster assignment for these images $\rho : \{I1, \cdots, In\} \rightarrow \{1, \cdots, N\}$. Two images Ii and Ij belong to the same cluster if and only if $\rho(Ii) = \rho(Ij)$. A cluster, in our definition, consists of only images of one object. Let us further assume that the images of a cluster are acquired at different viewpoints but under the same ambient illumination condition.

The problem so formulated is extremely general and without any further information, there is almost no visible structure on which to base the algorithm. One obvious structure one can utilize is the ambient distance metric of the image space. The

usual $L2$ metric or its derivatives, affine-invariant $L2$ distance or weighted $L2$ distance are such examples. By considering images as points in IRs, we are naturally led to the notion of appearance manifolds. Accordingly, the input images imply the existence of N sub manifolds of IRs, $\{M1, \cdots, MN\}$ such that two points xi, xj belong to the same cluster if and only if $xi, xj \square Mk$ for some $1 \leq k \leq N$, with each Mi denoting the appearance manifold of an object.

Implicit in the concept of appearance manifolds is the idea of local linearity. That is, if $x1, \cdots, xl$ are points belonging to the same cluster and if they are sufficiently close according to the distance metric; then each point xi can be well approximated linearly by its neighbors : $xk \approx \sum_{j!=k} ajxj$ for some real numbers aj .

Metric and local linearity are two very general geometric notions and they do not pertain only to image clustering problems. It is the action of the 2-D affine group G 4 that characterizes our problem as an image clustering problem rather than a general data-clustering problem. If $\{x1, \cdots, xn\}$ were data of a different sort, e.g., data from a metrological or high energy physics experiment, there will not be an explicit action of G. It is precisely because the 2-D nature of the images and the way we rasterize the image to form points in IRs, we can explicitly calculate the action of G given a sample point x. In particular, each appearance manifold Mi is invariant under G, i.e., if $x \square Mi$ then $\gamma(x) \square Mi$ for each $\gamma \square G$. In this sense, the clustering problem acquires a symmetry played by the 2-D affine group5.

In summary, three important elements to the image-clustering problem have been identified. First, there is the ambient $L2$, and its derivatives, metric of the image space. Second, each cluster has local linear structure. The metric and local linearity are the only two geometric structures we can utilize in designing the algorithm. The third element is the affine symmetry of the problem. The challenge was to design a clustering algorithm that takes into account these three elements. In a very general outline, what is needed is to design metric and local linear structures that are both invariant under the affine group G and to seek an interesting and effective coupling between the metric and linear structure, which are two rather disparate geometric notions. Surprisingly, using only these three very general structures, a clustering algorithm can be formulated in order to be effective for a variety of image clustering problems.

CHAPTER 4

FACE DATABASE CLUSTERING

4.0 Introduction

The process of clustering is an important function in information processing and machine learning. Clustering refers to grouping several data into one or more distinct groups based on certain pre-defined, or automatically determined, data characteristics. In particular, clustering has been extremely useful in automated classification of data, as in automated face recognition. In this chapter, a method for automated classification and recognition of face images from a random set of images is presented using *k-means clustering* and compared with general clustering methods in face clustering. The method is considerably speeded up using the *Discrete Cosine Transform* for dimensionality reduction of image data. The method results in obtaining one or more *central* images, each representing a different person's face found in the face database. Recognition can then be implemented by the nearest neighbor technique or by any other suitable technique. Before discussing the system, consider the following example. If the following database is used, and the nearest match of an entered face within a certain database of faces is sought, any face recognition algorithm would go through the whole set of faces comparing it with the entered face before finding the nearest one. Obviously this task is very time consuming and processor intense.

Fig. 8. The Regular Matching Process

An algorithm that automatically clusters faces together within a given database and automatically creates a center image for each cluster as a pre-recognition phase was designed. As for the recognition phase the entered face is only being compared to the center images, as shown in the following figure:

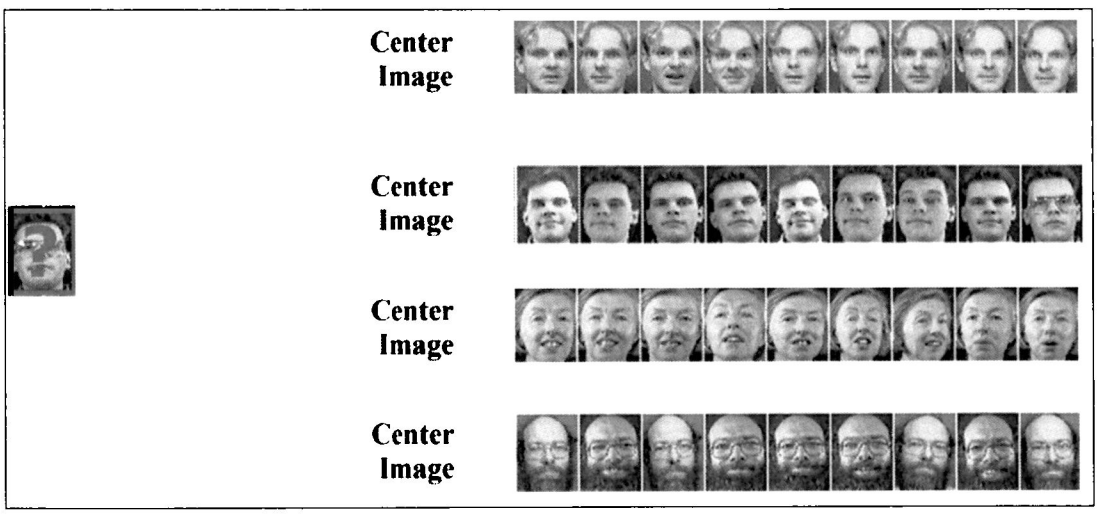


Fig. 9. The Matching with Only The Center Images

Example of the center images created by the system:

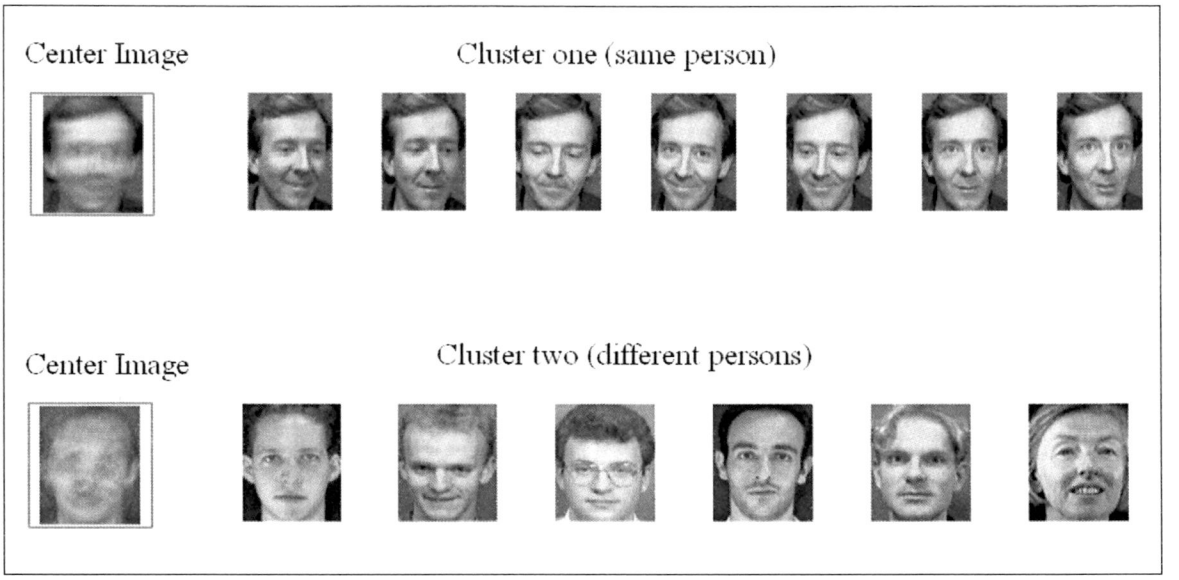


Fig. 10. Center Images Created for Different Clusters

4.1 Description of Our Algorithm

As mentioned earlier, clustering refers to grouping datasets into one or more *clusters* in such a way that all members of the same cluster are similar under one or more standard criteria. Each such cluster may also have a *central datum* that best represents the characteristics of the overall cluster. Several clustering algorithms are commonly used. Popular among these include the *k-means algorithm, fuzzy c-means algorithm* and *self-organizing maps*. In this project, use of the *k-means algorithm* is done for clustering of a database of human face images and the expected output is a number of clusters with one central image for each as shown in figure 10.

4.1.0 Image Clustering using the k-means Algorithm

In this method, initially, a database of face images is considered. This database would generally consist of faces of several persons taken at different times. In this algorithm, the number of distinct persons whose images are found within this database is first determined. This is the *only manual input* to the classification system; the rest is an all *automated* process. Suppose it is known that there are *N different types of faces* in the database. Then we use the *k-means clustering algorithm* to cluster the entire database into *N* different clusters. For this the *N* images are converted into *N* image vectors. From this, we obtain the *central image* for each cluster. This *central image* seems to be the average of many similar images; precisely, it is the *weighted average* of the member images that would best represent the characteristics of all images simultaneously.

4.1.1 Speeding up using the Discrete Cosine Transform

The *k-means algorithm* is among the simplest and fastest of all clustering algorithms. However, this algorithm would still consume an enormously large amount of time because an image vector would be thousands of elements long and tens of vectors are used for clustering. To reduce the time, we have made use of the DCT, which is a popular algorithm for data compression. The following are the steps for clustering using the DCT:

1. First, transform all image vectors into the DCT domain using the DCT algorithm. These vectors will be real vectors and each would be of the same size and an image in the database.
2. The entire DCT vectors for clustering were not used because that would again need

the same amount of time. Instead, only the first few points of the DCT were used. This does not cause much loss of information because DCT is a compression algorithm in which large data can be mapped into data with a smaller size.

3. Then, these DCT vectors were clustered using the k-means method as described above, again getting N central vectors. These vectors are of a smaller size than the image vectors.
4. To bring them back to the original size, zeros were simply padded to the existing vectors to make them of the original length.
5. The DCT central vectors were now converted into image vectors using the inverse DCT algorithm. This is similar to conversion between FFT and IFFT.
6. Upon doing this, the central images as results of clustering were obtained. Note that although the final images are not downsized, the processed vectors were much downsized. This gives a great increase in speed of execution.

4.1.2 Recovering Images from Shortened DCT Signals

The algorithm for using DCT as a compression technique for speeding up k-means clustering was explained in the previous section. In order to prove that DCT does indeed speed up the clustering process without affecting the image quality much, consider an example from the AT&T face set. The following figure shows an original image from this set:

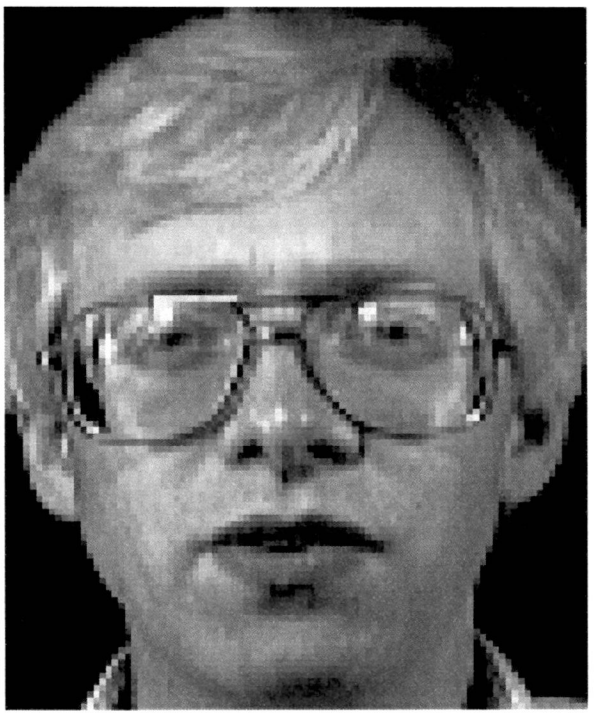


Fig. 11. An original image

It should be understood that this image is actually a 2-dimensional matrix of values. For clustering, it must be converted into a vector. This is simply obtained by

cascading all image rows in order to create a single long vector. The figure 15 on the adjoining page shows the image vector thus obtained.

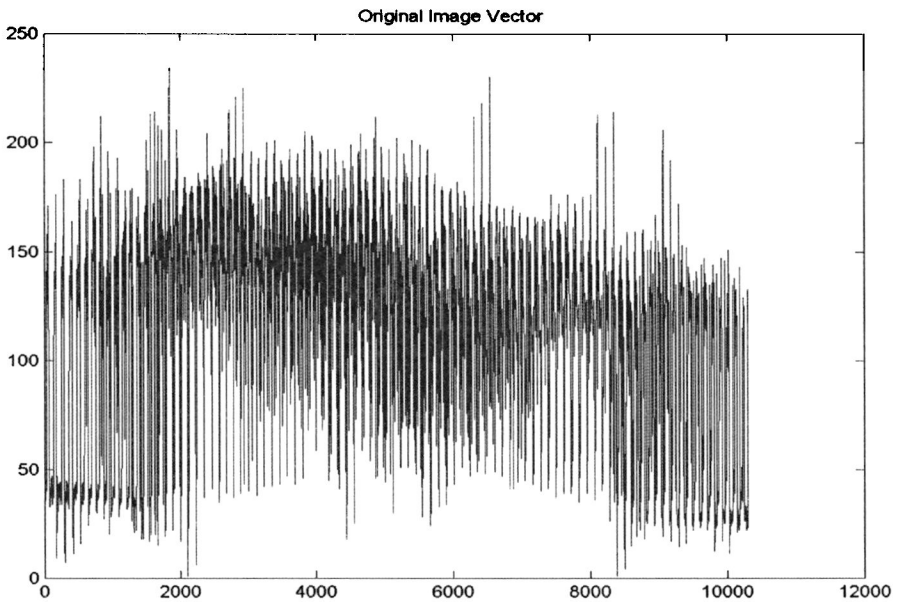


Fig. 12. Image Vector

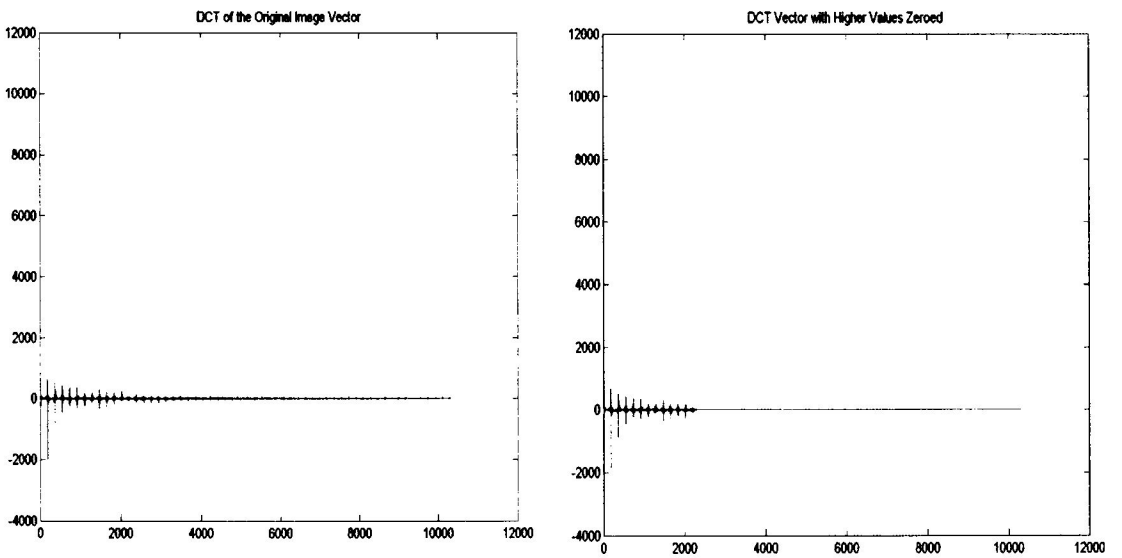


Fig. 13: Original DCT and shortened DCT (with higher values neglected)

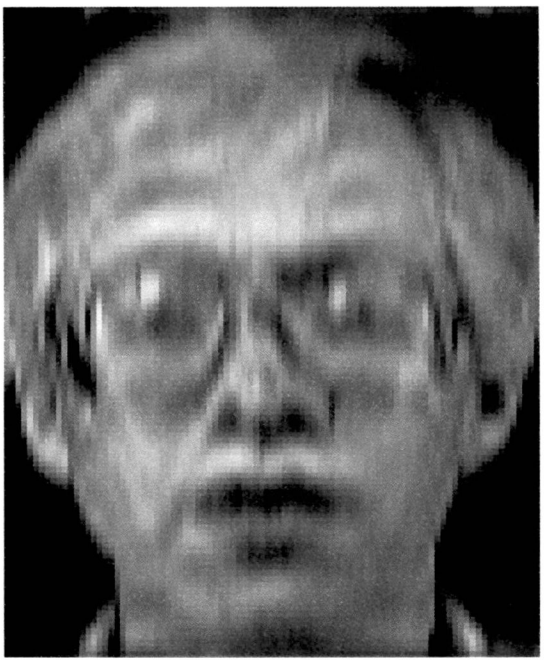


Fig. 14 Image Recovered from Shortened DCT Vector

The plots in figure 15 show the DCT of the image vector in figure 15. The second plot shows that the last eight thousand (8000) values of the DCT plot have been deliberately made zero. Since these values are already near zero, they can be neglected during clustering. Hence the length of the *actual subject vector* is reduced by 8000 points. It is observed that this does not affect the overall shape of the DCT curve. This is an important characteristic of the Discrete Cosine Transform. Under DCT, the *important information* is stored in the first values and the higher values store lesser important information of the original signal. Hence if the higher values are neglected, most of the information is still *preserved.* To prove this, the inverse DCT of the second plot in figure

15 was taken and converted into an image. Upon comparison with the image in figure 13, it is observed that the recovered image looks nearly the same as the original image. However, as earlier noted, the DCT vector *in use* during clustering is of a very small size. As a result, the process of clustering speeds up. The following figures show the results of a sample round of clustering. For this run, 50 images from the AT&T face database were loaded and were clustered into 5 distinct clusters. In a sample cluster shown next, the central (representative) image is shown and it is followed by the images within the database that were classified *into* the same database. This classification is done as a part of the clustering process using the *k-means* algorithm.

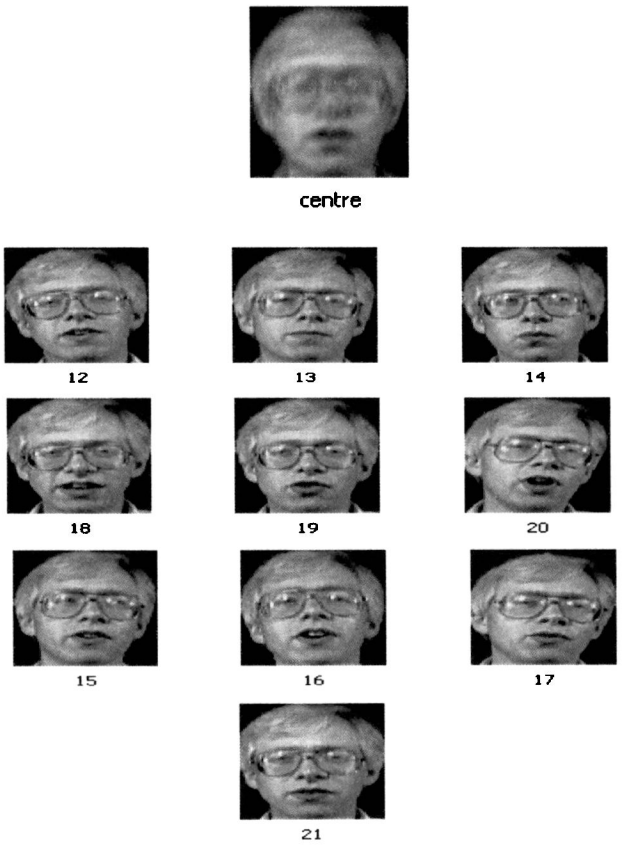


Fig. 15. Cluster Centre and Members

4.2 Performance Analysis

1. General Speed Increase: If the size of the clustered vectors is *L,* then the total execution time would roughly be proportional to *L^2 (square times).* Hence when the DCT vector size was decreased to 1/2, the speed is approximately increased four times. In this project, the DCT vector size was decreased to just $1/10^{th}$ of the original size and hence a speed increase of nearly hundred times was obtained.
2. Speed Advantage over PCA: The computational complexity of PCA algorithm for dimensionality reduction of images is *N^3* where *N* is the resolution of the vector. This means that it takes a cubic volume of time to process an image vector. As against this, the DCT is an algorithm, which can be effectively implemented using the Fast Fourier Transform (FFT), which is a fast algorithm with a very economic complexity. Because of this, the dimensionality reduction of data with DCT occurs in just a few seconds, which could take several minutes or hours for PCA. Hence the use of DCT has proved useful in compressing the data without losing significant information. Also, using PCA, recovery of images is difficult and not very accurate. However, the use of DCT enabled the obtainment of the final images in a *human readable form.* Hence the central images can be directly seen as normal images at the end.
3. Easy Recognition: Since the final central images obtained are normal images, it is possible to directly apply any recognition algorithm on these images, so that new images can be directly classified using these central images.

CHAPTER 5

THE USER MANUAL

5.0 Introduction

In this section a successful scenario from the execution to the results will be illustrated. The AT&T database was used for testing.

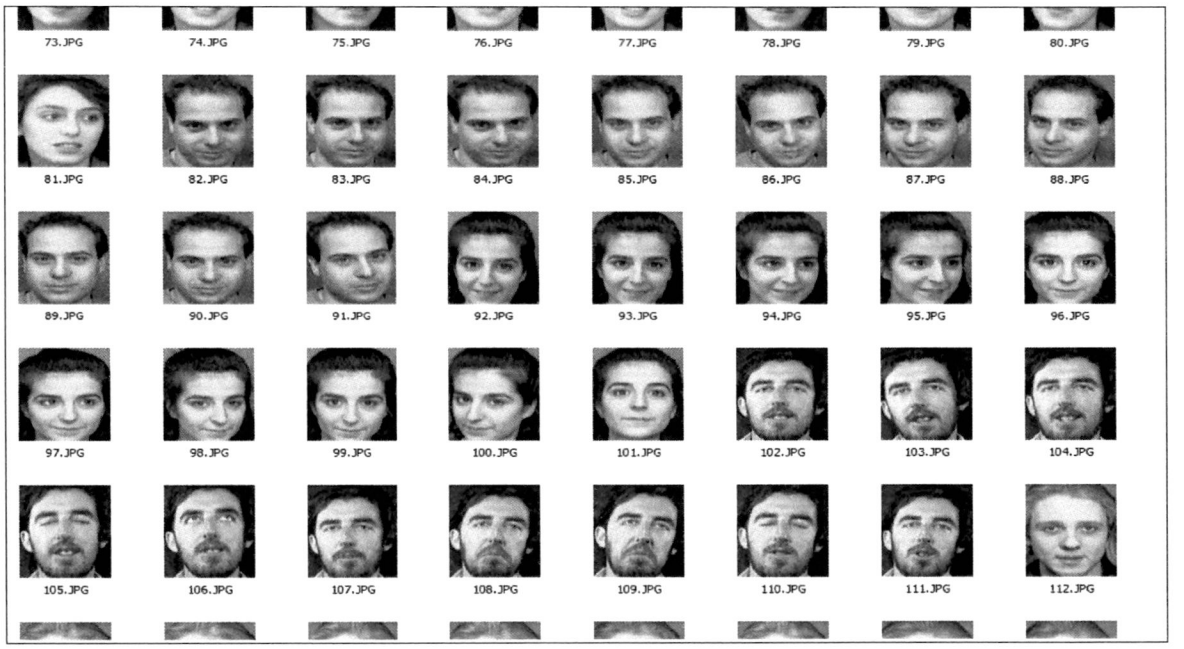


Fig. 16. AT&T Database Snapshot

5.1 Software Requirement

The project does not require any hardware installation. However, Matlab7 should be installed on a personal computer with any of the following operating systems: Windows 98, Windows 2000, Windows ME, Windows XP, Windows NT.

5.2 Getting Started

The user should first copy the files into the Matlab7 work folder. The files include the faceclustering.m , savefaces.m ,somtoolbox and the face databases. After running the Matlab7 command window, the user types the name of the function "faceclustering" then the GUI will pop up.

5.3 The Main Screen

The Main Screen has been designed to show the user a step-by-step approach from the beginning to the end displaying what is happening for educational purposes. It contains three text areas in which the user enters the full face database path, the extensions of the images of the targeted database, the total number of faces to be clustered, and the total clusters required. When the project is executed the opening screen will look like the following snapshot:

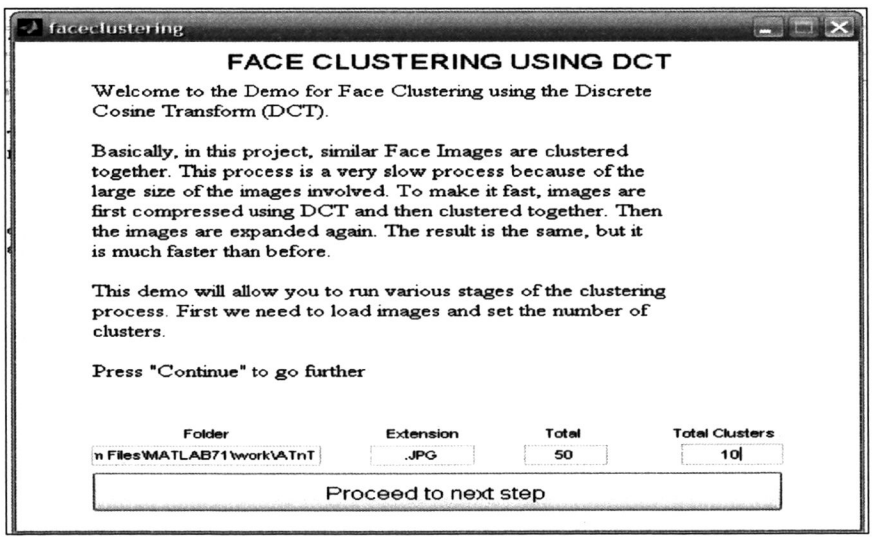


Fig.17. The Interface After Execution

5.4 Progress Screen

This screen tells the user that the faces have been successfully loaded. It includes one button that the user clicks to get the DCT of the images. Then a similar one to pops up and indicates that they have been successfully obtained.

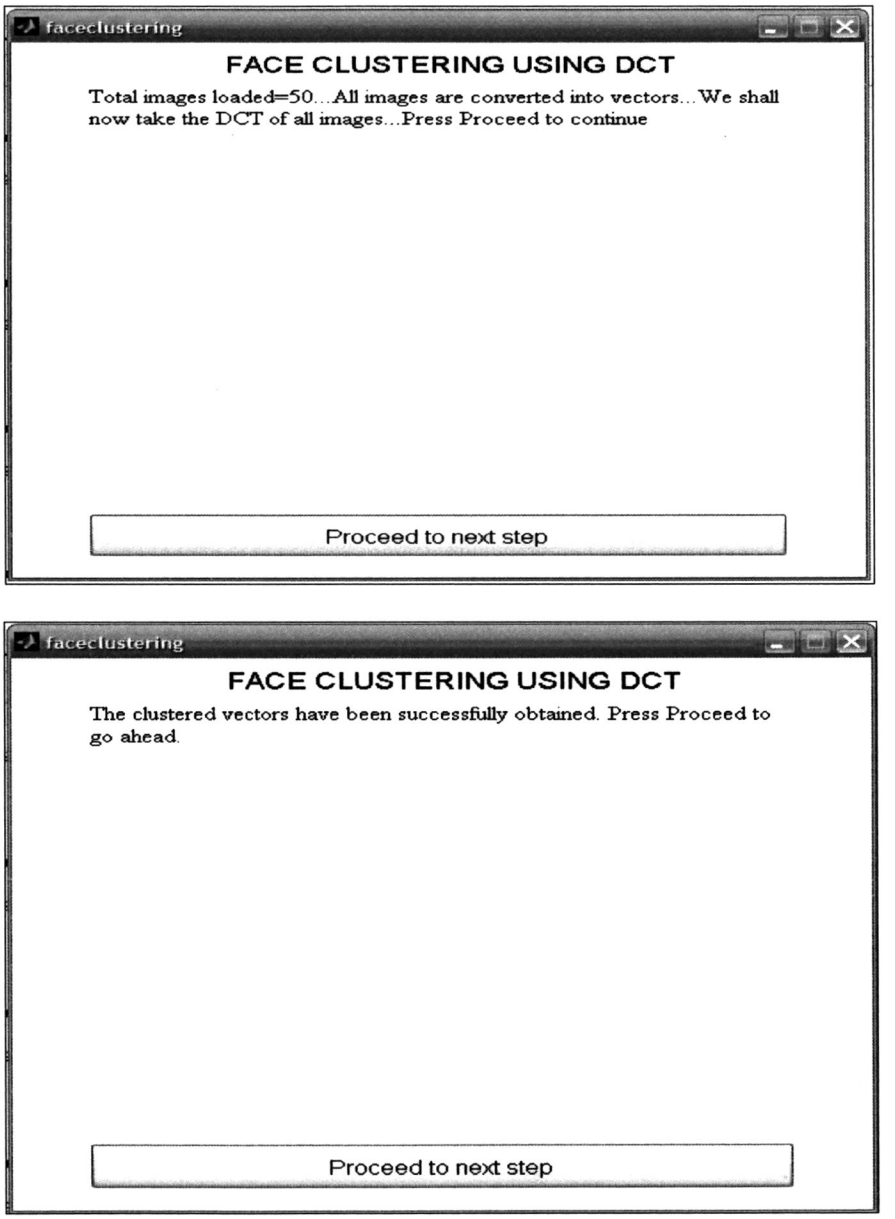


Fig. 18. Progress Screens

5.5 Showing The Clusters Screen

At this point the clusters and their central image have been created in this screen. The user enters the number of the cluster he wishes to display its central image.

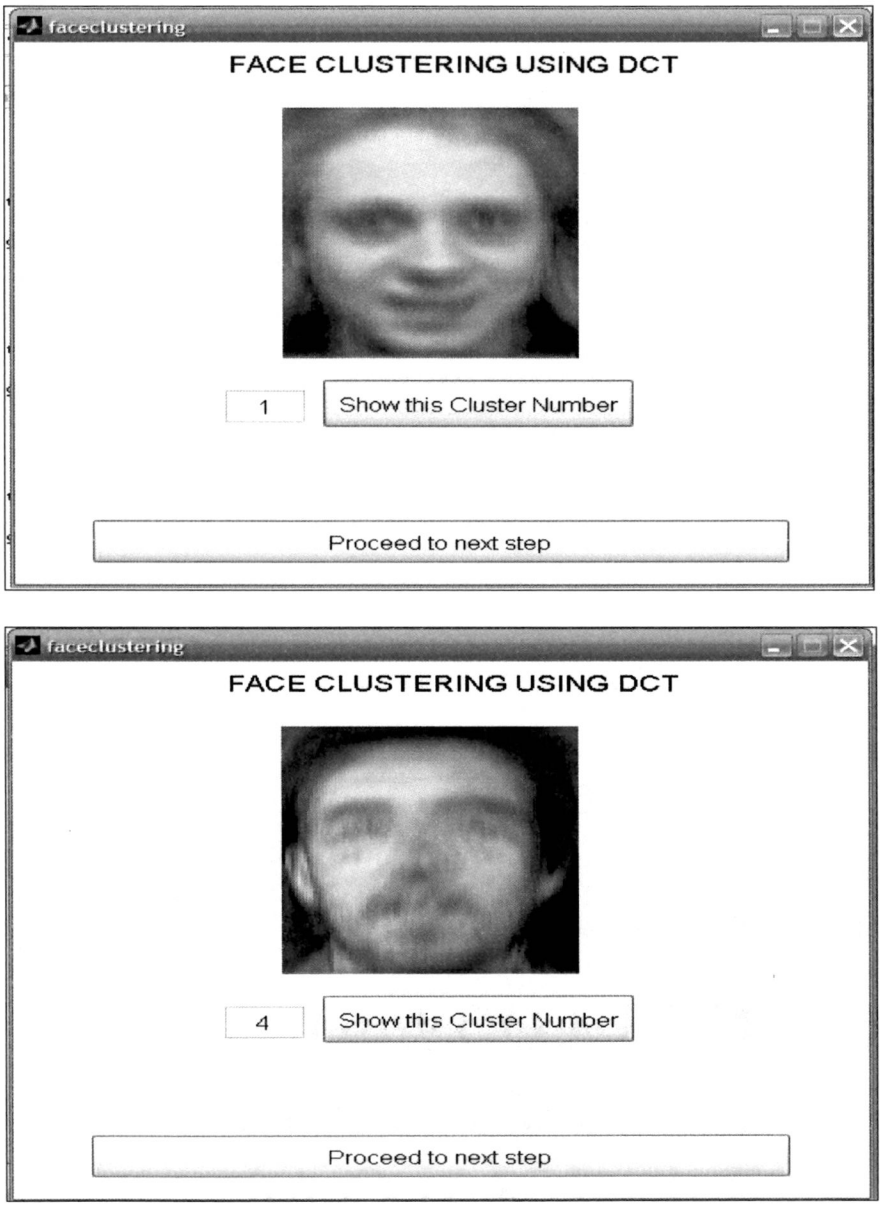


Fig.19. The Cluster Selection

5.6 The Testing Screen

Now that the clusters have been created the user can enter the face number he wishes to determine as belonging to a particular cluster.

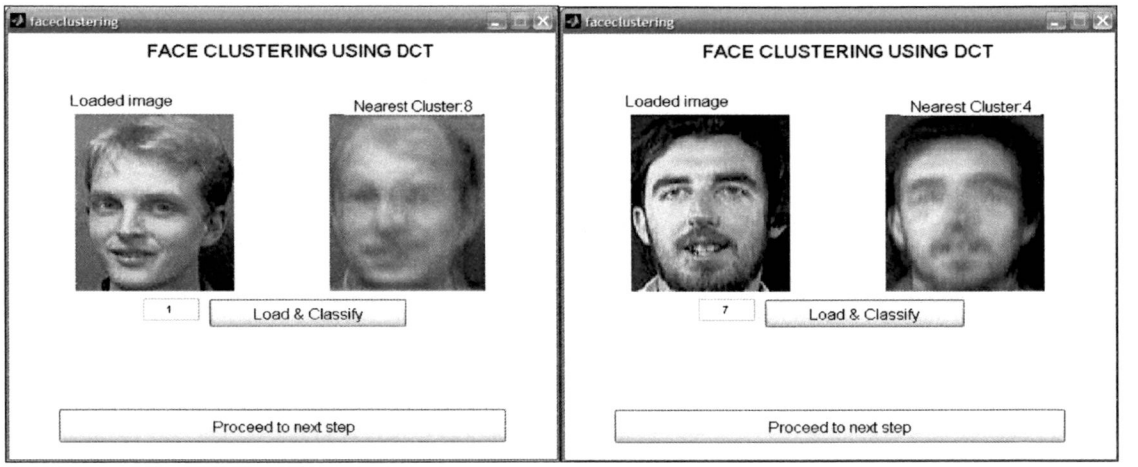


Fig.20. The Testing Screen

5.7 The Saving Screen

This screen includes a text area where the user enters the location in which he wishes the clusters to be saved.

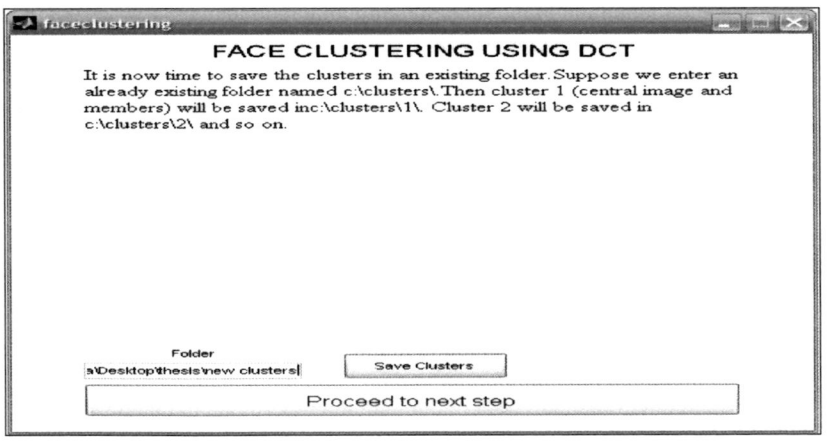


Fig. 21. The Saving Screen

5.8 The Results

The clusters are saved in the desired location and each has a central image.

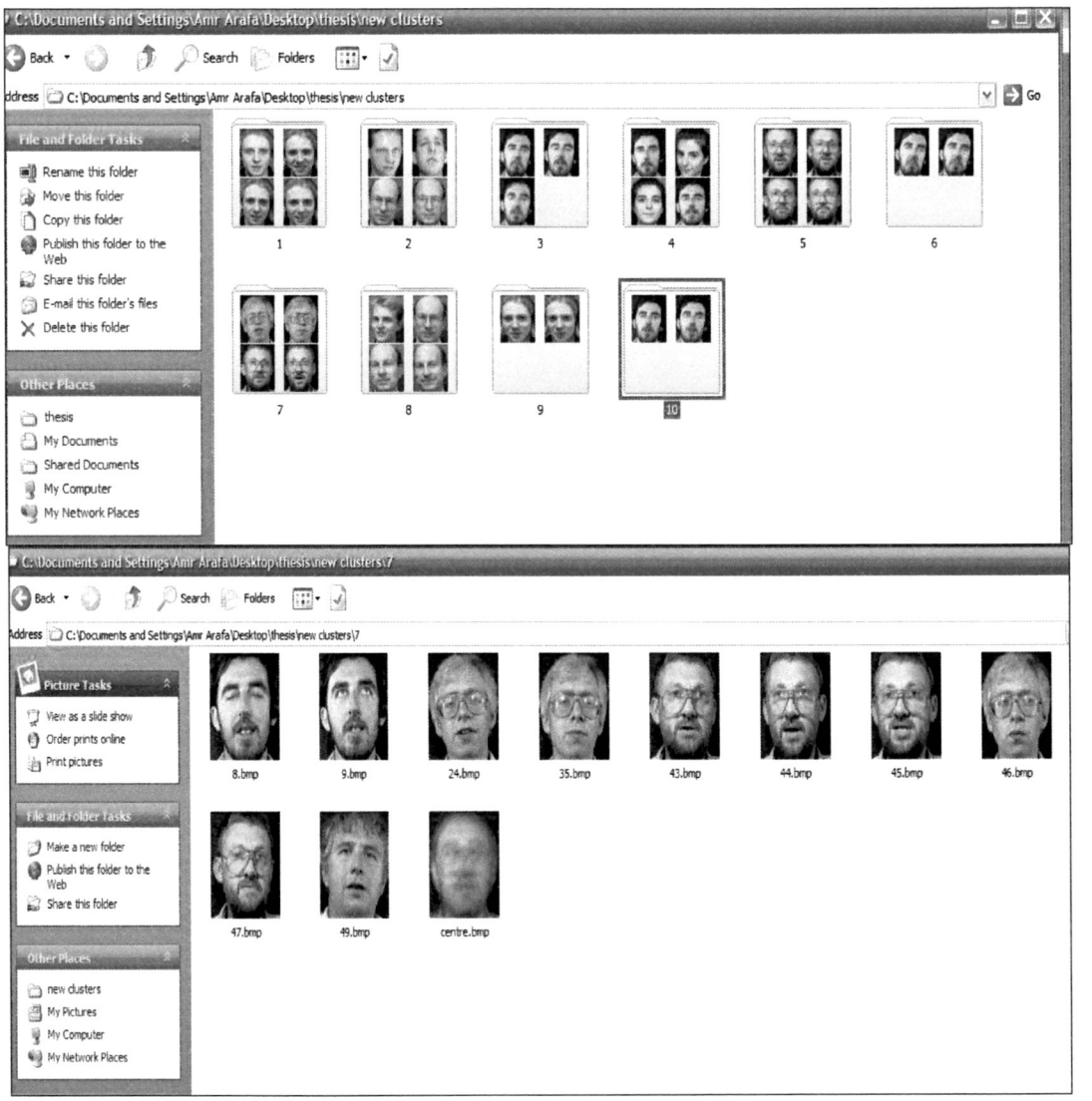


Fig. 22. The Result Cluster Folders

5.9 Conclusion

Face recognition has been introduced as an important method of biometric authentication, then the mostly used algorithms have been discussed and compared and the results have been shown. The Haar face detection algorithm was also tested over a number of well-known face databases and the results were shown. This state of the art clustering system was discussed along with a discussion on other existing clustering methods that this system overpowered.

APPENDIX A

FACE DETECTION SOFTWARE VALIDATION

In order to judge a face detection software, experimentation with this software in different face detection databases must take place. In this document, the following databases are discussed: (The combined MIT/CMU data set, CMU dataset II, AT&T face databases, Japanese female facial expression databases and Yale face database) and the Haar face detection algorithm test results will be presented on them.

Introduction

Because of its non-rigidity and complex three-dimensional (3-D) structure, the appearance of a face is affected by a large number of factors including identity, face pose, illumination, facial expression, age, occlusion, and facial hair. The development of algorithms robust to these variations requires databases of sufficient size that include carefully controlled variations of these factors. Furthermore, it is necessary to use common databases in order to comparatively evaluate algorithms. Collecting a high quality database is a resource-intensive task, but the availability of public face databases is important for the advancement of the field [6]. Here, four publicly available databases for face recognition, face detection, and facial expression analysis are reviewed, (these are the combined MIT/CMU data set, CMU dataset II, AT&T face databases, Japanese

female facial expression databases and Yale face database) And then the results are discussed and the comparison Graphs is shown.

The Databases

The combined MIT/CMU data set

Includes Total of 180 images.

Includes Total of 730 faces.

More info

- A number of images are included that do not contain any faces in order to test tolerance to clutter.
- 210 Faces are at angles of more than 10◦ from upright.

Example

Fig. 23. Example of the MIT/CMU database

CMU Test Set II

Includes Total of 208 images.

Includes Total of 441 faces.

More info

- 347 faces are in profile view.
- The images were all collected from the Internet.

Example

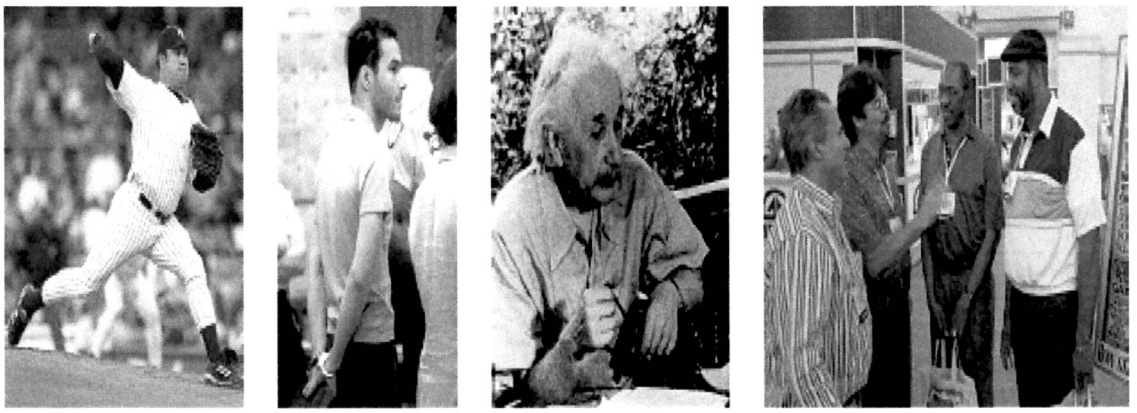


Fig. 24. Example of the CMU II database

AT&T database

Includes Total of 400 images.

Includes Total of 400 faces.

More info

- 150 face are profile (5 per person)
- 40 persons (10 images each)

Example

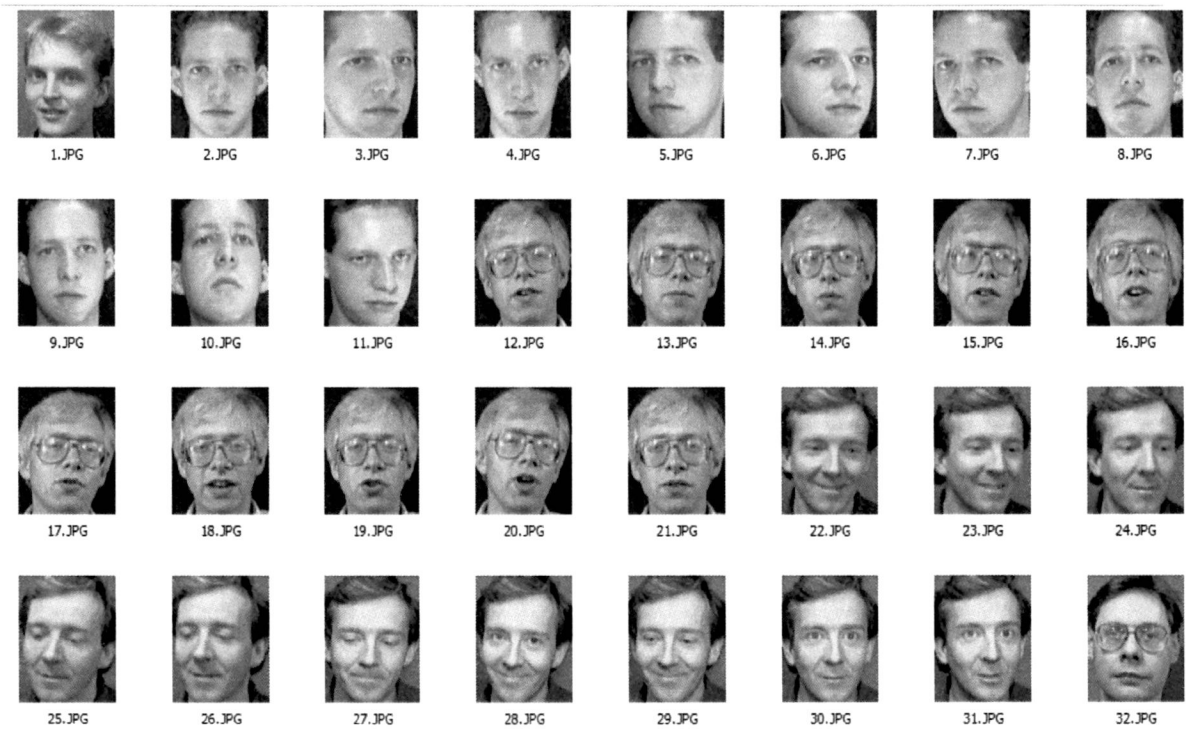


Fig. 25. Example of the AT&T database

Japanese Female Facial Expression (JAFFE) Database

Includes Total of 213 images.

Includes Total of 213 faces.

More info

Number of female models 10.

Number of expressions 7.

Example

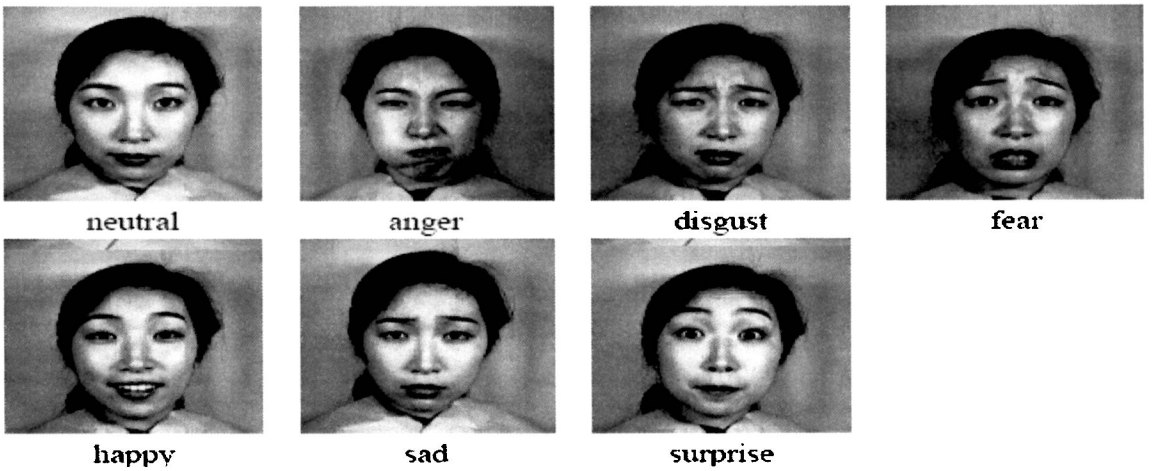


Fig. 26. Example of the JAFFE database

Yale database

Includes Total of 166 images.

Includes Total of 166 faces.

More info

Number of persons 10.

Number of poses 9.

Number of illumination 64.

Example

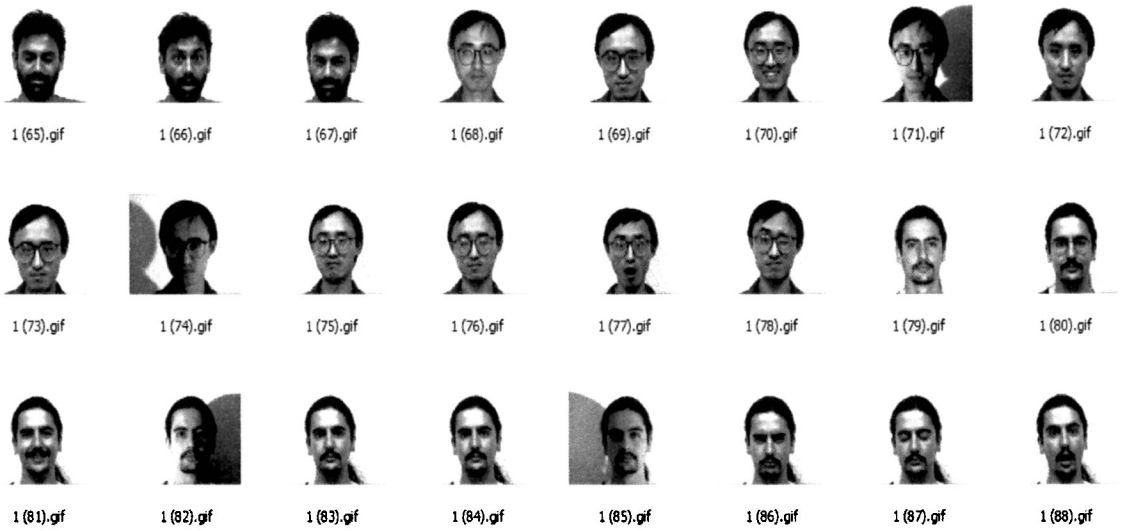


Fig. 27. Example of the YALE database

The Results

The results are calculated on both the non-detected faces and the miss detection, so that every time the algorithm does not detect a face, one is incremented to the non-detected total. Furthermore, every time the algorithm detects an area that is not a face, one is incremented to the mistakes total.

The following section shows both the non-detected total and the mistakes total, showing the error ratio based on each database's original size, followed by a graph of the results to allow for easier visualization.

MIT/CMU

300 non-detected.

19 mistakes (non face detection).

Total mistakes 319.

Error rate = (319 x 100) / 730 = 43.698%

Example of the errors

Fig. 28. Example of both non-detected error and mistakes error in the MIT/CMU database

CMU II

324 non-detected.

8 mistakes (non face detection).

Total mistakes 332.

Error rate = (343 x 100) / 441 = 77.777%

Example of the errors

Fig. 29. Example of both non-detected error and mistakes error in the CMU II database

AT&T

98 non-detected.

1 mistake (non face detection).

Total mistakes 99.

Error rate = (99 x 100) / 400 = 24.75%

Example of the errors

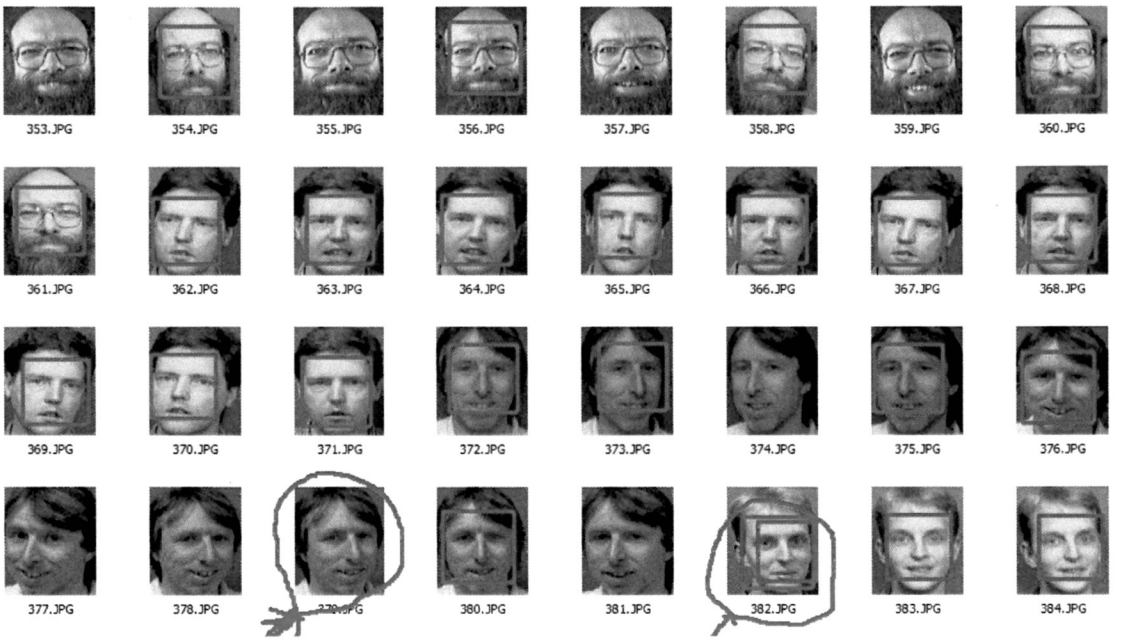


Fig. 30. Example of both non-detected error and mistakes error in the AT&T database

JAFFE

0 non-detected.

1 mistake (non face detection).

Total mistakes 1.

Ratio of the bad results = (1 x 100) / 213 = 0.469% faults

The only error

Fig. 31. The only "mistakes" error in the JAFFE database

YALE

0 non-detected.

0 mistakes (non face detection).

Total mistakes 0.

Ratio of the bad results = (0 x 100) / 166 = 0.0 % faults

Example of the results

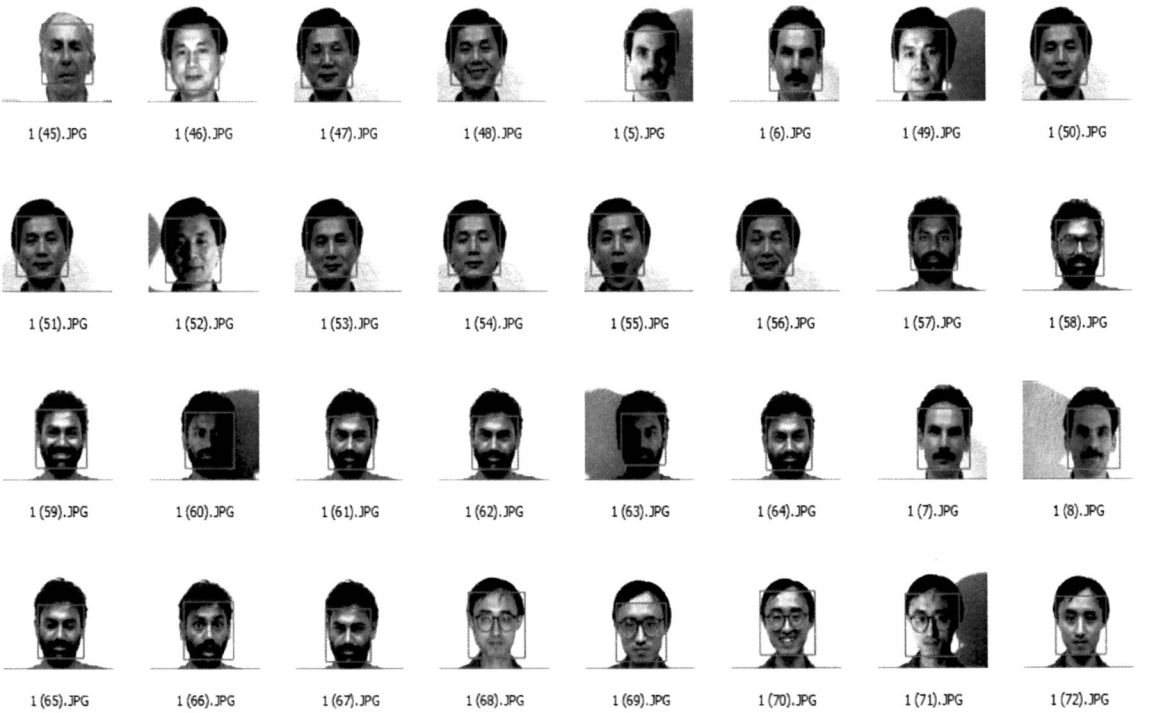


Fig. 32. Example of the YALE database results that had no errors.

Table 3. The Comparison on Different Face Databases

	MIT/CMU	CMU II	AT&T	JAFFE	YALE
Description	faces with different orientations and resolution	faces with different orientations (profiles mostly)	faces with different facial hair/accessories	faces with different expressions	faces with different expressions and lighting
Number of faces	730	441	400	213	166
Total Error rate	43.698%	77.777%	24.75%	0.469%	0%
Non-detected errors	300	324	98	0	0
Mistakes	19	8	1	1	0

The Results Graph

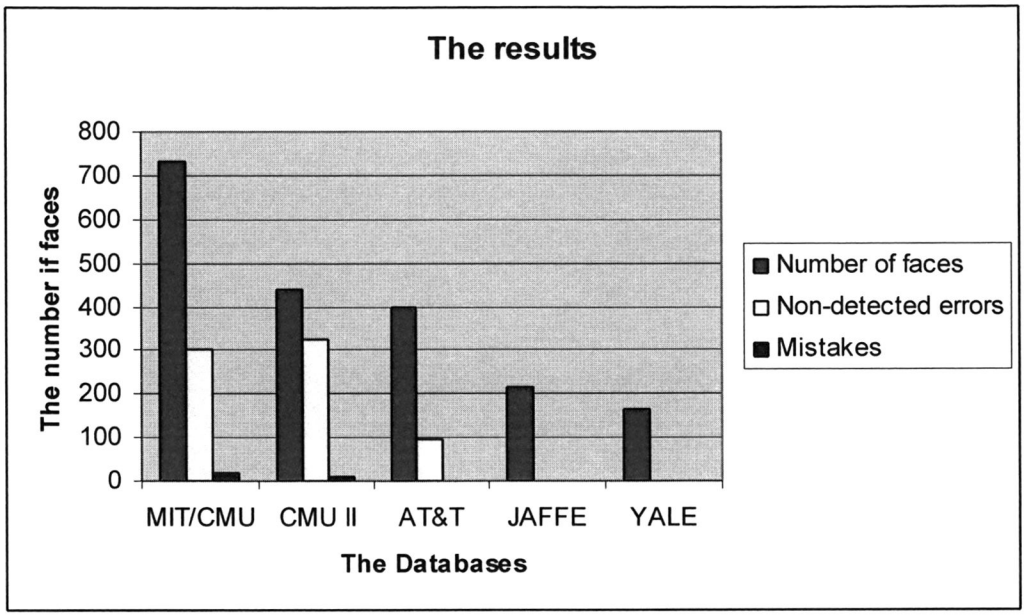


Fig. 33. The results shown in the graph

The ratio

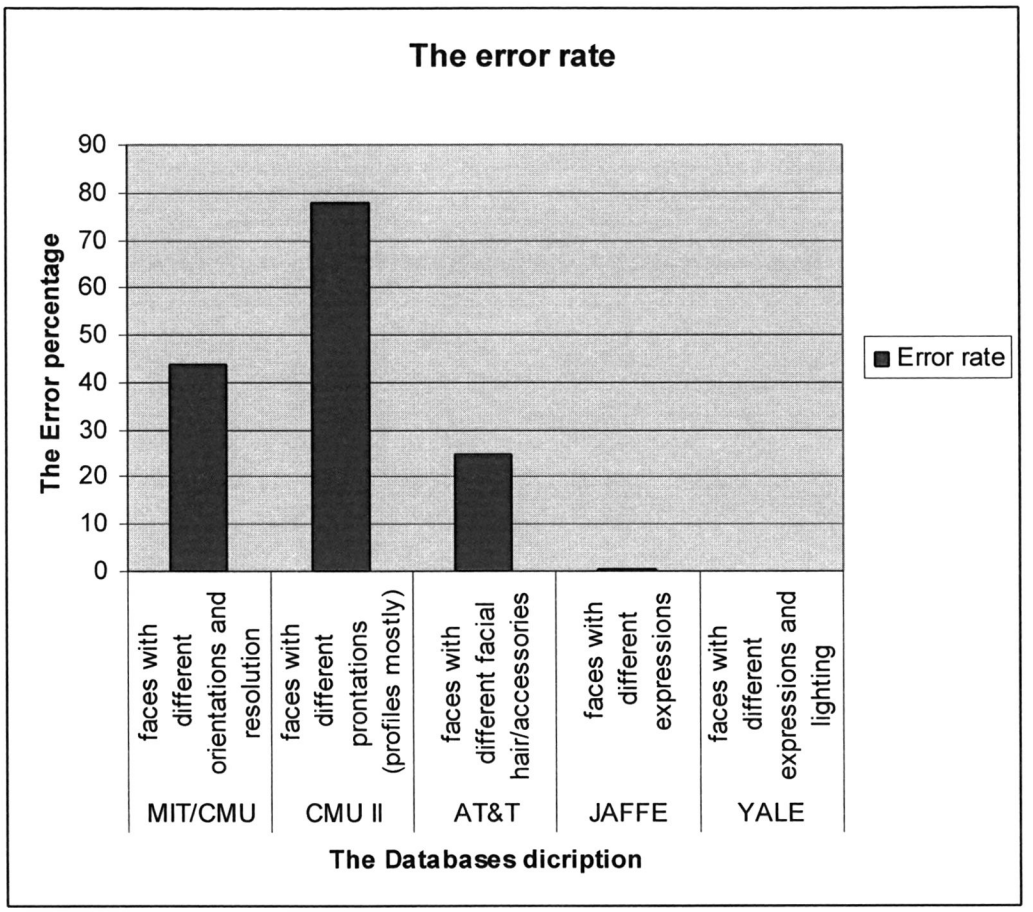


Fig. 34. The Error ratio with the description of the databases.

Conclusion

The detection showed very good results on the JAFFE and YALE databases, by looking at these databases info we find that they have the hardest illumination/expressions tricks among all the other databases we tested (illumination/expressions didn't really matter). On the other hand the worst results were noticed in the CMU II, the one with most profile images among the other databases we

tested, the MIT/CMU, the one with smaller size images and different orientations (upside down sometimes), and in the AT&T with a the facial hair and different poses.

APPENDIX B

(MATLAB CODES)

Face clustering

```
function varargout = faceclustering(varargin)
% FACECLUSTERING M-file for faceclustering.fig
%      FACECLUSTERING, by itself, creates a new FACECLUSTERING or raises the
existing
%      singleton*.
%
%      H = FACECLUSTERING returns the handle to a new FACECLUSTERING or the
handle to
%      the existing singleton*.
%
%      FACECLUSTERING('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in FACECLUSTERING.M with the given input
arguments.
%
%      FACECLUSTERING('Property','Value',...) creates a new FACECLUSTERING or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before faceclustering_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to faceclustering_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help faceclustering

% Last Modified by GUIDE v2.5 21-Oct-2006 07:45:32

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                'gui_OpeningFcn', @faceclustering_OpeningFcn, ...
                'gui_OutputFcn',  @faceclustering_OutputFcn, ...
                'gui_LayoutFcn',  [] , ...
                'gui_Callback',   []);
if nargin & isstr(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 faceclustering is made visible.
function faceclustering_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 faceclustering (see VARARGIN)

% Choose default command line output for faceclustering
handles.output = hObject;
handles.ftag=1;

% Update handles structure
guidata(hObject, handles);

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

 % --- Outputs from this function are returned to the command line.
function varargout = faceclustering_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 cmdNext.
function cmdNext_Callback(hObject, eventdata, handles)
% hObject    handle to cmdNext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if handles.ftag==1
   disp(handles.ftag);
   foldername=get(handles.editfolder,'String');
   extension=get(handles.editext,'String');
   totalimages=str2num(get(handles.edittotal,'String'));
   totalclusters=str2num(get(handles.editclusnum,'String'));
   handles.totalclusters=totalclusters;
   handles.totalimages=totalimages;
   [imlist, Lx, Ly]=loadimvecsname(foldername,extension, totalimages, handles);
   set(handles.editfolder,'Visible','off');
   set(handles.edittotal,'Visible','off');
   set(handles.editclusnum,'Visible','off');
   set(handles.editext,'Visible','off');
   set(handles.lblfolder,'Visible','off');
   set(handles.lbltotal,'Visible','off');
   set(handles.lblclusnum,'Visible','off');
   set(handles.lblclusnum,'Visible','off');
   set(handles.lblext,'Visible','off');
   handles.ftag=2;
   handles.imlist=imlist;
   handles.Lx=Lx;
   handles.Ly=Ly;
   guidata(hObject, handles); return;
end
if handles.ftag==2
   disp(handles.ftag);
   dctlist=dctstep(handles);
   handles.ftag=3;
   handles.d=dctlist(:,1:end-8000);
   set(handles.board,'String','DCT vectors obtained. Also the DCT vectors are downsized
for faster processing.');
   str=get(handles.board,'String');
   set(handles.board,'String',strcat(str,'Press Proceed to perform clustering'));
   guidata(hObject, handles); return;
end
if handles.ftag==3
   str='The program would now begin clustering of the images.';
   str=strcat(str,'In this step, similar images are grouped into a given number of clusters.');
```

```
   str=strcat(str,'PLEASE WAIT FOR THE NEXT MESSAGE...');
   set(handles.board,'String',str);
   %  CHANGE NUMBER OF CLUSTERS HERE
   [c, memb]=som_kmeans('seq',handles.d,handles.totalclusters);
   str='The DCT clusters have been obtained.';
   str=strcat(str,'We now take the inverse DCT of the vectors to get the images back.');
   str=strcat(str,'PLEASE WAIT FOR THE NEXT MESSAGE...');
   z=size(handles.imlist,2);
   c(:,end:z)=0;
   for i=1:handles.totalclusters
      ctr(i,:)=idct(c(i,:));
   end
   str='The clustered vectors have been successfully obtained. Press Proceed to go
ahead.';
   set(handles.board,'String',str);
   handles.ctr=ctr;
   handles.memb=memb;
   handles.ftag=4;
   guidata(hObject, handles); return;
end
if handles.ftag==4
   disp(handles.ftag);
   set(handles.board,'Visible','off');
   set(handles.axesc,'Visible','on');
   set(handles.editcnum,'Visible','on');
   set(handles.cmdShowCluster,'Visible','on');
   handles.ftag=5;
   guidata(hObject, handles); return;
end
if handles.ftag==5
   disp(handles.ftag);
   set(handles.board,'Visible','on');
   set(handles.axesc,'Visible','off');
   set(handles.editcnum,'Visible','off');
   set(handles.cmdShowCluster,'Visible','off');
   str='The various clusters were just shown. We can now see the processed images';
   str=strcat(str,' and their Nearest Clusters. For this, press Proceed.');
   set(handles.board,'String',str);
   handles.ftag=6;
   guidata(hObject, handles); return;
end
if handles.ftag==6
   disp(handles.ftag);
```

```
   cla;
   set(handles.board,'Visible','off');
   set(faceclustering,'CurrentAxes',handles.axes1);
   set(handles.axesc,'Visible','off');
   set(handles.editcnum,'Visible','off');
   set(handles.cmdShowCluster,'Visible','off');
   set(handles.axes1,'Visible','on');
   set(handles.axes2,'Visible','on');
   set(handles.editinum,'Visible','on');
   set(handles.cmdClass,'Visible','on');
   set(handles.imlabel,'Visible','on');
   set(handles.cluslabel,'Visible','on');
   handles.ftag=7;
   guidata(hObject, handles); return;
end
if handles.ftag==7
   set(handles.axes1,'Visible','off');
   set(handles.axes2,'Visible','off');
   set(handles.editinum,'Visible','off');
   set(handles.cmdClass,'Visible','off');
   set(handles.imlabel,'Visible','off');
   set(handles.cluslabel,'Visible','off');
   set(handles.editfolder,'String','');
   set(handles.editfolder,'Visible','on');
   set(handles.cmdSave,'Visible','on');
   set(handles.board,'Visible','on');
   set(handles.lblfolder,'Visible','on');
   str='It is now time to save the clusters in an existing folder. ';
   str=strcat(str,'Suppose we enter an already existing folder named c:\clusters\. ');
   str=strcat(str,'Then cluster 1 (central image and members) will be saved in ');
   str=strcat(str,'c:\clusters\1\. Cluster 2 will be saved in c:\clusters\2\ and so on.');
   set(handles.board,'String',str);
end

function [imlist, Lx, Ly]=loadimvecs(impath, ext, totalnum, handles)
board=handles.board;
compressionfactor=1;
for i=1:totalnum
   %CHANGE THE TOTAL NUMBER OF FACES (PRESENTLY 391) IF YOU ARE
USING SOME OTHER LIBRARY
   %CHANGE THE PATH 'd:\matlab7\attfaces\' TO THE DIRECTORY WHERE THE
   %NUMBERED FACES ARE STORED. THE IMAGES SHOULD BE NUMBERED
AS 1.PGM AND
```

```
  %ONWARDS AS 4.PGM. 5.PGM AND SO ON. SO IF ANY OTHER FORMAT IS
USED, THE
  %CORRESPONDING CHANGES MUST BE MADE IN THE LINE BELOW.
  temp=strcat(impath,'\',num2str(i),ext);
  im=imread(temp);
  ztemp=size(im);
  if(length(ztemp)~=2)
    im=rgb2gray(im);
  end
  %RESIZE THE IMAGE TO HALF SIZE BY I.E. FACTOR 2 (CHANGE 2 TO A
HIGHER FACTOR IF THE IMAGE
  %IS TOO BIG).IF THE IMAGES ARE SMALL ENOUGH, SIMPLY MAKE
im2=im1; IN
  %THE LINES BELOW.
  Lx=uint8(size(im,1)/compressionfactor);
  Ly=uint8(size(im,2)/compressionfactor);
  im2=imresize(im,double(uint8(size(im)/compressionfactor)));
  %CONVERT THE IMAGES INTO A SET OF LONG VECTORS
  imlist(i,:)=im2(:);
  %DISPLAY THE NUMBER OF IMAGES LOADED SO FAR...DELETE THIS "IF"
PART LINES IF NOT
  %REQUIRED.
  if mod(i,20)==0
    set(board,'String',strcat('Total images loaded=',num2str(i)));
  end
end
set(board,'String',strcat('Total images loaded=',num2str(i)));
str=get(board,'String');
set(board,'String',strcat(str,'...All images are converted into vectors'));
str=get(board,'String');
set(board,'String',strcat(str,'...We shall now take the DCT of all images'));
str=get(board,'String');
set(board,'String',strcat(str,'...Press Proceed to continue'));
%THE FUNCTION kmeans NEEDS THE VECTORS TRANSPOSED...
imlist=double(imlist);

function [imlist, Lx, Ly]=loadimvecsname(impath, ext, totalnum, handles)
board=handles.board;
compressionfactor=1;
g=GetFilesWithExtension(impath,ext);
for i=1:totalnum
  %CHANGE THE TOTAL NUMBER OF FACES (PRESENTLY 391) IF YOU ARE
USING SOME OTHER LIBRARY
```

```
   %CHANGE THE PATH 'd:\matlab7\attfaces\' TO THE DIRECTORY WHERE THE
   %NUMBERED FACES ARE STORED. THE IMAGES SHOULD BE NUMBERED
AS 1.PGM AND
   %ONWARDS AS 4.PGM, 5.PGM AND SO ON. SO IF ANY OTHER FORMAT IS
USED, THE
   %CORRESPONDING CHANGES MUST BE MADE IN THE LINE BELOW.
%    temp=strcat(impath,'\',num2str(i),ext);
   im=imread(g{i});
   ztemp=size(im);
   if(length(ztemp)~=2)
      im=rgb2gray(im);
   end
   %RESIZE THE IMAGE TO HALF SIZE BY I.E. FACTOR 2 (CHANGE 2 TO A
HIGHER FACTOR IF THE IMAGE
   %IS TOO BIG).IF THE IMAGES ARE SMALL ENOUGH, SIMPLY MAKE
im2=im1; IN
   %THE LINES BELOW.
   Lx=uint8(size(im,1)/compressionfactor);
   Ly=uint8(size(im,2)/compressionfactor);
   im2=imresize(im,double(uint8(size(im)/compressionfactor)));
   %CONVERT THE IMAGES INTO A SET OF LONG VECTORS
   imlist(i,:)=im2(:);
   %DISPLAY THE NUMBER OF IMAGES LOADED SO FAR...DELETE THIS "IF"
PART LINES IF NOT
   %REQUIRED.
   if mod(i,20)==0
      set(board,'String',strcat('Total images loaded=',num2str(i)));
   end
end
set(board,'String',strcat('Total images loaded=',num2str(i)));
str=get(board,'String');
set(board,'String',strcat(str,'...All images are converted into vectors'));
str=get(board,'String');
set(board,'String',strcat(str,'...We shall now take the DCT of all images'));
str=get(board,'String');
set(board,'String',strcat(str,'...Press Proceed to continue'));
%THE FUNCTION kmeans NEEDS THE VECTORS TRANSPOSED...
imlist=double(imlist);

function [imlist, Lx, Ly]=loadimvecs2(impath, ext, totalnum, handles)
board=handles.board;
compressionfactor=1;
direc=dir(impath);
```

```
files=direc.name(3:end);
for i=1:totalnum
  %CHANGE THE TOTAL NUMBER OF FACES (PRESENTLY 391) IF YOU ARE
USING SOME OTHER LIBRARY
  %CHANGE THE PATH 'd:\matlab7\attfaces\' TO THE DIRECTORY WHERE THE
  %NUMBERED FACES ARE STORED. THE IMAGES SHOULD BE NUMBERED
AS 1.PGM AND
  %ONWARDS AS 4.PGM, 5.PGM AND SO ON. SO IF ANY OTHER FORMAT IS
USED, THE
  %CORRESPONDING CHANGES MUST BE MADE IN THE LINE BELOW.
  temp=strcat(impath,'\',files(i));
  im=imread(temp);
  ztemp=size(im);
  if(length(ztemp)~=2)
    im=rgb2gray(im);
  end
  %RESIZE THE IMAGE TO HALF SIZE BY I.E. FACTOR 2 (CHANGE 2 TO A
HIGHER FACTOR IF THE IMAGE
  %IS TOO BIG).IF THE IMAGES ARE SMALL ENOUGH, SIMPLY MAKE
im2=im1; IN
  %THE LINES BELOW.
  Lx=uint8(size(im,1)/compressionfactor);
  Ly=uint8(size(im,2)/compressionfactor);
  im2=imresize(im,double(uint8(size(im)/compressionfactor)));
  %CONVERT THE IMAGES INTO A SET OF LONG VECTORS
  imlist(i,:)=im2(:);
  %DISPLAY THE NUMBER OF IMAGES LOADED SO FAR...DELETE THIS "IF"
PART LINES IF NOT
  %REQUIRED.
  if mod(i,20)==0
    set(board,'String',strcat('Total images loaded=',num2str(i)));
  end
end
set(board,'String',strcat('Total images loaded=',num2str(i)));
str=get(board,'String');
set(board,'String',strcat(str,'...All images are converted into vectors'));
str=get(board,'String');
set(board,'String',strcat(str,'...We shall now take the DCT of all images'));
str=get(board,'String');
set(board,'String',strcat(str,'...Press Proceed to continue'));
%THE FUNCTION kmeans NEEDS THE VECTORS TRANSPOSED...
imlist=double(imlist);
```

```
function dctlist=dctstep(handles)
imlist=handles.imlist;
for i=1:handles.totalimages
   dctlist(i,:)=dct(imlist(i,:));
end


% --- Executes during object creation, after setting all properties.
function editcnum_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editcnum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function editcnum_Callback(hObject, eventdata, handles)
% hObject    handle to editcnum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editcnum as text
%        str2double(get(hObject,'String')) returns contents of editcnum as a double


% --- Executes on button press in cmdShowCluster.
function cmdShowCluster_Callback(hObject, eventdata, handles)
% hObject    handle to cmdShowCluster (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
str=get(handles.editcnum,'String');
i=str2num(str);
x=uint8(reshape(handles.ctr(i,:),handles.Lx,handles.Ly));
set(faceclustering,'CurrentAxes',handles.axesc);
imshow(x);
handles.ftag=5;
```

```
guidata(hObject, handles); return;


% --- Executes during object creation, after setting all properties.
function editinum_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editinum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function editinum_Callback(hObject, eventdata, handles)
% hObject    handle to editinum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editinum as text
%        str2double(get(hObject,'String')) returns contents of editinum as a double


% --- Executes on button press in cmdClass.
function cmdClass_Callback(hObject, eventdata, handles)
% hObject    handle to cmdClass (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
str=get(handles.editinum,'String');
i=str2num(str);
x=uint8(reshape(handles.imlist(i,:),handles.Lx,handles.Ly));
set(faceclustering,'CurrentAxes',handles.axes1);
imshow(x);
[ctrim, ctrnum]=NearestCluster(i,handles);
x=reshape(uint8(ctrim),handles.Lx,handles.Ly);
set(faceclustering,'CurrentAxes',handles.axes2);
imshow(x);
str=strcat('Nearest Cluster:',num2str(ctrnum));
```

```
set(handles.cluslabel,'String',str);
guidata(hObject, handles); return;

function [ctrim, ctrnum]=NearestCluster(i,handles)
imvector=double(handles.imlist(i,:));
ctrlist=double(handles.ctr);
for i=1:size(ctrlist,1)
   dvec(i,:)=abs(ctrlist(i,:)-imvector);
   d(i)=sum(dvec(i,:));
end
f=find(d==min(d(:)));
ctrim=uint8(ctrlist(f,:));
ctrnum=f;

function StoreClusters(parentdir, handles)
%THE SYNTAX OF THE FUNCTION IS:
%StoreClusters(parentdir, ctrlist, imlist, lx,ly, memb);
%HERE THE VARIABLES MEAN EXACTLY THE SAME AS THEY MEAN IN THE
PREVIOUS
%FUNCTIONS.
%FOR EXAMPLE, IF parentdir='c:\clusters' AND THERE ARE 5 CLUSTERS, THEN
THE PROGRAM CREATES
%DIRECTORIES: 'c:\clusters\1' to 'c:\clusters\5'.
%IN EACH DIRECTORY, THE CENTRAL IMAGE IS STORED AS 'centre.bmp'.
ALSO, THE
%MEMBER IMAGES WITHIN imlist TOO ARE STORED IN THE
CORRESPONDING
%DIRECTORIES. HENCE EACH DIRECTORY NOW CONTAINS A CLUSTER
CENTRE AND ALL
%CLUSTER MEMBERS.
ctrlist=handles.ctr;
imlist=handles.imlist;
lx=handles.Lx;
ly=handles.Ly;
memb=handles.memb;
for i=1:size(ctrlist,1)
   mkdir(parentdir,num2str(i));
   ctrtemp=uint8(reshape(ctrlist(i,:),lx,ly));
   temp=strcat(parentdir,'\',num2str(i),'\centre.bmp'); disp(temp);
   imwrite(ctrtemp,temp);
   f=find(memb==i);
   for j=1:length(f)
      temp=strcat(parentdir,'\',num2str(i),'\',num2str(f(j)),'.bmp'); disp(temp);
```

```
      imtemp=uint8(reshape(imlist(f(j),:),lx,ly));
      imwrite(imtemp,temp);
   end
end


pause(3);
disp(strcat('All clusters have been stored in numbered directories in:',parentdir));


% --- Executes during object creation, after setting all properties.
function editfolder_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editfolder (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function editfolder_Callback(hObject, eventdata, handles)
% hObject    handle to editfolder (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editfolder as text
%        str2double(get(hObject,'String')) returns contents of editfolder as a double


% --- Executes during object creation, after setting all properties.
function edittotal_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edittotal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edittotal_Callback(hObject, eventdata, handles)
% hObject    handle to edittotal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edittotal as text
%        str2double(get(hObject,'String')) returns contents of edittotal as a double


% --- Executes during object creation, after setting all properties.
function editext_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function editext_Callback(hObject, eventdata, handles)
% hObject    handle to editext (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editext as text
%        str2double(get(hObject,'String')) returns contents of editext as a double


% --- Executes during object creation, after setting all properties.
function editclusnum_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to editclusnum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
  set(hObject,'BackgroundColor','white');
else
  set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function editclusnum_Callback(hObject, eventdata, handles)
% hObject    handle to editclusnum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editclusnum as text
%        str2double(get(hObject,'String')) returns contents of editclusnum as a double


% --- Executes during object creation, after setting all properties.
function editnew_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editnew (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
  set(hObject,'BackgroundColor','white');
else
  set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function editnew_Callback(hObject, eventdata, handles)
% hObject    handle to editnew (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of editnew as text
```

```
%       str2double(get(hObject,'String')) returns contents of editnew as a double


% --- Executes on button press in cmdSave.
function cmdSave_Callback(hObject, eventdata, handles)
% hObject    handle to cmdSave (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
StoreClusters(get(handles.editfolder,'String'),handles);


function flist=GetFilesWithExtension(path, ext)
direc=struct2cell(dir(path));
num=size(direc,2);
count=0;
for i=1:num
   temp=direc(i);
   if strcmp(class(temp{1}),'char');
      str=strcat(path,'\',direc(i));
      disp(class(str{1}));
      [j, e, p]=FileNameAttributes(str{1});
      if strcmp(e,ext)
         count=count+1
         flist{count}=str{1};
      end
   end
end

function [justname, extension, path]=FileNameAttributes(str)
fg=strfind(str, '\');
fb=0;
ff=0;
if length(fg)>0
   fb=fg(end);
end
fg=strfind(str,'/');
if length(fg)>0
   ff=fg(end);
end
if fb>ff
   f=fb;
else
   f=ff;
```

```
end
fg=strfind(str,'.');
if length(fg)>0
   fd=fg(end);
   justname=str(f+1:fd-1);
   extension=str(fd:end);
else
   justname=str(f+1:end);
   extension='';
end
path=str(1:f);
```

REFERENCES:

[1] Biometrics - A Look at Facial Recognition John D. Woodward, Jr., Christopher Horn, Julius Gatune, and Aryn Thomas

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

[3] T.Kanade, "Computer Recognition of Human Face", Basel and Stutgard: Birkhauser 1973.

[4] M.D. Kelly, "Visual Identification of People by Computer," Technical Report AI-I30 Stanford AI Project, Stanford, CA, 1970

[5] Face Recognition by Humans Pawan Sinha*, Benjamin J. Balas, Yuri Ostrovsky, Richard Russell Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology,

[6] C.T. Zahn. Graph-theoretic methods for detecting and describing gestalt clusters. IEEE Transactions on Computers 2001.

[7] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood estimation from incomplete data via the EM algorithm. Journal of the Royal Statistical Society. Series B (Methodological) 1977.

[8] Y. Cheng. Mean shift, mode seeking, and clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence 1995.

[9] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 2002.

[10] T. Hofmann and J.M. Buhmann. Pairwise data clustering by deterministic annealing. IEEE Transactions on Pattern Analysis and Machine Intelligence 1997.

[11] B. Fischer and J.M. Buhmann. Path-based clustering for grouping smooth curves and texture segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 2003.

[12] J. Shi and J. Malik. Normalized cuts and image segmentation. In Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition 1997.

[13] F. Chung. Spectral Graph Theory. American Mathematical Society, 1997.

[14] A density-based algorithm for discovering clusters in large spatial databases with noise. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu.

[15] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. 1998 ACM SIGMOD International Conference on Management of Data, pages 94{105, June 1998

[16] W.A. Barrett, A survey of face recognition algorithms and testing results, in: Signal, Systems, and Computer Conference of IEEE, 1998, pp. 301-305.

[17]MOVING OBJECT DETECTION,TRACKING AND CLASSIFICATION FOR SMART VIDEO SURVEILLANCE - August, 2004

[18] MOVING OBJECT DETECTION AND TRACKING IN WAVELET COMPRESSED VIDEO – August, 2003

[19] Face Databases. Ralph Gross. The Robotics Institute, Carnegie Mellon University

[20] Image compression using the Discrete Cosine Transform. The Mathematica Journal, 4(1), 81-88.

[21] M. Turk and A. Pentland (1991). "Eigenfaces for recognition". Journal of Cognitive Neuroscience 3 (1): 71–86.

[22] Face recognition in furrier spectra -- Hagen Spies-Interdisciplinary Center for Scientific Computing, Univ. of Heidelberg

[23] Face Recognition and data analysis-- Ian Schechter, USA & Michael Vorburger, Switzerland

[24] View-based object recognition Holger Bekel, Ingo Bax Lecture in summer 2004 University of Bielefeld

[25] A tutorial in principle component analysis , Lindsay I.Smith.

[26] Luigi Rosa Matlab facerecognition code.

[27] Matt's Matlab Tutorial Source Code Page.

[28] Face recognition in furrier spectra -- Hagen Spies- Univ. of Heidelberg

[29] Face recognition cognitive and computational processes. Sam S.Rakover and Baruch Cahlon.

[30]Face Recognition Using the Discrete Cosine Transform ZIAD M. HAFED AND MARTIN D. LEVINE Center for Intelligent Machines, McGill University,

[31] Face Recognition Jens Fagertun Kongens Lyngby 2005 Master Thesis IMM-Thesis-2005-74

[32] Face Recognition Based on Polar Frequency Features YOSSI ZANADept And ROBERTO

[33] Affinity Relation Discovery in Image Database Clustering and Content-based Retrieval - Mei-Ling Shyu, Shu-Ching Chen, Min Chen and Chengcui Zhang.

[34] Image Compression Using the Discrete Cosine Transform Andrew B. Watson NASA Ames Research Center

[35]Image Analysis for Face Recognition, Xiaoguang Lu Dept. of Computer Science & Engineering, Michigan State University.

[36] A Comparison of Face Detection Algorithms Ian R. Fasel1 and Javier R. Movellan2

[37] FERET (Face Recognition Technology) Recognition Algorithm Development and Test Results by P. Jonathon Phillips, Patrick J. Rauss, and Sandor Z. Der

[38] Eigenfaces for face recognition . Mattew turk and Alex Pentland. Visions and modeling group , the media laboratory,MIT.

[39] Biologically-based Face Recognition using Gabor Filters and Log-Polar Images María José Escobar*, Javier Ruiz-del-Solar

[40] Unsupervised Image-Set Clustering Using an Information Theoretic Framework Jacob Goldberger, Shiri Gordon, and Hayit Greenspan

[41] A continuous probabilistic framework for image Matching, Hayit Greenspan and Jacob Goldberger

[42] The CSU Face Identification Evaluation System User's Guide: Version 5.0 Ross Beveridge, David Bolme, Marcio Teixeira and Bruce Draper

[43] Automatic Face Recognition System Based on Local Fourier-Bessel Features Yossi Zana, Roberto M. Cesar-Jr and Regis de A. Barbosa

[44] Eigenfaces vs. Fisherfaces: Recognition Using Class Speci_c Linear Projection Peter N. Belhumeur Jo~ao P. Hespanha David J. Kriegman

[45] Face Detection and clustering for video indexing applications Csaba Czirjek, Noel O'Connor, Sean Marlow and Noel Murphy