AFIT/GE/ENG/91D-54

Face Recognition with the Karhunen-Loève Transform

THESIS

Pedro Fermin Suarez
Captain, USAF

AFIT/GE/ENG/91D-54

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

Face Recognition with the Karhunen-Loève Transform

5. FUNDING NUMBERS

Pedro F. Suarez, Captain, USAF

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/91D-54

Craig Arndt
AL/CFHI
Wright-Patterson AFB OH 45433

9. SPONSORING/MONITORING AGENCY REPORT NUMBER

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

The major goal of this research was to investigate machine recognition of faces. The approach taken to achieve this goal was to investigate the use of the Karhunen-Loève Transform (KLT) by implementing flexible and practical code. The KLT utilizes the eigenvectors of the covariance matrix as a basis set. Faces were projected onto the eigenvectors, called eigenfaces, and the resulting projection coefficients were used as features. Face recognition accuracies for the KLT coefficients were superior to Fourier based techniques. Additionally, this thesis demonstrated the image compression and reconstruction capabilities of the KLT. This thesis also developed the use of the KLT as a facial feature detector. The ability to differentiate between facial features provides a computer communications interface for non-vocal people with cerebral palsy. Lastly, this thesis developed a KLT based axis system for laser scanner data of human heads. The scanner data axis system provides the anthropometric community a more precise method of fitting custom helmets.

14. SUBJECT TERMS			15. NUMBER OF PAGES
Face Recognition, Karhunen-Loève, Image Processing			125
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT/GE/ENG/91D-54

Face Recognition with the Karhunen-Loève Transform

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

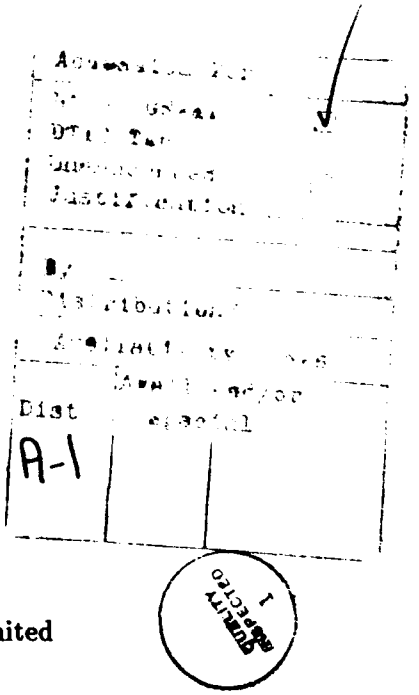
Requirements for the Degree of

Masters Science in Electrical Engineering

Pedro Fermin Suarez, B.S.E.E.

Captain, USAF

December, 1991



Approved for Public Release; distribution unlimited

Acknowledgments

I would like to thank my committee Major Steve Rogers, Dr. Matthew Kabrisky, and Captain Dennis Ruck for their guidance and insight. Thanks to my committee, the weekly thesis meetings were not only the academic highlight of my week, but a valuable exercise in technical problem solving. In addition, I would like to thank our sponsor Mr. Craig Arndt from the Armstrong Lab.

I would like to thank Captain Greg Tarr and Captain Kevin Priddy for their valuable assistance with just about everything. I want to thank my counterpart Captain Jim Goble for his periodic sanity checks during our face recognition research. I also want to thank Dan Zambon for keeping the computers going.

Lastly, I do want to thank all my Dayton friends, including Carrie, for their support and assistance in getting me through the last year and a half.

Pedro Fermin Suarez

Table of Contents

List of Figures

Abstract

The major goal of this research was to investigate machine recognition of faces. The approach taken to achieve this goal was to investigate the use of the Karhunen-Loève Transform (KLT) by implementing flexible and practical code. The KLT utilizes the eigenvectors of the covariance matrix as a basis set. Faces were projected onto the eigenvectors, called eigenfaces, and the resulting projection coefficients were used as features. Face recognition accuracies for the KLT coefficients were superior to Fourier based techniques. Additionally, this thesis demonstrated the image compression and reconstruction capabilities of the KLT. This thesis also developed the use of the KLT as a facial feature detector. The ability to differentiate between facial features provides a computer communications interface for non-vocal people with cerebral palsy. Lastly, this thesis developed a KLT based axis system for laser scanner data of human heads. The scanner data axis system provides the anthropometric community a more precise method of fitting custom helmets.

Face Recognition with the Karhunen-Loève Transform

I. Introduction

General Issue

Face recognition is an active area of research at the Air Force Institute of Technology (AFIT). The AFIT interest in face recognition stems from the Department of Defense (DoD) need to provide support for human security efforts. For example, by scanning the faces of people entering a secure or restricted area, a face recognition system could be used to control entry. Similar defense related applications of face recognition can easily be found in antiterrrorism and antinarcotic operations. As an answer to the DoD need, AFIT developed the Autonomous Face Recognition Machine (AFRM) which scans a room and identifies all the faces in the room. Unfortunately, the AFRM has limited accuracy. In order to improve the accuracy of machine based face recognition, this thesis investigates the use of the Karhunen-Loève Transform (KLT).

The following section provides the thesis background and justification. The third section formally states the research problem. In support of the problem statement, the following sections are presented: research objectives, research questions, methodology, standards, and scope. The last section provides an overview of the thesis.

Background

The AFRM is the first working face recognition system developed at AFIT. Originally completed in 1985 by Russel, subsequent additions were performed by Smith, Lambert, Sander, and Robb (18, 21, 10, 19, 14). To recognize subjects, the AFRM windows the face into six regions; the AFRM then performs a Discrete Fourier Transform (DFT) on each window (10:3-33–3-36). The subject is recognized from a set of known images using a minimum distance classification scheme (10:3-40). The performance of the AFRM, or any other pattern recognition system, is primarily dependent on the features used for

classification. Depending on the test conditions, the AFRM accuracy can vary from 56% to 90% (14:35–36). The poor accuracy of AFRM can be attributed to the use of Fourier coefficients as features. In this instance, the Fourier coefficients are not an adequate feature set for recognition. It has been shown that the KLT is an optimal transform based on minimum mean square error of reconstruction and maximum entropy of the representation (4:125). This thesis determines the optimality of the KLT feature set for face recognition.

Problem Statement

The primary problem is to recognize human faces. The key task associated with this problem is determining a good set of features that would allow a machine to accurately recognize human faces.

Research Objectives

The objective of this research is to investigate the optimality of the KLT. The secondary research objective is to develop an architecture and methodology for face recognition with the KLT.

Research Questions

The research completed for this thesis answers the following questions:

1. Can the KLT be implemented in a numerically practical manner?
2. Does the Karhunen-Loève orthogonal transformation of the face provide a better feature set for face recognition than traditional Fourier techniques?
3. What recognition accuracies can be expected from a Karhunen-Loève basis set?
4. Does holistic face recognition, which utilizes the whole face as a feature, provide better results than localized AFRM techniques?
5. Can a KLT based system be used as a computer communications interface for the non-vocal?
6. Can the KLT be used as an axis system for the anthropometry community?

Methodology

As part of this thesis, face recognition software is developed on the Silicon Graphics 4D Personal Iris workstation. The system software, which is written in ANSI C, is developed using a modular design approach. The algorithms are sufficiently flexible to be used on faces as well as any other image recognition problem, e.g. tanks, trucks, and jeeps.

This research investigates the KLT and develops software to implement the KLT and Fourier Transform. This thesis also demonstrates the image coding and reconstruction capabilities of the KLT. Most importantly, the KLT coefficients will be used for face recognition and the result will be compared to more traditional Fourier techniques. The recognition will utilize an Euclidean distance or first nearest neighbor classifier. Lastly, two additional applications of the KLT will be investigated. The first KLT application investigates the abilities of the KLT to differentiate facial features which would permit the KLT to be used as a facial feature communications interface for the non-vocal. The second application investigates the correlation properties of the Karhunen-Loève for use as an axis system for the anthropometry community.

Standards

The most important performance criteria is classification accuracy which is the percentage of correct classifications out of the total number of inputs. The inputs consist of both known and unknown subjects.

Scope

The focus of the thesis is feature selection and recognition. This thesis assumes that the detection and segmentation phases of the recognition problem are adequately met by the AFRM. The assumption is valid if the whole face is used as a feature and Lambert's moving target indicator is used.

Overview of Thesis

This chapter provides a brief background and defines the problem, objectives, questions, methodology, standards, and scope of the research. Chapter 2 provides additional background and a review of current research. The methodology used to solve the problem is presented in Chapter 3. The results of the research are presented in Chapter 4. Chapter 5 summarizes the significant results and provides recommendations. Appendix A lists the source code utilized for the thesis.

II. Background

Introduction

This chapter provides background and evaluates past and current research in the area of face recognition. Early face recognition systems approached the face recognition problem from an individual facial feature perspective, for example, the Fast Fourier Transform (FFT) of the eyes or the distance between the eyes (14, 5). Those early systems suffered from poor accuracy and segmentation. Recent research has approached face recognition from a more holistic approach by using the whole face as a feature (27, 2).

This chapter begins with an evaluation of the Air Force Institute of Technology (AFIT) Autonomous Face Recognition Machine (AFRM) (14). To emphasize that a good face recognizer requires good features, L. D. Harmon's profile recognizer is evaluated (5). Lastly, the chapter concludes with the evaluation of two more recent holistic face recognition techniques. At the Massachusetts Institute of Technology (MIT), Turk and Pentland have developed an approach using a holistic Karhunen-Loève Transform (KLT) (27). The second approach is from the University of California at San Diego, where Cottrell has developed a neural network based face recognition technique (2).

Air Force Institute of Technology Autonomous Face Recognition Machine

The design goal of the AFIT AFRM is to reliably recognize faces without human intervention. Face recognition which is a relatively simple task for humans can not be reliably accomplished by machines without a great deal of human intervention. The AFRM, AFIT's first attempt at face recognition, began with Routh's Ph'd dissertation on the Cortical Thought Theory (CTT). The CTT states that the brain uses the center of mass calculation, or gestalt, of an image for identification and storage (16). With Routh's theoretical work in place, the AFRM progressed from initial construction, by Russel, to subsequent modifications by various AFIT students (14, 10).

The AFRM, which consists of a video camera and a computer, first acquires the image. Acquiring the image consists of placing the subject in the field of view of the camera. Segmentation, or locating the face in the image, is accomplished with one of

two algorithms. The Smith algorithm searches throughout the image for the brightness 'signature' of a face (14:4). For instance, the brightness signature of a face is the change in brightness that occurs as the camera pans across the eyes. The second acquisition algorithm uses a moving target indicator, also called a spatio-temporal filter, to segment the moving pixels of the face from the stationary background (10:3-2). With acquisition and segmentation complete, preprocessing must be accomplished to provide normalized images for feature extraction and recognition.

AFRM preprocessing consists of brightness normalization, contrast enhancement, and scaling. One of the most significant results of Lambert's work is the development of a brightness normalization technique referred to as 'Lambertization' (10:E-2 – E-3). 'Lambertization' consists of scanning an image pixel by pixel and computing the local average of a pixel and its neighbors. The localized average is subtracted from that pixel and its neighbors. The resulting image is normalized with respect to local brightness. With the image normalized, the AFRM begins feature extraction and recognition.

To extract features, a window is drawn around the face. The window is divided into six sub-regions. A gestalt is calculated on each of the six sub-regions (10:2-25). A later modification, successfully made by Robb, substitutes the low frequency Fourier coefficients for the gestalt. This substitution is valid if you consider Mahler's animal cracker experiment, which validated Kabrisky's theory that the low frequency Fourier coefficients are a good model for the human visual recognition system (12, 6). The gestalt or Fourier coefficients are then stored in the data base for all known subjects. With features extracted, the remaining task is recognition and classification.

To recognize, the AFRM acquires and segments a test face until the gestalt or Fourier coefficients are determined. The Euclidean distance between the test subject and every subject in the data base is calculated. A minimum distance implies a match between the test and data base subjects.

The performance of the AFRM can be best measured by Robb's results in 1989. Using uncontrolled lighting and background, Robb's accuracy varied between 56% and 90% with a test set of 50 subjects (14:35-36). Although the AFRM performance is satisfactory,

the accuracy is insufficient for many real applications. The following techniques seek to dramatically improve recognition performance by finding a better set of features.

Profiles as Features

Can a set of features be found that permits accurate recognition of faces? In L.D. Harmon's *Machine Identification of Faces*, this question is answered (5). Harmon utilizes facial profiles as features for face recognition. Figure 2.1 exhibits the angles and distances between landmarks used by Harmon as features. Using principal-component analysis, also called eigen or covariance analysis, Harmon isolates 15 optimal features out of 38 features (5:108). Harmon's recognition accuracy is on the order of 98-100%, with a population of 112 manually segmented subjects (5:104). Harmon's results prove that with good features, an accurate face recognizer can be built. Harmon's work is being reevaluated by Hiroshi Agawa as a means of coding three dimensional face images (1).

Holistic Face Recognition

The AFRM and Harmon's profile recognizer are traditional feature based systems. Each part of the face is a feature, and as a result, a larger amount of measurement error is introduced. In holistic face recognition, the whole face is a feature which in turn results in less measurement error and improved recognition accuracy. For example, instead of using interpupillary distance as a feature, holistic features would consist of the Fourier coefficients of the entire face. There are two current techniques that implement holistic approaches to face recognition. At MIT, Pentland and Turk generate orthogonal eigenfaces with the KLT (27). At UCSD, Cottrell generates nonorthogonal basis functions with a neural network (2).

Karhunen-Loève Transforms

Eigenvalues. Eigenvalue analysis is a very powerful engineering tool. In image processing, eigen analysis supports image coding and compression applications. Eigenvalues

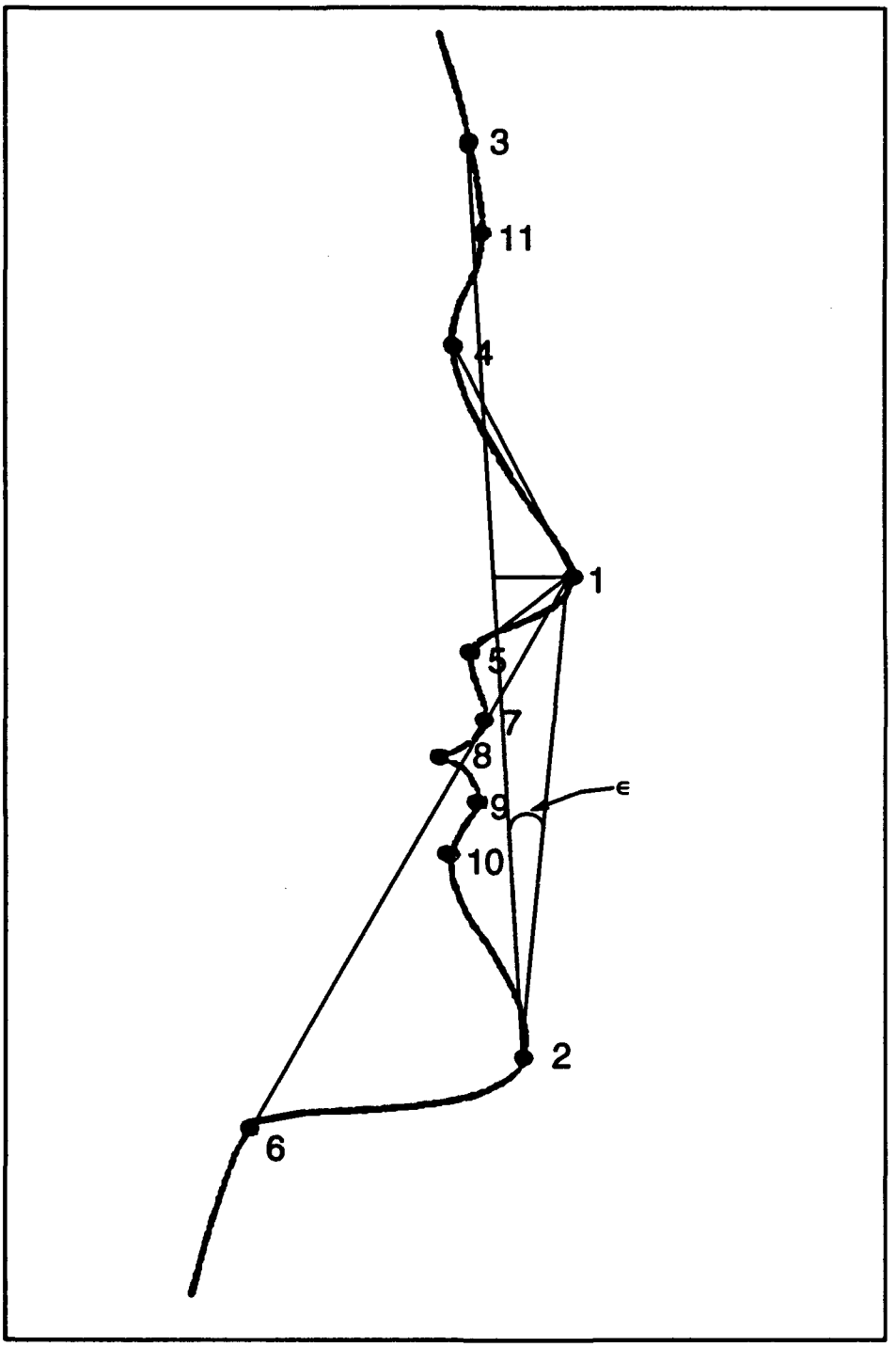


Figure 2.1. Harmon's profile with critical locations for feature extractions annotated. (5:97)

are the solution for λ in the equation

$$\mathbf{C_x}\vec{x} = \vec{x}\lambda \tag{2.1}$$

where $\mathbf{C_x}$ is a n^2 by n^2 matrix, $\vec{x}$ is an eigenvector, and λ is a scalar called the eigenvalue (9:345). The set of non-trivial solutions to the above equation are the eigenvalues which are also referred to as the spectrum of $\mathbf{C_x}$ (9:346). Each eigenvalue has a corresponding eigenvector, $\vec{x}$. The eigenvalues are found from the solutions to the following determinant

$$|\mathbf{C_x} - \lambda\vec{\mathbf{I}}|\vec{x} = 0 \tag{2.2}$$

where I is the identity matrix (9:347). The resulting eigenvalues are used to calculate the eigenvectors. As a by-product of the eigenvalue determination, the resulting eigenvectors are orthogonal.

Theory of the Karhunen-Loève Transform. The KLT is an orthogonal transform similar to the Fourier Transform. Unlike Fourier, which has a basis set consisting of sines and cosines, the KLT basis set consists of the eigenvectors of the covariance matrix. The KLT is also guaranteed to provide the most efficient and accurate transformation by minimizing mean square error (MSE) in reconstruction and maximizing entropy of the representation.

The KLT can be calculated in the three basic steps. First, obtain the mean and covariance function from the images. Second, calculate the eigenvalues for the centered covariance matrix, where centered implies that the mean has been subtracted off the image. For each eigenvalue, calculate the respective eigenvector and rearrange the n^2 eigenvectors into descending eigenvalue order. Select only the k largest eigenvalued eigenvectors out of all n^2 eigenvectors. This has the effect of greatly reducing the dimensionality of the pattern recognition problem, i.e. simplifying an n^2 problem to k, while minimizing MSE. The resulting k eigenvectors are the new KLT basis set. Lastly, the image can be transformed into the new basis set (3).

Before beginning the KLT explanation, a definition is in order. First, define the n by n pixel image as a vector

$$\vec{x^i} = \begin{pmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_{n^2}^i \end{pmatrix} \tag{2.3}$$

where $\vec{x^i}$ is the vector describing the i^{th} image and x_1^i is the first pixel of the i^{th} image. The construction of $\vec{x^i}$ is shown in Figure 2.2. The mean vector, $\vec{m_x}$, is calculated by

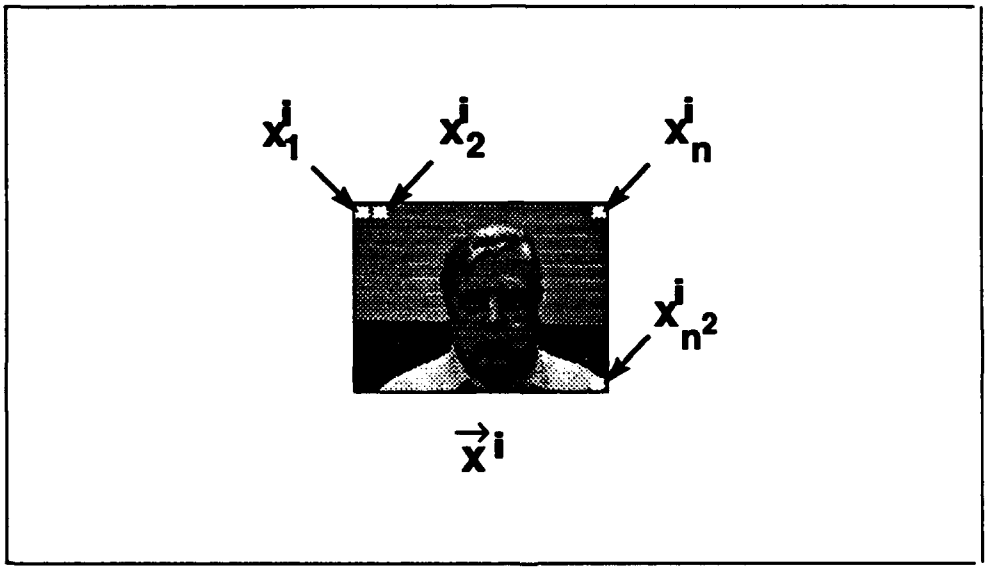


Figure 2.2. Definition of i^{th} image vector, $\vec{x^i}$, where the first pixel is $\vec{x_1^i}$ and the last pixel is $\vec{x_{n^2}^i}$.

averaging each pixel over all M faces in the training set

$$\vec{m_x} = 1/M \sum_{i=1}^{M} \vec{x^i} \tag{2.4}$$

where M is the number of faces in the training set and $\vec{x^i}$ is the vector describing the i^{th} face (4:123). The construction of $\vec{m_x}$ is shown in Figure 2.3.

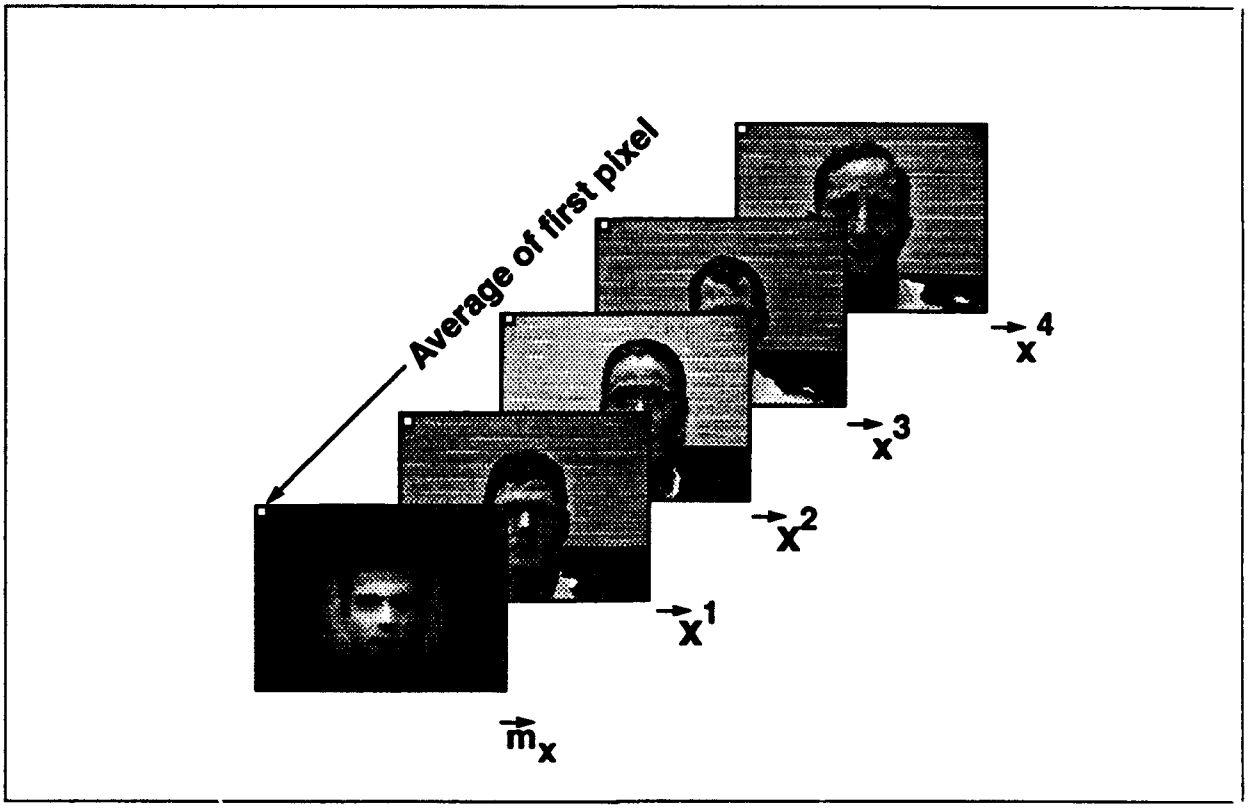


Figure 2.3. Definition of mean face, $\vec{m_x}$, where the first pixel of the mean face is the average of the first pixel of all the training images.

The covariance matrix, $\mathbf{C_x}$, is calculated via

$$\mathbf{C_x} = 1/M \sum_{i=1}^{M} (\vec{x^i} - \vec{m_x})(\vec{x^i} - \vec{m_x})^T \tag{2.5}$$

$\mathbf{C_x}$ results in an n^2 by n^2 matrix which has the general form of

$$\mathbf{C_x} = 1/M \begin{pmatrix} \sum_{i=1}^{M} x_1^i x_1^i & \sum_{i=1}^{M} x_1^i x_2^i & \cdots & \sum_{i=1}^{M} x_1^i x_{n^2}^i \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{M} x_{n^2}^i x_1^i & \sum_{i=1}^{M} x_{n^2}^i x_2^i & \cdots & \sum_{i=1}^{M} x_{n^2}^i x_{n^2}^i \end{pmatrix} - \vec{m_x}\vec{m_x}^T \tag{2.6}$$

where x_1^i is the first pixel of the i^{th} image. If the mean image is subtracted off all of the images, then

$$\mathbf{C_x} = 1/M \begin{pmatrix} \sum_{i=1}^{M} x_1^i x_1^i & \sum_{i=1}^{M} x_1^i x_2^i & \cdots & \sum_{i=1}^{M} x_1^i x_{n^2}^i \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{M} x_{n^2}^i x_1^i & \sum_{i=1}^{M} x_{n^2}^i x_2^i & \cdots & \sum_{i=1}^{M} x_{n^2}^i x_{n^2}^i \end{pmatrix} \tag{2.7}$$

The covariance matrix represents the information content or relative change in each image. By finding the covariance between the first pixel and every other pixel in the image, a judgement can be made as to the amount of information in the first pixel. For example, a pixel that does not change relative to other pixels provides little information, while a pixel with high covariance provides more information. Finding the eigenvectors and eigenvalues of the covariance matrix transforms the covariance matrix into an orthogonal space. In effect, the relative pixel changes are mapped into an orthogonal space. To an engineer, this orthogonal transformation represents an optimum distance between vectors and minimum decision errors which is the ultimate pattern recognition goal. The last KLT step of selecting the k highest valued eigenvectors creates a basis set with only the largest variance eigenvectors.

The eigenvalue solution to the covariance matrix is an n^2 simultaneous equation. This fairly lengthy computation can be accomplished with an approximation which will be demonstrated in a future section.

As previously mentioned, the resulting n^2 eigenvalues and eigenvectors are rank ordered with the highest eigenvalued vector in row one. Generally, the number of eigenvectors to be kept for reconstruction or recognition is determined using the MSE. The MSE between the reconstruction $\vec{\hat{x}}$ and the original image, $\vec{x}$, is

$$MSE = \sum_{j=1}^{n^2} \lambda_j^2 - \sum_{j=1}^{k} \lambda_j^2 \tag{2.8}$$

which simplifies to

$$MSE = \sum_{j=K+1}^{n^2} \lambda_j^2 \tag{2.9}$$

where λ_j is the eigenvalue, n^2 is the total number of eigenvalues, and k is the number of eigenvectors used for reconstruction (4:125). Using the MSE, the k largest eigenvalued eigenvectors are selected resulting in

$$\mathbf{u} = \begin{pmatrix} e_{11} & e_{12} & \cdots & e_{1n^2} \\ e_{21} & e_{22} & \cdots & e_{2n^2} \\ \vdots & & & \\ e_{k1} & e_{k2} & \cdots & e_{kn^2} \end{pmatrix} \tag{2.10}$$

where $\mathbf{u}$ is the truncated KLT matrix, e_{kn} is the n^{th} value of the k^{th} eigenvector, and the rows represent the eigenvectors associated with the λ_k eigenvalue (4:124). Each row represents a single eigenface, and k basis functions define the space. In vector form, the original image $\vec{x}$ is transformed by

$$y_k = \vec{u_k}(\vec{x} - \vec{m_x}) \qquad k = 1 \ldots M \tag{2.11}$$

where y_k is the k^{th} element of the projection vector, $\vec{y}$, in KLT space, $\vec{u_k}$ is the k^{th} eigenvector, $\vec{x}$ is the original image, and $\vec{m_x}$ is the mean face (4:125). In KLT space, $\vec{y}$ represents an image in a coordinate system defined by the eigenvectors. To reconstruct the image, multiply $\vec{y}$ by the inverse transformation, $\mathbf{u}^{-1} = \mathbf{u}^{\mathbf{T}}$, and add the mean face

$$\vec{\tilde{x}} = (\sum_{k=1}^{m} \vec{u_k}^T y_k) + \vec{m_x} \tag{2.12}$$

$\vec{\tilde{x}}$ is the reconstructed image, $\vec{u_k}$ is the k^{th} eigenface, y_k is the k^{th} KLT coefficient, $\vec{m_x}$ is the average face, and M is the number of images in the training set. With a theoretical understanding of the KLT, an application by Turk and Pentland can be evaluated.

Turk/Pentland Eigenface. The Matthew A. Turk and Alex P. Pentland eigenface is an attempt to recognize faces as a holistic entity (27:43). The Turk/Pentland approach is heuristically simple. First, with the KLT, develop an optimal basis set for recognition (27:45). With the basis set, transform the images and develop a data base with the known subjects and their KLT coefficients.

To recognize an unknown subject, the test image is transformed into KLT space, and the Euclidean distance of the KLT coefficients to each known subject is calculated. If the minimum distance meets a certain threshold, the image is classified as known. If the image does not meet the minimum distance criteria, the face is classified as unknown. For example, Figure 2.4 shows a two dimensional, two eigenvectors (u_1, u_2), face space. $\vec{y_1}$ which falls into class Ω_1 is classified as known. On the other hand, $\vec{y_2}$ which does not fall into the circled region of a known class is classified as unknown. Lastly, $\vec{y_3}$ represents a non-face (26:49) because it is relatively far from face space. This type of classification can be readily implemented with a nearest neighbor classifier or with a more robust Adaptive Resonance Theory (ART) Neural Network (15). With that explanation, a more analytical evaluation of Turk and Pentland follows.

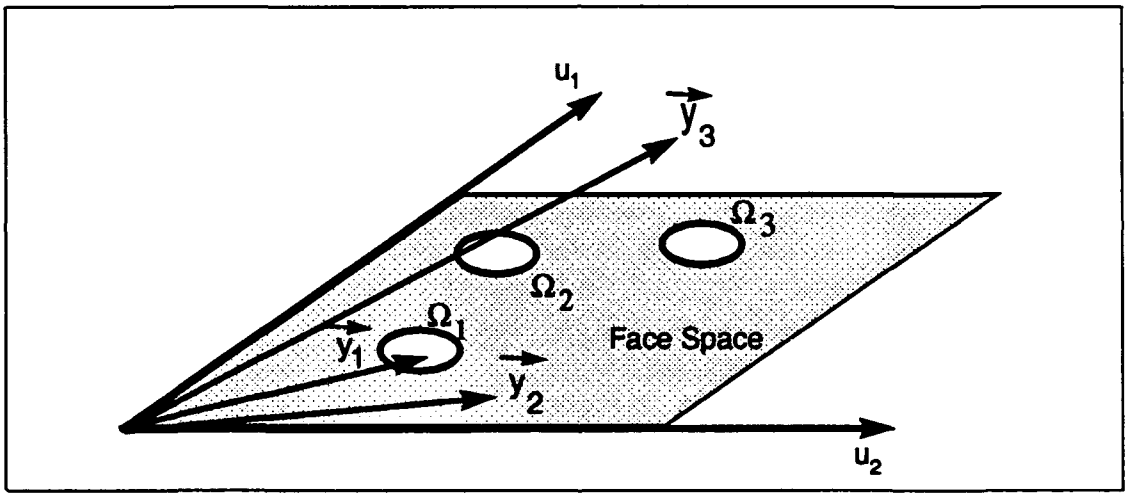


Figure 2.4. A simplified 2-D face space where u_1 and u_2 are the eigenface basis set and the known individuals are Ω_k. (27:49)

KLT Approximation. The KLT is numerically cumbersome. According to Turk, the n^2 by n^2 solution of the covariance matrix can be greatly simplified if the number of points in the image space, i.e. n^2, is much greater than the M training faces (26:6). For example, a training set of 40 subjects, each 128 by 128 pixels, would satisfy the condition that $M << n^2$, where $M = 40$ and $n^2 = 128^2$. The following technique is based on the previous condition.

Turk and Pentland simplify the KLT by estimating $\mathbf{C_x}$. First, if $\mathbf{A}$ is defined as the matrix which contains each training face as a column

$$\mathbf{A} = \left(\Phi_1 | \quad \Phi_2 | \quad \ldots \quad | \Phi_M \right) \tag{2.13}$$

where Φ_i is the i^{th} statistically centered training face. The first statistically centered training face would equal the first face minus the mean face, $\Phi_1 = \vec{x^1} - \vec{m_x}$. Typically, the KLT is created by first calculating

$$\mathbf{C_x} = \mathbf{A}\mathbf{A}^\mathbf{T} \tag{2.14}$$

where the eigenvectors of $\mathbf{C_x}$ are determined using Equations 2.1 and 2.2. Substituting $\mathbf{A}\mathbf{A}^\mathbf{T}$ for $\mathbf{C_x}$ in Equation 2.2 yields

$$\mathbf{A}\mathbf{A}^\mathbf{T}\vec{u_k} = \vec{u_k}\lambda \tag{2.15}$$

where $\mathbf{A}$ is the training matrix, $\mathbf{A}^\mathbf{T}$ is the transpose matrix, $\vec{u_k}$ is the eigenvector, and λ is the eigenvalue. In effect, $\vec{u_k}$ represents a vector that when multiplied by the covariance matrix, $\mathbf{C_x} = \mathbf{A}\mathbf{A}^\mathbf{T}$, equals the same eigenvector scaled by λ.

Now consider the eigenvectors of $\mathbf{L} = \mathbf{A}^\mathbf{T}\mathbf{A}$, the resulting matrix is only M by M. An eigen solution to $\mathbf{A}^\mathbf{T}\mathbf{A}$ would result in

$$\mathbf{A}^\mathbf{T}\mathbf{A}\vec{v_i} = \vec{v_i}\mu_i \tag{2.16}$$

where the eigenvectors of $\mathbf{L}$ are $\vec{v_i}$ and the eigenvalues are μ_i. If both sides are premultiplied by $\mathbf{A}$, the result, which is very reminiscent of the solution to Equation 2.15, is

$$\mathbf{A}\mathbf{A}^\mathbf{T}\mathbf{A}\vec{v_i} = \mathbf{A}\vec{v_i}\mu_i \tag{2.17}$$

where the eigenvectors of $\mathbf{A}\mathbf{A}^\mathbf{T}$ are $\mathbf{A}\vec{v_i}$ and the eigenvalues are μ_i. Now substitute $\mathbf{C_x}$ for AA^T

$$\mathbf{C_x}\mathbf{A}\vec{v_i} = \mathbf{A}\vec{v_i}\mu_i \tag{2.18}$$

$\mathbf{A}\vec{v_i}$ and μ_i are the respective eigenvectors and eigenvalues of the covariance matrix $\mathbf{C_x}$. Therefore, finding $\vec{v_i}$, the eigenvectors of **L**, and multiplying $\vec{v_i}$ by **A** results in $\vec{u_k}$, the desired eigenvectors of $\mathbf{C_x}$ (26:7).

To summarize, first generate **A** with the training faces. Second, multiply $\mathbf{A^T}$ and **A** to get **L**. Next, find the eigenvalues and eigenvectors of **L**, and multiply the resulting eigenvectors, $\vec{v_i}$, by **A** to get the desired eigenvectors, $\vec{u_k}$, of the original covariance matrix

$$\vec{u_k} = \sum_{j=1}^{M} v_{kj}\Phi_j \qquad i = 1 \cdots M. \tag{2.19}$$

where Φ_j is the j^{th} training face, v_{kj} is the j^{th} element of the k^{th} eigenvector of **L**, and M is the number of training faces. For instance, the first eigenface, $\vec{u_1}$, is a linear combination of each element in the first eigenvector, $\vec{v_1}$, with its corresponding training face in **A** (26:6-7). The matrix **L** has simplified the eigen analysis from n^2 to M (27:7).

With the **u** transformation matrix, Turk transforms the image into face space with the following transformation

$$\omega_n = \vec{u_n}(\vec{x^i} - \vec{m_x}) \qquad n = 1 \cdots k \tag{2.20}$$

where $\vec{x^i}$ is the image, $\vec{m_x}$ is the mean face, $\vec{u_n}$ is the n^{th} eigenface, ω_n is the coefficient of the n^{th} eigenface, and k is the total number of truncated eigenfaces (27:46). The values of ω_n determine the location of the transformed image in multidimensional face space. Lastly, Turk calculates the Euclidean distance to determine which face class is nearest the unknown face

$$\epsilon_j = |\Omega - \Omega_j| \tag{2.21}$$

where ϵ_j is the distance to each known face class , Ω is the unknown face class, and Ω_j is the vector for the j^{th} known face class (27:47). Pentland first classifies an unknown face as either face or not face, based upon ϵ_j being below a specified threshold. The minimum ϵ_j determines class membership.

The Pentland and Turk system utilizes a Sun workstation. The working system generates a seven dimensional eigenface face space. The recognition accuracy is 96% for a

population of 16 subjects (27:19). Figure 2.5 displays Pentland's reconstruction.

Figure 2.5. On the left is the original image. On the right the Turk and Pentland reconstructed image. (27:47)

According to Pentland, the approach does have limitations. Pentland's KLT is sensitive to head size, which forces size normalization. A limitation in segmentation also exists requiring Pentland and Turk to use a spatio-temporal filter similar to the moving target indicator of the AFRM (27:52–53).

Neural Networks and Face Recognition

Garrison W. Cottrell, at UCSD, implements a neural network to recognize faces (2). Cottrell's neural network is another holistic face recognition technique. Cottrell first trains his neural network using a back propagation neural network configured as an autoassociator, Figure 2.6 a. The input into the autoassociator is a 64 by 64 pixel image. At the output, the autoassociator monitors the MSE between the input and the output. The autoassociator iteratively adjusts the weights of the hidden layer nodes until the MSE is minimized. Minimizing MSE results in the input being reproduced at the output. Once the MSE is minimized, training is complete. The outputs of the hidden nodes are features that are used for recognition by another neural network.

To recognize faces, the input layer and hidden layer of the autoassociator, Figure 2.6 b, are input into a backpropagation network (BPN). The BPN is trained to recognize the nonorthogonal projections of each subject. If a test face which is input into the two layer network, Figure 2.6 b, activates one of the outputs of the BPN, the test subject is classified as known. If the test face does not activate a BPN output, the test image is classified as unknown.

The limitations of Cottrell's network are similar to Pentland's. However, Cottrell's neural network approach has one key advantage over the eigenface approach. The neural network technique does not have the computational burden of the KLT (27:53). This advantage is eliminated if the KLT is implemented on a neural network as demonstrated by Tarr (24).

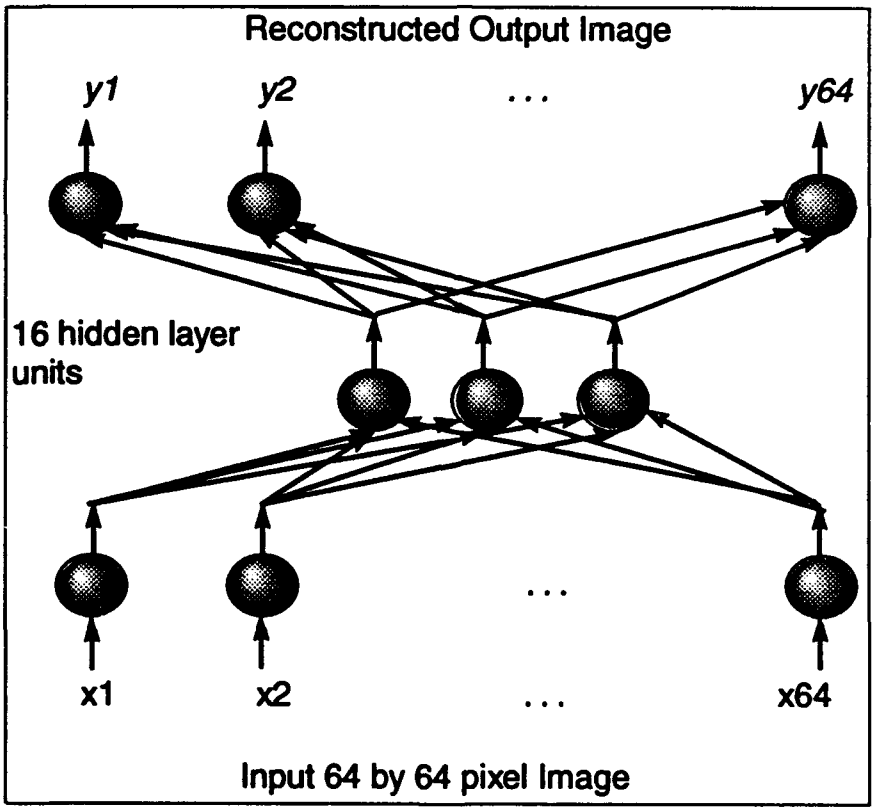


Figure 2.6. After the above Cottrell autoassociator is trained, the autoassociator input and hidden layers are used as the input to a second network which in turn classifies the faces.

Conclusion

This chapter evaluated AFIT's AFRM, Harmon's face profile research, Pentland and Turk's eigenfaces, and Cottrell's autoassociator. All the research implies that the primary factor in recognition accuracy is feature selection. More recent research indicates that holistic face recognition provides improved results over individual facial feature techniques. With these considerations, this thesis will concentrate on exploiting holistic KLT approaches to face recognition.

III. Methodology

General Issue

One of the objectives of this thesis is to evaluate the Karhunen-Loève Transform (KLT) as a possible basis set for face recognition. The evaluation of the KLT is divided into three phases. The first phase consists of using the KLT with only one training image presented multiple times. The purpose of the first phase is to simplify the problem in order to evaluate the algorithmic implementation of the KLT in C. The second phase consists of image reconstruction using both small and large population training sets. The third phase utilizes the KLT for recognition and classification. As a by-product of the three phases, an evaluation of finding or segmenting faces in KLT space will be performed. Additionally, a facial feature communicator for the non-vocal and a KLT based axis system will be implemented. Two additional KLT applications will be evaluated. Before the three phases are presented, this chapter begins with an overview of the primary software routines utilized for the thesis effort.

Code development

Introduction. The software written in ANSI C consists of three main algorithms. The first algorithm, kl_transform2.c, calculates the KLT of the input training set. With the eigenfaces generated by kl_transform2.c, the second algorithm, recon.c, calculates the KLT coefficients and reconstructs the input image. The last algorithm, find_min.c, classifies the coefficients based on Euclidean distance. In addition to the previous routines, two KLT preprocessing routines, gwind.c and C_late5.c, will be detailed. The software routine gwind.c windows the images with a Gaussian. The second preprocessing routine, C_late5.c, centers all the images relative to a reference. All software listings for this thesis can be found in Appendix A.

KLT software. The primary function of the KLT program, kl_transform2.c, is to calculate the optimum basis set, or eigenvectors, and the average vector of a training set. The training set used to create the eigenvectors defines the basis set, therefore a training

set of faces generates an optimum basis of eigenfaces. Note that the eigenfaces are only optimum for images in the training set. As the images become less like the training images, the optimality quickly disappears. This implies that a basis set of eigenfaces would only be suitable for faces. If a KLT basis set is desired for tanks, for example, the training set would have to be modified to include tanks. The resulting basis set would then provide eigentanks. The routine kl_transform2.c is primarily used on faces in this thesis, but the software can readily be implemented with any training set of images.

The KLT software is developed with two key references: Turk and Pentland's *Eigenfaces for Face Recognition* and Gonzalez and Wintz *Digital Image Processing* (26, 4). The C program kl_transform2.c which is documented in Appendix A is executed from the Unix command line with

```
kl_transform trainingfile sizeofimage numberoftrain
```

where trainingfile is the name of the file which lists the faces in the KLT training set, sizeofimage is the total image size in pixels, and numberoftrain is the number of images in the training set. The block diagram of kl_transform2.c is shown in Figure 3.1.

First, each training face has the average face subtracted off resulting in

$$\mathbf{A} = \left(\Gamma_1 | \quad \Gamma_2 | \quad \ldots \quad | \Gamma_M \right) \tag{3.1}$$

where A is the KLT training set matrix, Γ_M is the M^{th} statistically centered image, and M represents the total number of training images. To simplify the calculation of the KLT, rather than calculating the eigenvectors of the covariance matrix $\mathbf{C_x} = \mathbf{A}\mathbf{A}^{\mathbf{T}}$, the eigenvectors of $\mathbf{L}$ are found

$$\mathbf{L} = \mathbf{A}^{\mathbf{T}}\mathbf{A} \tag{3.2}$$

The *Numerical Recipes in C* routine jacobi.c finds the solution to Equation 2.2, where $\mathbf{L}$ is substituted for C_x (13). A second *Numerical Recipes in C* routine eigsort.c sorts the eigenvectors generated by jacobi.c in decreasing eigenvalue order (13). Note that the resulting eigenvectors from $\mathbf{L}$ are only of length M which is the size of the training set. One additional step is required to arrive at the appropriate n^2 length eigenvectors. This

is accomplished with Equation 2.19 to find $\vec{u_k}$ the n^2 length eigenvector. The outputs of kl_transform2.c are the average face, eigenfaces, and eigenvalues.

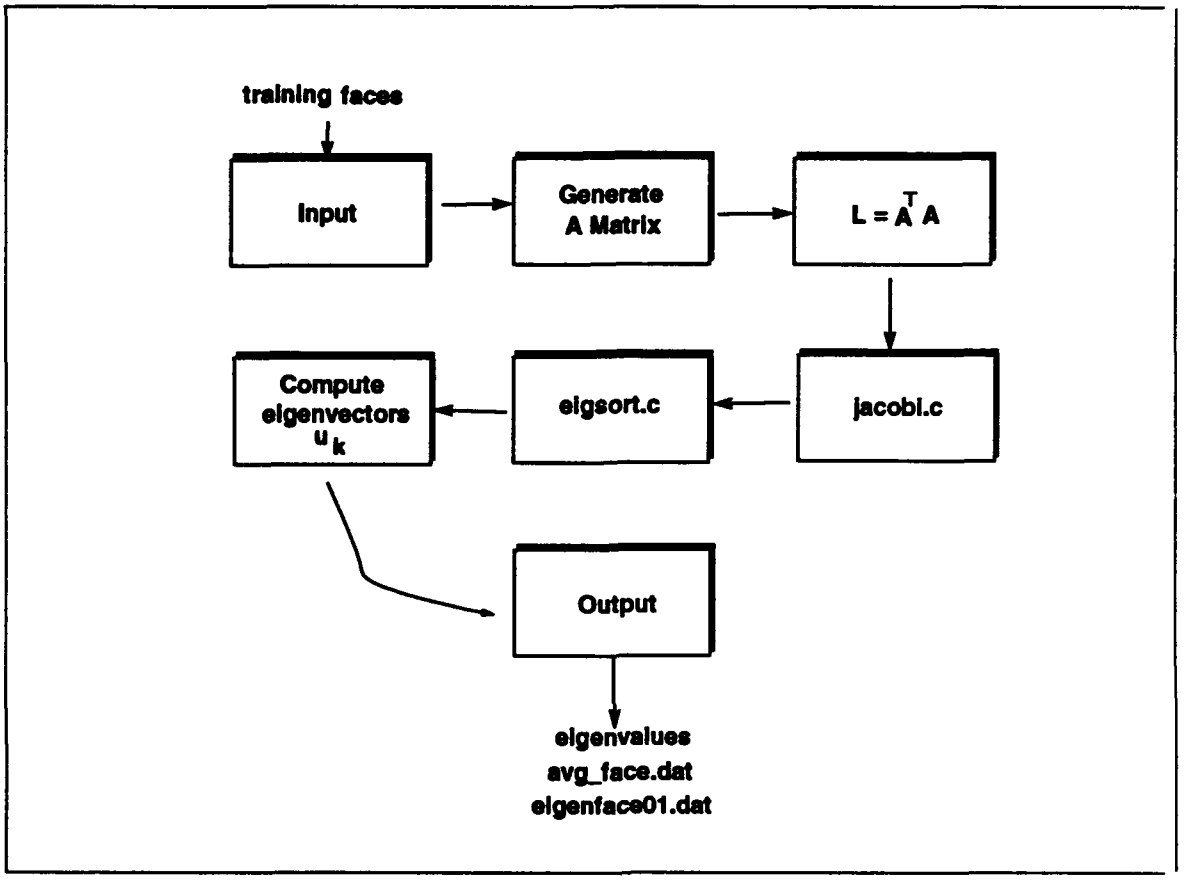


Figure 3.1. Block diagram for the C program kl_transform2.c which finds the eigenfaces, i.e. KLT basis set, and the average face for a given training set.

Reconstruction of faces. Thus far the eigenfaces and average face have been calculated with the program kl_transform2.c. This subsection explains the implementation of the software routine recon.c. The program recon.c calculates the KLT coefficients of the image, and with the eigenspace representation of the image, reconstructs the original image.

The C program recon.c which is documented in Appendix A is invoked at the Unix prompt as follows

```
recon inputimage numberinrecon sizeofimage
```

where inputimage.gra is the image to be reconstructed, inputimage.rec is the reconstruction, numberinrecon is the number of eigenfaces used in the reconstruction, and sizeofimage is the total number of pixels in the input image. First, recon.c reads the input image, the average face, and the eigenfaces needed for reconstruction. The image $\vec{x}$ is then projected into the new eigen coordinate system. The new vector $\vec{y}$ is created by

$$y_k = \vec{u_k}(\vec{x} - \vec{m_x}) \qquad k = 1 \ldots M \tag{3.3}$$

where y_k is the k^{th} element or coefficient of the k^{th} eigenface, $\vec{u_k}$ is the k^{th} eigenface, $\vec{x}$ is the input image, $\vec{m_x}$ is the average face, and M is the number of images in the training set. The vector $\vec{y}$ is the projection of the input image into KLT space. These coefficients are also features for recognition and classification. The reconstruction is the weighted sum of the coefficients of $\vec{y}$ with the appropriate eigenface. In vector form

$$\vec{\tilde{x}} = (\sum_{k=1}^{m} \vec{u_k}^T y_k) + \vec{m_x} \tag{3.4}$$

where $\vec{\tilde{x}}$ is the reconstructed image, $\vec{u_k}$ is the k^{th} eigenface, y_k is the k^{th} KLT coefficient, $\vec{m_x}$ is the average face, and M is the number of images in the training set. The outputs of recon.c are the reconstructed image and the KLT coefficients.

Euclidean Distance Classifier. The previous two software routines provided an eigenface basis set, generated KLT coefficients, and reconstructed an image from its KLT coefficients. To actually recognize test from prototype faces, a classifier must be implemented. In this thesis a Euclidean distance or first nearest neighbor (1-NN) classifier is utilized. The routine find_min.c documented in Appendix A is executed from the Unix command line with

```
find_min trainfile testfile #_train  #_test #_features
```

where trainfile is a file containing the prototype vectors, testfile is a file containing the test vectors, #_train is the total number of prototype vectors, #_test is the total number of test vectors, and #_features is the number of features in each vector. Each test and prototype

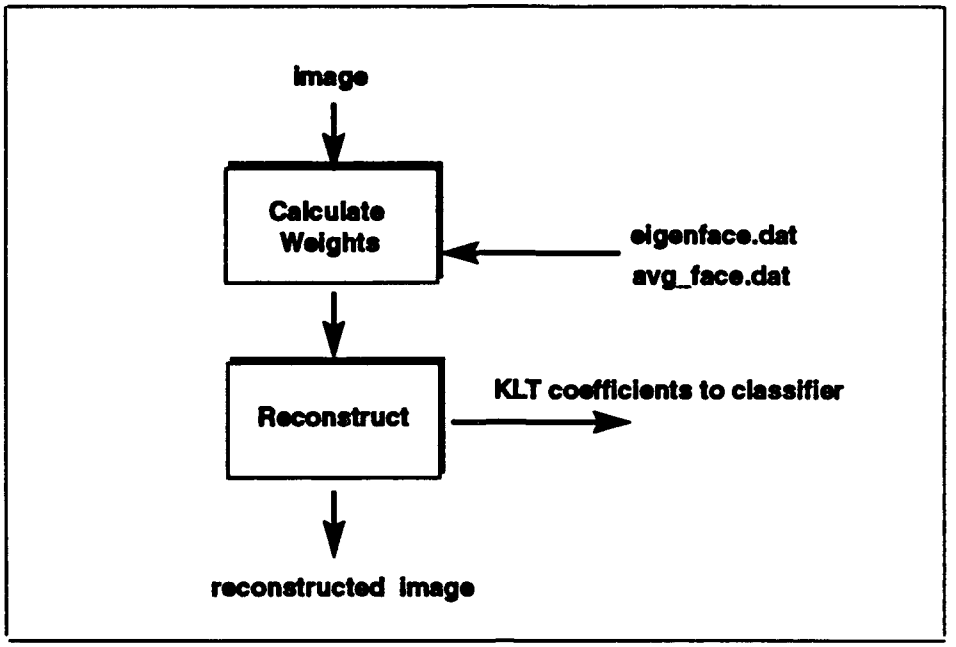


Figure 3.2. Block diagram for C program recon.c

vector ends with a string or label identifying the vector class. The program find_min.c statistically normalizes the test and prototype features. The normalized Euclidean distance from the test vector to each prototype vector is calculated. The minimum distance from test to prototype implies that the test vector is of the same class as the prototype. After program completion, find_min.c lists each test vector with the associated winning prototype and distance. The probability of error of a Euclidean distance or 1-NN classifier approximates a Bayesian classifier as the number of test and prototype vectors approaches infinity (25:83). In practice, with more than twelve classes the 1-NN approximates the Bayesian classifier.

Preprocessing Faces with a Gaussian Window. Applying a Gaussian window to face images has several advantages. A Gaussian windowed face, Figure 3.3, has its center accentuated and the outline of the face de-emphasized. From a recognition prospective, de-emphasizing the outline of the head de-emphasizes hair and background and emphasizes the center of the face. Both effects are desireable for recognition. From a similar biological perspective, when humans recognize faces, the eyes spend more time at the center of the face and less on the outline(11). Figure 3.4 demonstrates a typical eye scan pattern of a

human examining a face. The eye scan pattern which is taken with an occulometer is in a sense Gaussian with considerably more scans on the center of the face than on the outline. An additional benefit of Gaussian windowing is that the centering of the images is focused on the eyes instead of the outline of the head. In short, Gaussian windowing the face is biologically motivated, provides more desirable recognition, and more desirable centering.

Figure 3.3. An example of a face windowed with a Gaussian.

The window is impiemented with the software routine gwind.c, see Appendix A. The program takes an input image and multiplies each pixel by a corresponding pixel in the Gaussian window, Figure 3.5. First the routine gwind.c prompts for the image size, and the routine then prompts for the x mean, x variance, y mean, and y variance. The center of the Gaussian window is defined by x mean and y mean while x variance and y variance define the amount of change in the x and y direction. For example, Figure 3.5 demonstrates a Gaussian window with parameters: x mean=65, x variance=35, y mean=55, y variance=50. The center of the 128 by 128 pixel Gaussian window is right 65 pixels and up 55 pixels, and the window variances define the tightness of the window. In most instances, the mean values stay constant and the variances are varied.

Preprocessing Faces by Centering. The KLT is not shift invariant. In other words, shifting a face in a scene provides different KLT coefficients. This undesirable effect can be eliminated by centering the images relative to a reference. The centering operation is performed by a simple routine C_late5, see Appendix A. The program does a Fast Fourier Transform correlation of the input image and an already centered reference face. The correlation indicates how much shift is necessary in x and y pixels to align the

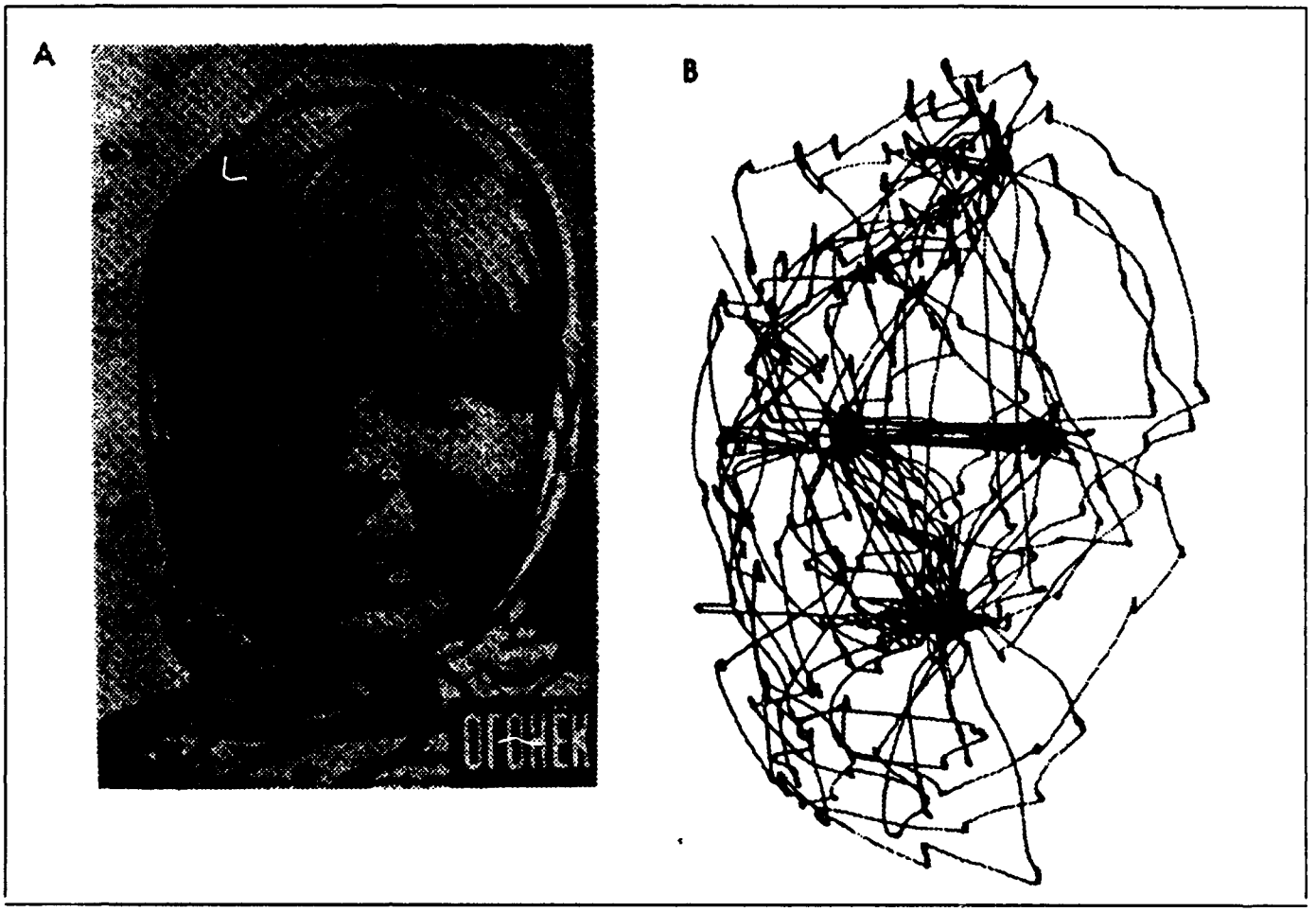


Figure 3.4. The original scene of the little girl is on the left and the resulting occulometer pattern is on the right. Note the concentration of scans on the center of the face. (11)

input image with the reference. The routine then shifts the input image the appropriate amount and stores the shifted image. Unless otherwise indicated, the program C_late5 does not implement energy normaliztion.

Testing. To test the code, a simple KLT problem is devised. The training set consists of nine instantiations of a single face, with each face differing by the addition of various amounts of Gaussian noise. Properly running software provides accurate reconstruction and a rapidly decaying mean square error (MSE) plot. The results of this test are provided in the next chapter.

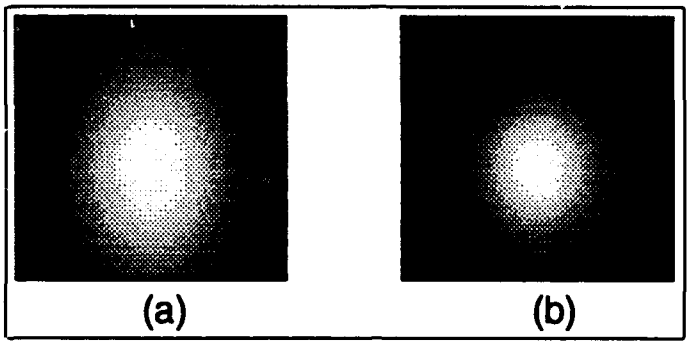


Figure 3.5. a. A Gaussian window with parameters x mean=65, x variance=35, y mean=55, and y variance=50. b. A Gaussian window with parameters x mean=65, x variance=25, y mean=55, and y variance=30.

KLT Reconstruction

Small Population Reconstruction using the KLT. The KLT is typically seen in text books as an image compression technique (4). To evaluate the KLT, a test is set up with a small population of six faces in the training set. The faces are recorded on video tape and digitized using Snapshot, a NeXT application package. The input images, 320 by 480 pixels, are scaled to 256 by 256 pixels. To prevent blurred reconstructions, all the images are centered. A Gaussian window is then applied which decreases the effects of hair line, head size, and background on recognition. A second centering operation is performed to concentrate centering on the center of the face , i.e. eyes. With all the images aligned, the KLT is implemented with kl_transform2.c. As previously outlined, kl_transform2.c creates the average face and eigenfaces. After the eigenfaces and average face are calculated, any of the six faces can be reconstructed with recon.c. The results of the reconstruction are provided in the next chapter.

Large Population Reconstruction using the KLT. The reconstruction of images from large KLT populations seems to be a logical extension to the small population test. The primary objective of the following tests is to evaluate the quality of reconstruction with different size training sets and Gaussian windows.

The first test evaluates a training set of 27 subjects. Each subject has three different instantiations for a total of 81 faces in the training set. The Gaussian window of Figure

3.5 a is applied. This test evaluates reconstruction of images in and out of the training set.

The second test evaluates the same training set as in test one but with a tighter Gaussian window around the faces. The Gaussian window of Figure 3.5 b is applied. This test evaluates the effects of different window sizes on reconstruction quality.

The third test evaluates 27 training faces with only one instantiation per test subject. The Gaussian window of Figure 3.5 a is applied. This test evaluates the effects on reconstruction of using only one version of each training face.

The fourth test evaluates reconstruction with only 18 faces in the training set and the Gaussian window of Figure 3.5 b is applied. This test evaluates training set size reduction and image reconstruction quality.

The fifth test demonstrates the effects of not using Gaussian windowed images. Lastly, the sixth test evaluates reconstruction of a subject not in the KLT training set.

The basic set-up, for all the tests is: develop the KLT, reconstruct faces, and, lastly, heuristically evaluate the quality of the reconstruction for each test. The results of the large population reconstruction are provided in Chapter 4.

KLT and Recognition

Introduction. Previous experiments evaluated the reconstruction capabilities of the KLT, but the goal of this research is to develop a set of features for accurate face recognition.

Single Person Verification. The first evaluation of the KLT recognition capability consists of single person verification. For example, a subject enters a controlled access facility and claims to be person A. The subject then enters into a terminal a personal identification number (pin) code. At the same time a video camera takes a picture of the face. The KLT coefficients of the image verifies if person A is truly person A, or an impostor with the right pin.

This problem is evaluated as a two class problem with class one consisting of the test subject A and class two consisting of every other person in the data base. A KLT with

18 different training images encodes the images. Using only the first five KLT coefficients, a Backpropagation Network (BPN) with momentum classifies the data (22). Various experiments are run varying the number of input features and hidden layer nodes. Lastly, using the same test scenario and data, a Euclidean distance classifier measures recognition accuracy. The results of single person verification are provided in Chapter 4.

Multi-person Recognition. Determining the identity of a subject from multiple subjects is the multi-person recognition problem. The population of faces consists of fifty-five subjects. Each subject is digitized into 128 by 128 pixel frame and is part of the KLT training set. All the images are shifted, aligned, and windowed as in Figure 4.5. Executing kl_transform2.c provides the average face, eigenfaces, and eigenvalues. With the eigenfaces, the features can be extracted using recon.c.

Before the features are extracted, the six images are divided into test and prototype sets. The test set consists of two images which will be compared to the remaining four in the prototype set. The program recon.c extracts the features providing the KLT coefficients of each image. With the KLT coefficients of both the test and prototype sets, a Euclidean distance classifier, find_min.c, compares the test set to the prototype set. A minimum distance implies a match. The aforementioned multi-person test is performed again, except only forty images are included in the KLT training set. The results of multi-person recognition are provided in Chapter 4.

As a means of comparing the KLT to a more common technique, the low frequency FFT magnitude coefficients are used for recognition. The FFT is energy normalized and the zero frequency response (DC) is the center of the FFT. Since the FFT magnitude of the face, Figure 3.6, is symmetric, only the top half of the FFT magnitude is utilized. For example, a first order block of the FFT magnitude consists of a block with boundaries defined by DC plus one coefficient to the right, left, and up. The first order block provides six coefficients because the block is three by two.

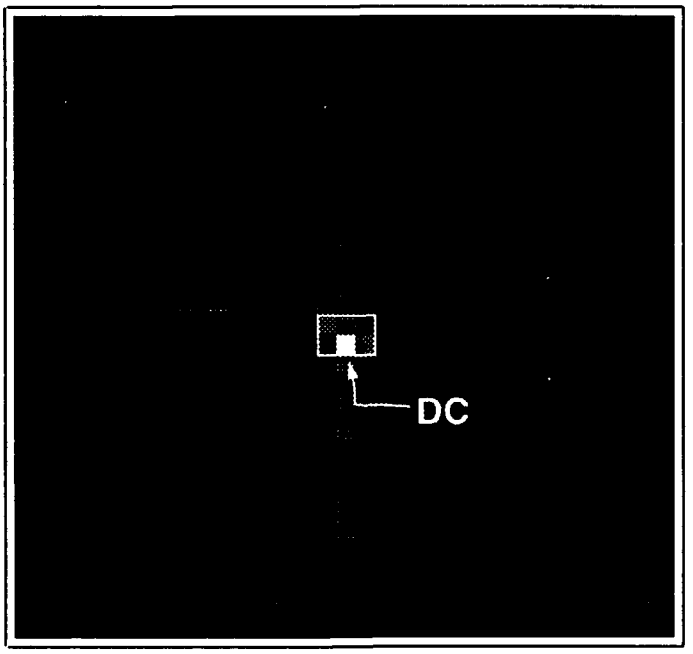


Figure 3.6. FFT magnitude of a face with the first order 3 by 2 block

Finding Faces in Face Space

Finding a face in a scene is the first step in any pattern recognition system. The Air Force Institute of Technology (AFIT) Autonomous Face Recognition Machine (AFRM), implements a spatio temporal filter or moving target indicator to segment a face from the overall scene. Segmention based on the KLT would segment on 'faceness' which would be more desireable than segmentation based on motion. Two approachs will be evaluated.

The first segmentation approach will scan a scene. Each sub-image scanned will be projected into face space. If the Euclidean distance of the KLT coefficients of the sub-image is less than a certain threshold, a face is probably present. This approach is the same approach taken by Pentland (27).

The second segmentation approach will take the FFT of the scene and of the eigen-faces. An energy normalized correlation between the scene and eigenface is performed to determine the location of the face in the scene. The results of both techniques are provided in Chapter 4.

Karhunen-Loève based Facial Feature Communicator for the Non-vocal

Many times a Masters thesis only serves to enhance academic understanding. In this instance, however, an application of the Karhunen-Loève may provide a child with cerebral palsy a means of communicating with the outside world. The child has cerebral palsy and is incapable of speaking. Although the child can not speak, she can manage facial expressions. The following test evaluates the ability of the KLT eigenvectors or eigen images to differentiate between two expressions. The first expression, tongue out, is classified as a 'yes'. The second expression, mouth open, is classified as a 'no'.

Two 128 by 128 pixel images of the subject shown in Figure 3.7 are selected as prototype images. The two prototype images are used as training images to generate the

Figure 3.7. The two prototype images of the non-vocal subject. On the left is the tongue out or 'yes'. On the right the mouth open or 'no'.

KLT eigenvector basis set. The prototype images are first Gaussian windowed with Figure 3.8 to remove the effects of the background and enhance the mouth area. The images are then aligned with C_late5.c a FFT based correlation algorithm. Finally, the KLT is calculated with kl_transform2.c of Appendix A generating the average vector and the first two eigenvectors. A block diagram of the aforementioned process is shown in Figure 3.9.

With the KLT basis, the next step consists of calculating the KLT coefficients. Figure 3.10 summarizes the following procedure. An input image is first Gaussian windowed with Figure 3.10. The image is then aligned. The KLT coefficients are calculated with the

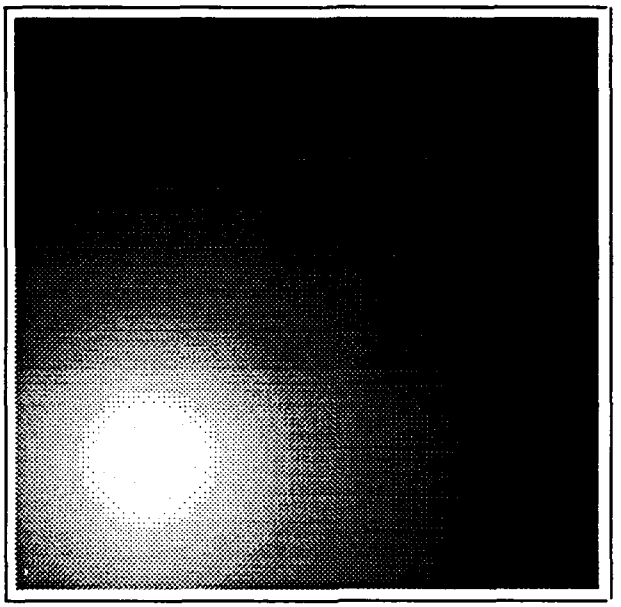


Figure 3.8. The Gaussian window used on the images of the non-vocal subject.

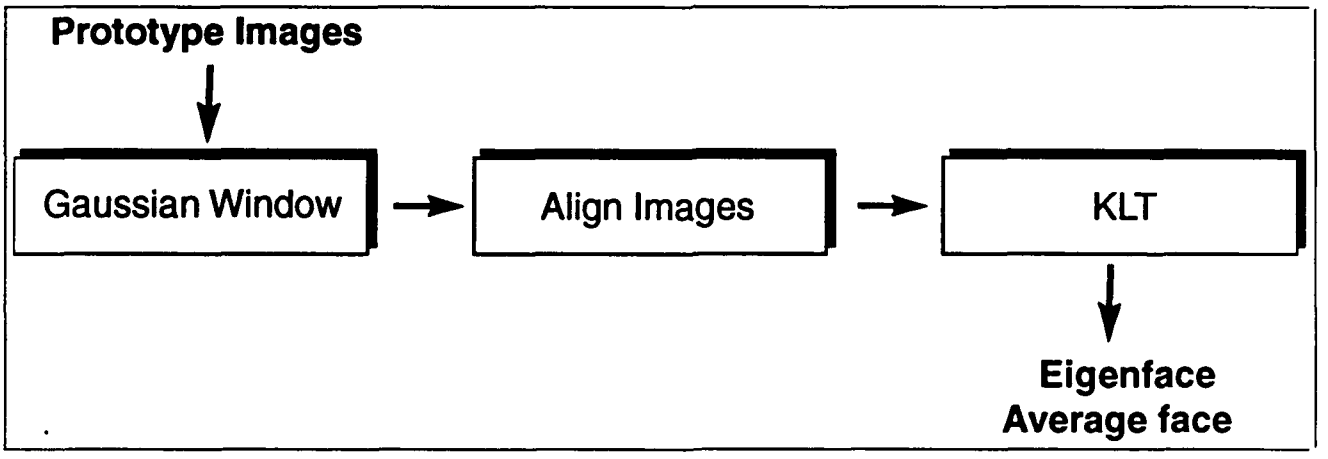


Figure 3.9. The process used to generate the KLT basis set from the prototype images of the non-vocal subject.

routine from Appendix A recon.c. Calculating the KLT coefficients is done with Equation 3.3 and consists of subtracting off the mean from the input image and then taking the dot product between the image and each of the two eigenvectors. The two KLT coefficients are classified with the Appendix A routine find_min.c, a Euclidean distance classifier. Before the test images of Figure 3.11 are classified, the prototype images are processed through Figure 3.10 to determine the prototype KLT coefficients. With the prototype KLT coefficients for the 'yes' and 'no' facial expressions, the test images can be projected into Karhunen-Loève space and classified with the NN classifier. The minimum distance to a prototype determines if the input or test image is a 'yes' or 'no'. The results of this test are presented in Chapter 4.

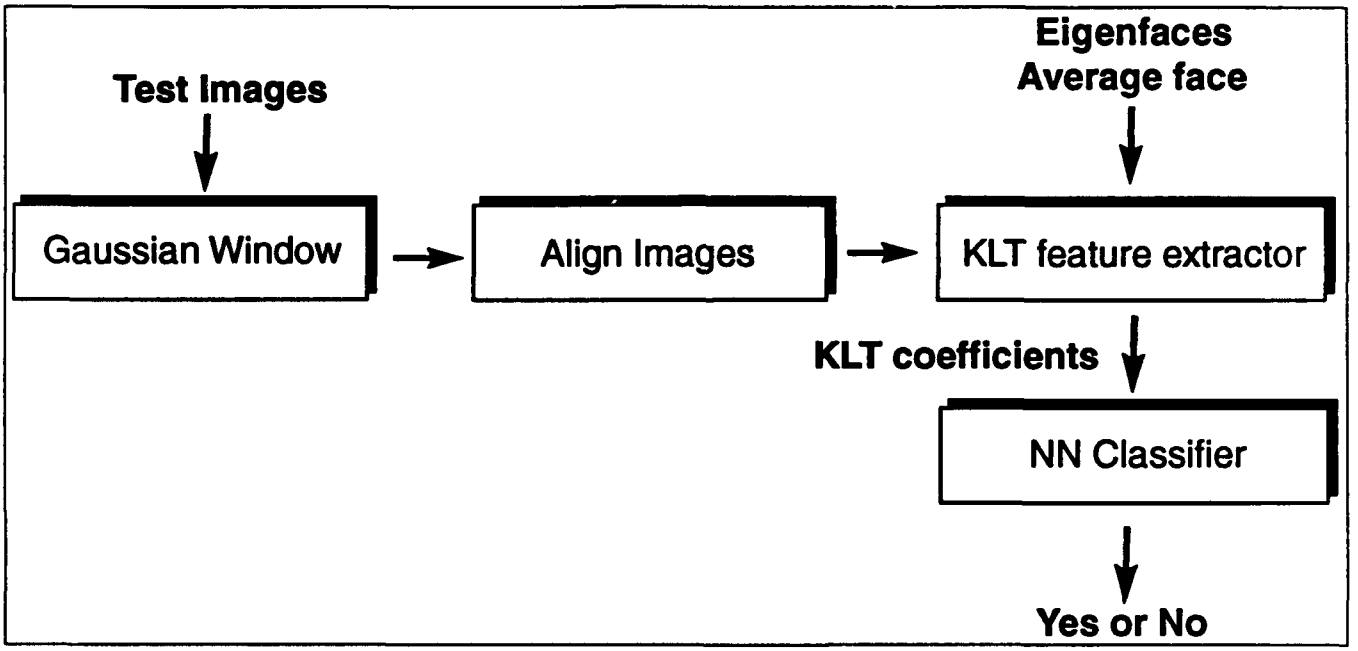


Figure 3.10. The process used to calculate the KLT coefficients and classify the images as 'yes' or 'no'.

Karhunen-Loève Axis System

The Karhunen-Loève Transform (KLT) is typically utilized in image compression or recognition applications. However, if the KLT is to be truly useful as a block transform, it is essential to be able to correlate using the KLT. A sponsor of this research had a

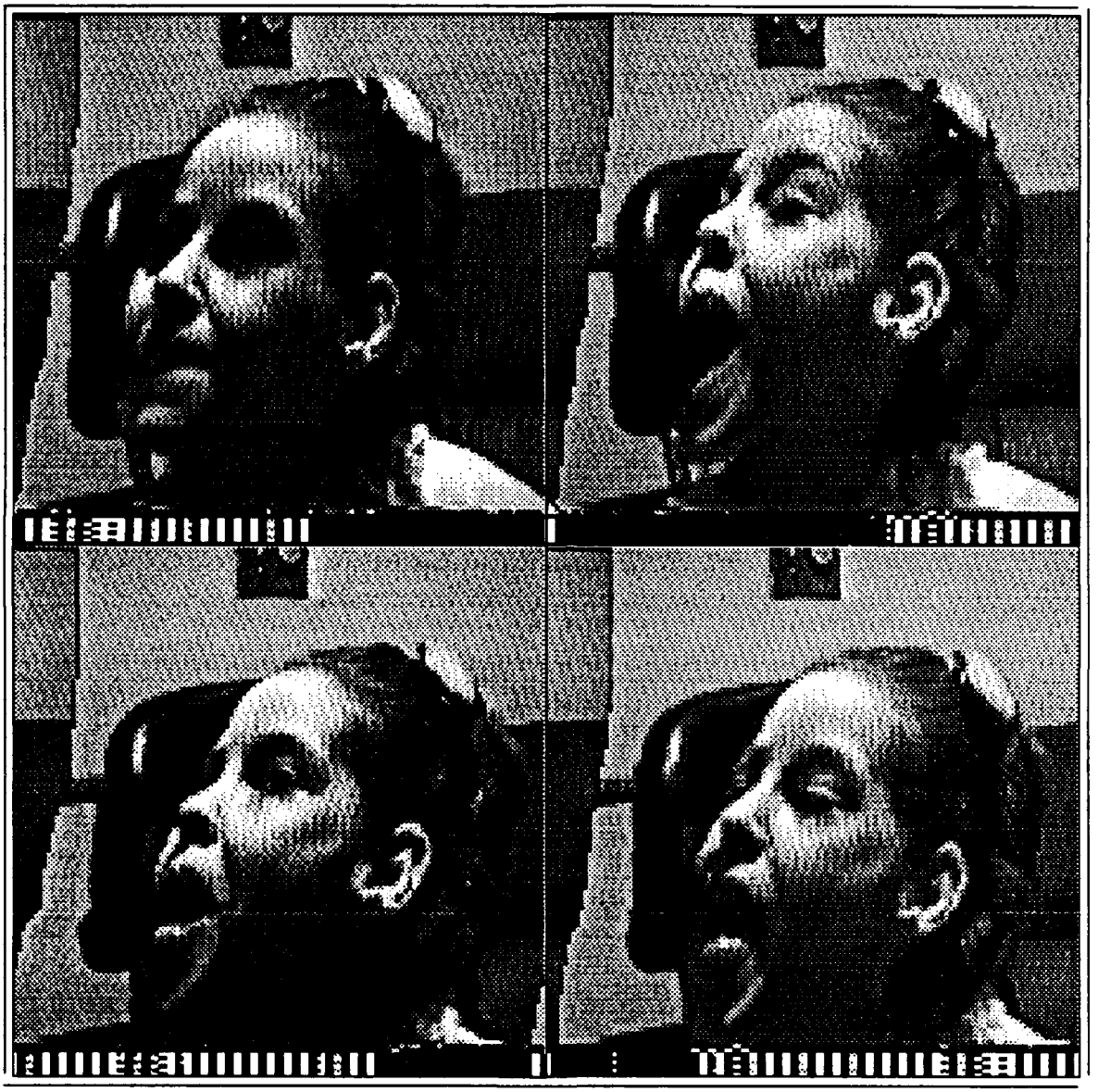


Figure 3.11. The two top test images are mouth open or 'no'. The two bottom test images are tongue out or 'yes'.

unique data set of human heads taken with a laser radar scanner. The scanner distance measurements of the head are used to fit flight gear on pilots. The laser scanner provides detailed measurements for all of the features of the face and head.

A current short coming of the laser scanner data collection is the lack of an axis system. An axis system speeds the search of features by placing all the heads in the same relative position. Like a map with a longitude and latitude coordinate system, the axis system simplifies the task of finding different land marks on the head. Therefore, searching for noses will always occur in the same general area of the axis system. The following research demonstrates the use of Karhunen-Loève eigenvectors or eigen images as an axis system and compares the results to an equivalent Fourier based system.

Laser Scan Data The laser scan data of the head consists of distance measurements of the head. The scanner goes around the head and collects the data. The training images will define the basis set; therefore, selection of the training set should be representative of the laser scan data. The data consists of a matrix which is 256 data points in the vertical direction and 512 data points in the horizontal direction. The distance measurements are converted from distance to gray scale using the routine kl_med.c which is documented in Appendix A. The converted gray scale 512 by 256 pixel image is shown in Figure 3.12.

First a KLT training set of nine laser scanner head images is chosen at random to represent the entire population. The training images are aligned manually to improve the quality of the resulting eigen images. The KLT is calculated using the C program kl_transform2.c which is listed in Appendix A. The routine kl_transform2.c provides the Karhunen-Loève eigenvectors and average vector.

With the basis set calculated, a non-training image Figure 3.12 a, is selected and shifted. After every shift, the projection into Karhunen-Loève space is calculated using only the first eigenvector. The projection is calculated with the Appendix A routine recon.c which uses Equation 3.3 where $\vec{u_1}$ is the first eigen image, $\vec{x^1}$ is the shifted image, and ω_1 is the KLT coefficient. The goal of the test is to demonstrate that a correlation peak would exist when the test image is aligned with the eigenvector or eigen image. The head scan image is shifted at 20 pixel increments, 300 pixels to the right and to the left.

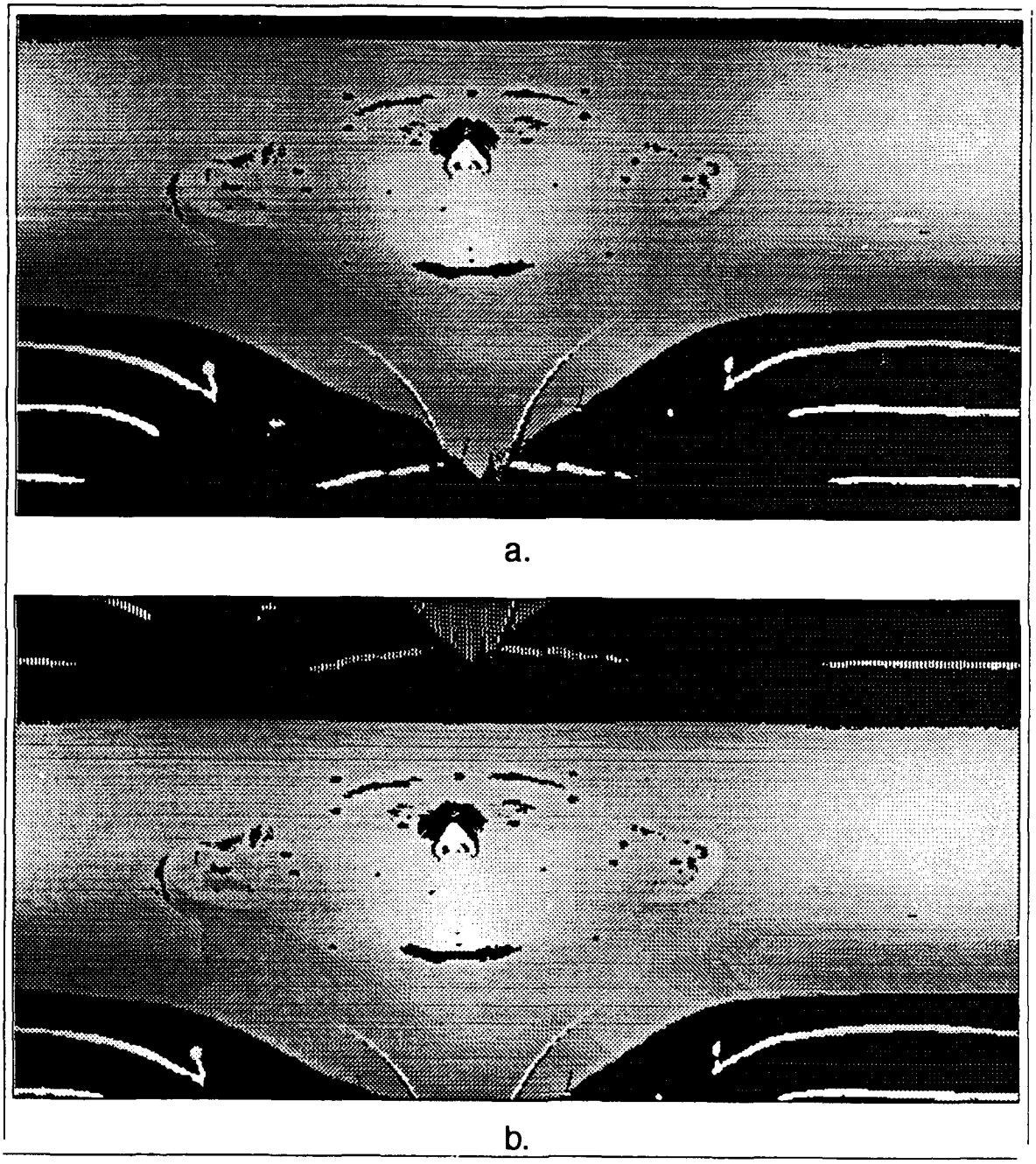


Figure 3.12. Two laser scans of the head. The top is the original 512 by 256 pixel image, and the bottom is the shifted to axis system image.

Simultaneously, the image is shifted at 10 pixel increments, 90 pixels up and down. Once the peak is found, the test image is again shifted at one pixel increments to find a precise peak.

As a means of comparison, the FFT correlation between the head scanner image and the first eigen image is calculated. A correlation peak implies that the head scan is aligned with the eigen image. The results of this test are presented in Chapter 4.

Conclusion

This chapter first presented various KLT and support algorithms developed for this thesis. The chapter then presented the methodology used to investigate the reconstruction of images using the KLT with both large and small population training sets. Thirdly, the chapter provided the methodology used for face recognition using the KLT and FFT. The last three sections presented the methodology used to investigate a face finder, a facial feature communicator for the non-vocal, and a KLT based axis system. The results of the methodology are presented in the next chapter.

IV. Results

Code Development and Testing

In order to test the two main algorithms, kl_transform2.c and recon.c, the following test is devised. The first algorithm, kl_transform2.c, performs the Karhunen-Loève Transform (KLT). The kl_transform2.c training set includes nine instantiations of the same face, Figure 4.1 a. The nine faces differ only by the addition of various amounts of Gaussian noise. After program execution, the resulting eigenfaces, Figure 4.1 b – d, demonstrate that most of the information or energy is in the first eigenface. The mean square error

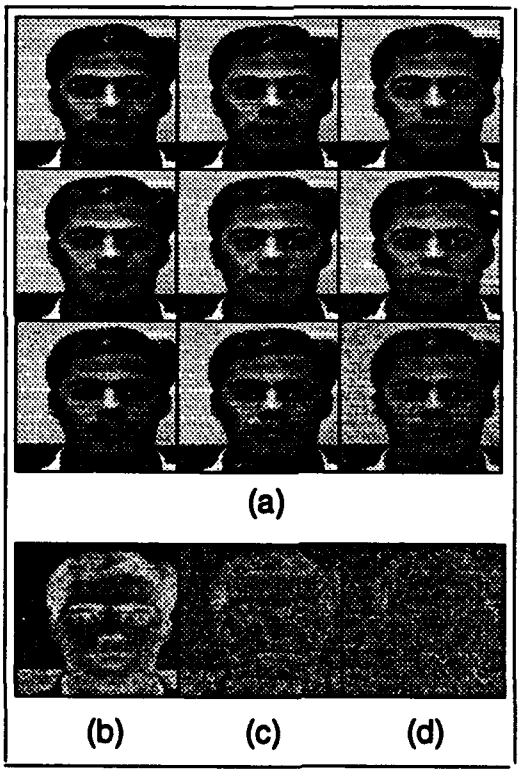


Figure 4.1. (a) The KLT training set of nine instantiations of same face with various amounts of Gaussian noise (b) first eigenface of the nine images (c) second eigenface (d) third eigenface

(MSE) plot of Figure 4.2 also indicates that a major part of the energy or information is in the first eigenvector. Since all of the images are basically the same face perturbed by Gaussian noise, it is reasonable to conclude that only one eigenface is needed to represent

all nine faces. This example confirms the operation of the code and more importantly provides insight into the KLT. The first eigenface represents most of the face energy whereas the second and third eigenfaces represent the orthogonal projections of the changes induced by the addition of Gaussian noise (7). Figure 4.4 demonstrates a two dimensional eigenspace where it can be seen that most of the energy in the faces can be projected onto the first eigenface and the changes introduced by the Gaussian noise can be taken into account by the second orthogonal eigenface.

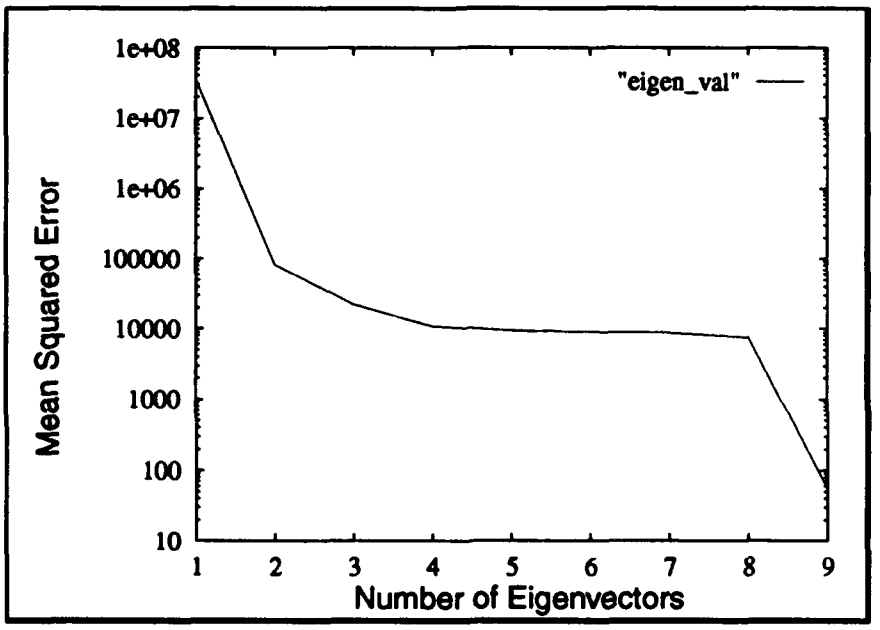


Figure 4.2. Mean squared error of reconstruction of Figure 4.3 a - c

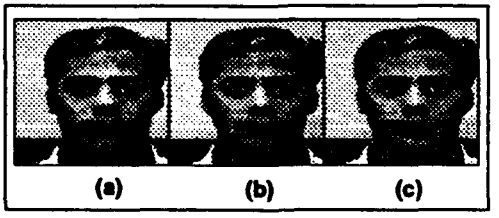


Figure 4.3. (a) Reconstruction using one eigenface (b) using two eigenfaces (c) using all nine eigenfaces

The second algorithm test of recon.c utilizes the eigenfaces from kl_transform2.c. A face is selected from the nine training faces and is provided as an input to recon.c. The

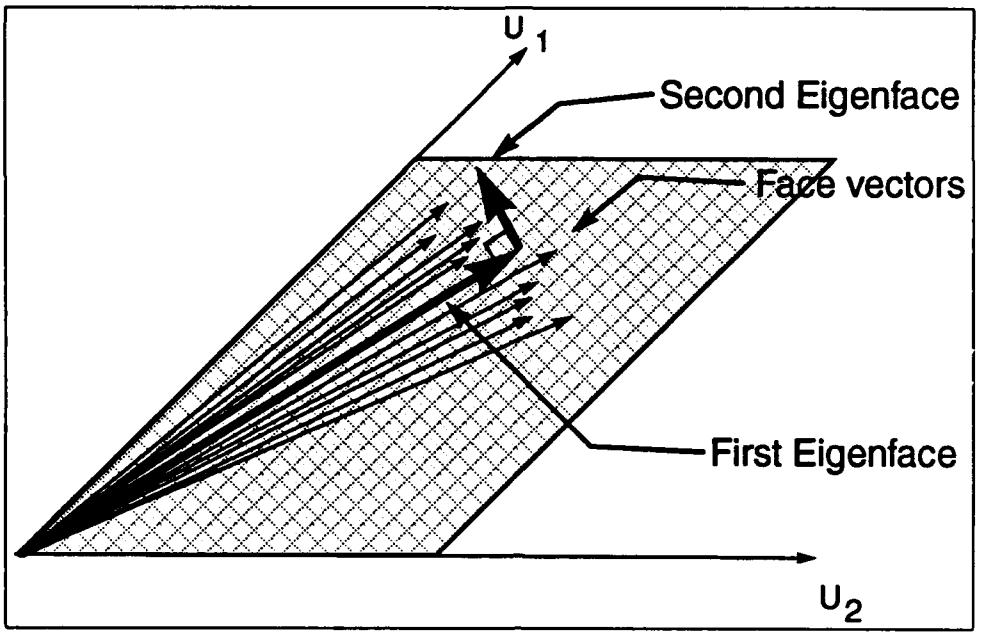


Figure 4.4. An example of a two dimensional eigenspace where all nine images are the same face with various amounts of Gaussian noise. The first eigenface, shown as a bold vector, represents most of the energy in the faces, and the second eigenface accounts for the Gaussian perturbations.(7)

program recon.c generates the KLT coefficients and reconstructs the original image as shown in Figure 4.3. As indicated earlier, the MSE plot of Figure 4.2 indicates that the reconstruction should only need one eigenface. Figure 4.3 confirms that the reconstruction is acceptable using only one eigenface.

KLT Reconstruction

Small Population Reconstruction. The test setup, Figure 4.5, consists of six training faces, each scaled from a 320 by 480 image to 256 by 256 pixels. The program C_late5.c of Appendix A aligns the images. A Gaussian window is applied with gwind.c of Appendix A and a second alignment with C_late5.c concentrates on the eyes, nose, and mouth of the images. As discussed in Chapter 2, the second alignment improves the reconstruction and recognition performance. Using kl_transform2.c, the KLT creates the average face, eigenvalues, and six eigenfaces.

Figure 4.6 indicates 10% MSE reconstruction results when using three eigenfaces.

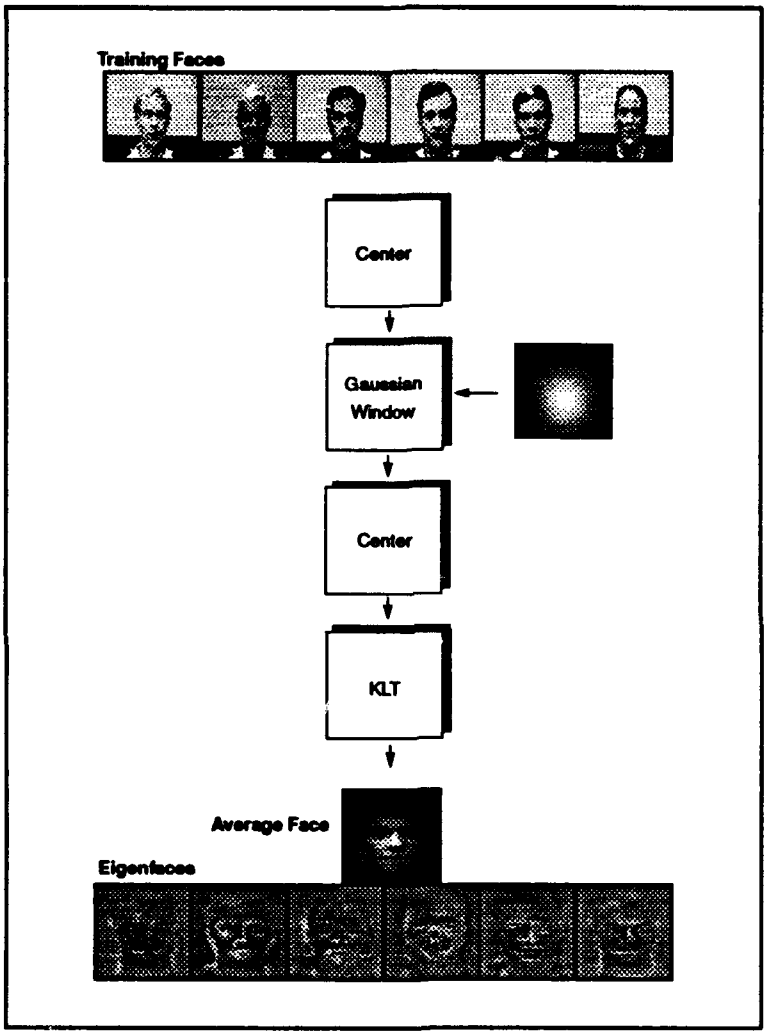


Figure 4.5. The KLT process consists of first defining a training set, centering the images with C_late5.c, windowing with gwind.c, centering once again, and lastly performing the KLT with kl_transform2.c. The result is the average face, eigenface, and eigenvalues (not shown). Note all source code is listed in Appendix A.

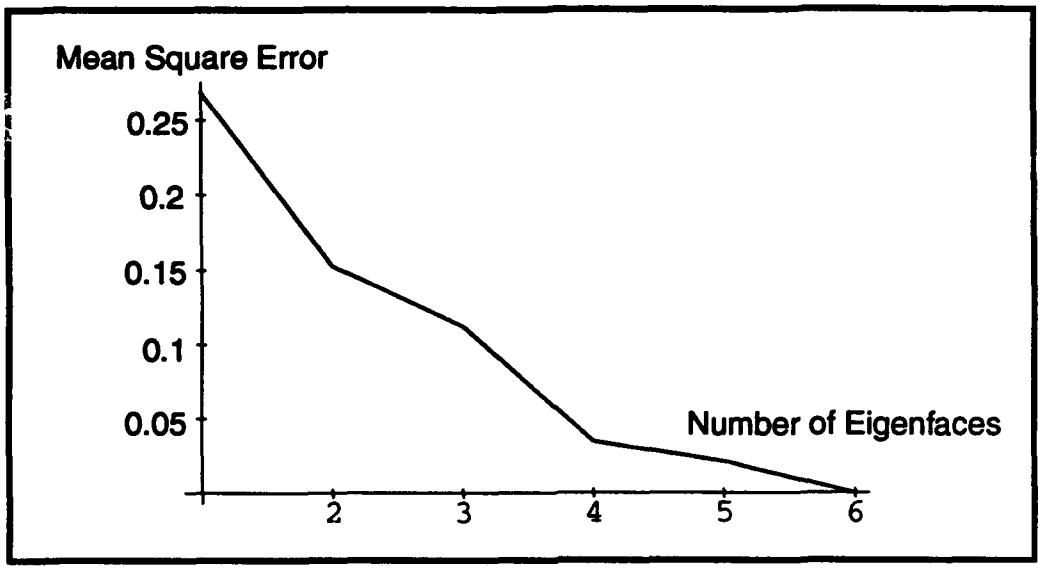


Figure 4.6. Reconstruction MSE of small population

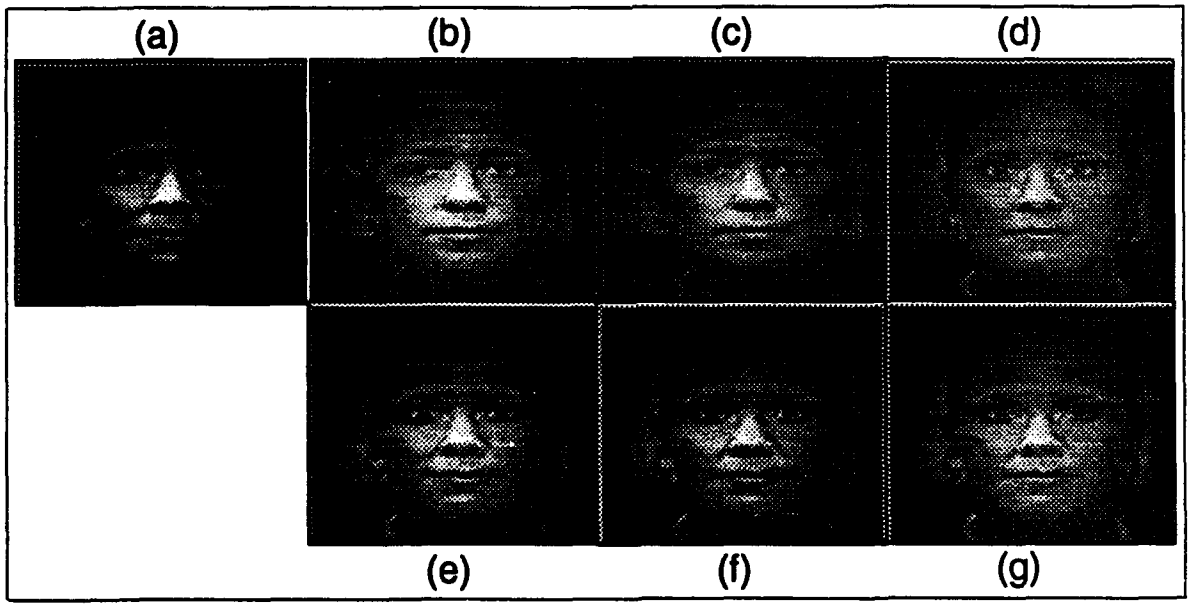


Figure 4.7. Small population reconstruction using (a) original image; reconstruction using (b) one eigenface (c) two eigenfaces (d) three eigenfaces (e) four eigenfaces (f) five eigenfaces (g) six eigenfaces

An image is first reconstructed using one eigenface, Figure 4.7 b, and the result has no resemblance to the original of Figure 4.7 a. By the addition of the third eigenface, the reconstruction of Figure 4.7 d appears acceptable. The addition of eigenfaces four through six, Figure 4.7 e-g, adds very little to the quality of the reconstruction.

The KLT permits the reconstruction of any of the six faces using only three of the eigenfaces. Representing an image by only three coefficients represents a dramatic compression of approximately 21,000:1. In a communications sense, instead of transmitting 256 by 256 pixels, only three coefficients need to be transmitted provided that both sender and receiver have copies of the first three eigenfaces and average face. However, this significant compression is costly. First, the compression is limited to a small population of images, six in this instance. Second, a significant amount of processing is necessary to determine the eigenfaces. Additionally, both sender and receiver must have copies of the eigenfaces. The pre and post processing impact would be less worrisome if the population could be increased. Phase two of KLT reconstruction consists of larger population reconstruction experiments.

Large Population Reconstruction. The first test evaluates a KLT training set of three 128 by 128 pixel images per subject and a population of 27 subjects. The images are Gaussian windowed with Figure 3.5 a which has the following parameters: x mean=65, x variance=35, y mean=55, y variance=50. This test evaluates the size of the training set and reconstruction quality of images in and out of the training set. The reconstruction, using 20 eigenfaces, is shown in Figure 4.8. The first column is the original image. The second column is the reconstruction of a KLT training set image. The third column is an image of the subject not in the KLT training set. The results indicate that the KLT reconstruction for a large population is much worse than the small population reconstruction. Column three indicates that the KLT can reconstruct images not in the training set provided the subject is in the KLT training set.

The second test evaluates the effects of the Gaussian window on the reconstruction. The window is tightened as in Figure 3.5 b to: x mean=65, x variance=25, y mean=55, y variance=30. The resulting quality is unaffected by window changes, Figure 4.9.

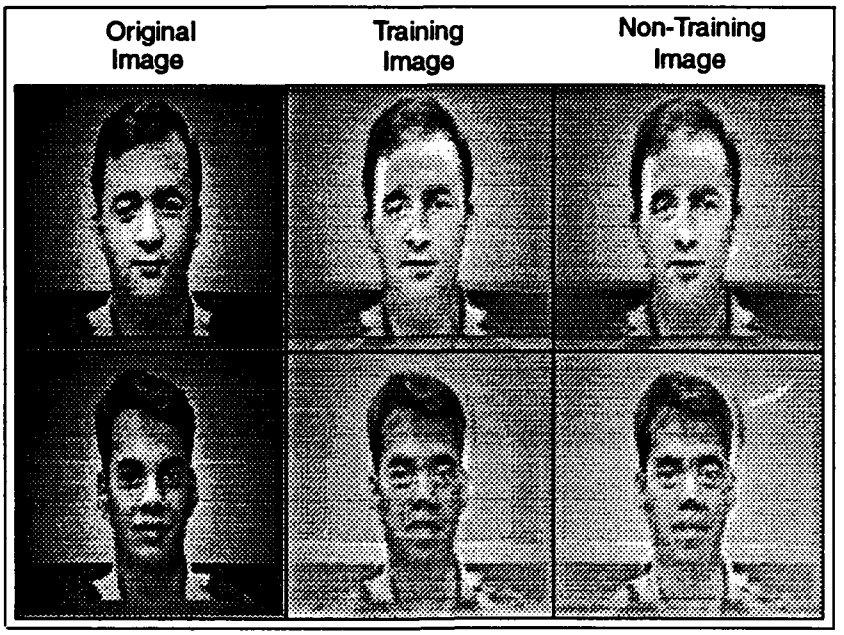


Figure 4.8. Test 1: reconstruction with 20 eigenfaces of large population, 81 training images

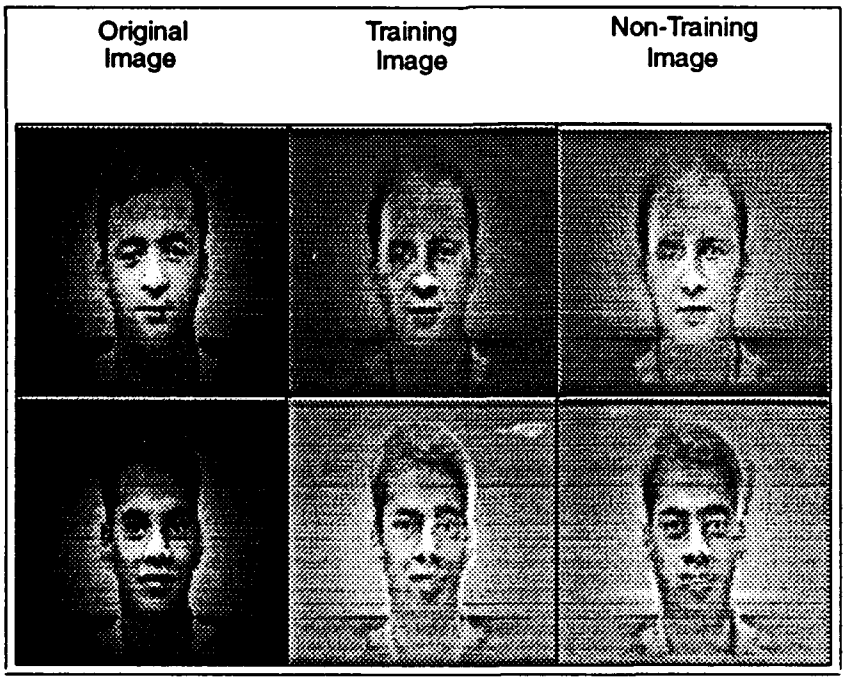


Figure 4.9. Test 2: reconstruction with 20 eigenfaces of large population, 81 training images, and tight Gaussian window

The third test to improve reconstruction quality varies the amount of training faces used per subject from three to one, for a total of twenty-seven pictures. The faces are Gaussian windowed as in the first test with: x mean=65, x variance=35, y mean=55, y variance=50. The results of Figure 4.10 are similar to those of the first test, but a close examination of the images reveals a marginal decrease in reconstruction quality. For example, note the loss of detail around the eyes.

The fourth test, Figure 4.11, demonstrates the use of a smaller set of eighteen training subjects with the following Gaussian window: x mean=65, x variance=25, y mean=55, y variance=30. Once again, marginal decrease in reconstruction quality results.

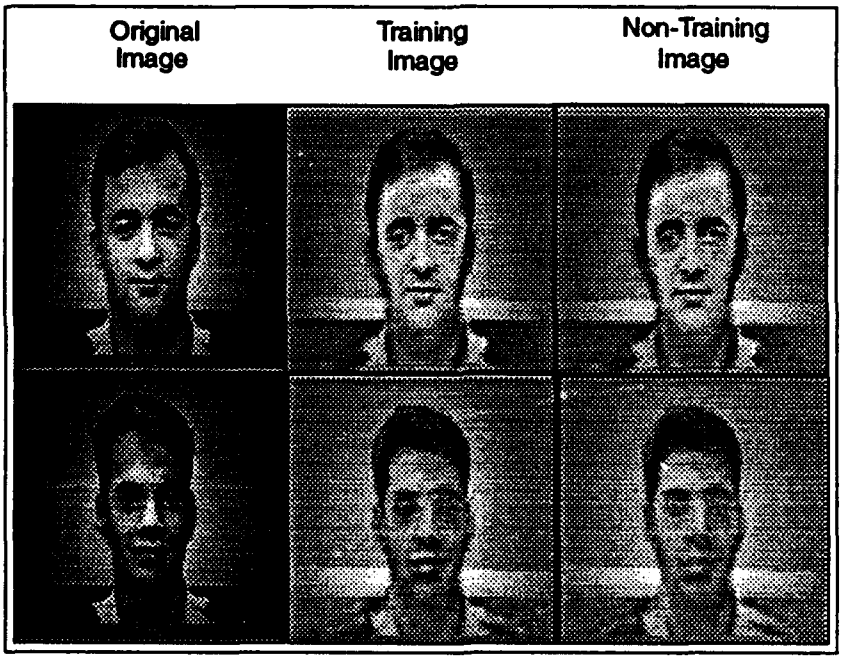


Figure 4.10. Test 3: Reconstruction with 20 eigenfaces of large population, 27 training images, and only one image per subject

The fifth test, Figure 4.12, demonstrates the effects of not using Gaussian windowed images. Without a Gaussian window, the alignment of C_late5.c performs poorly. A poor alignment results in the blurred reconstruction seen in Figure 4.12. This test demonstrates the importance of centering or aligning the images, and it proves the shift sensitivity of the KLT.

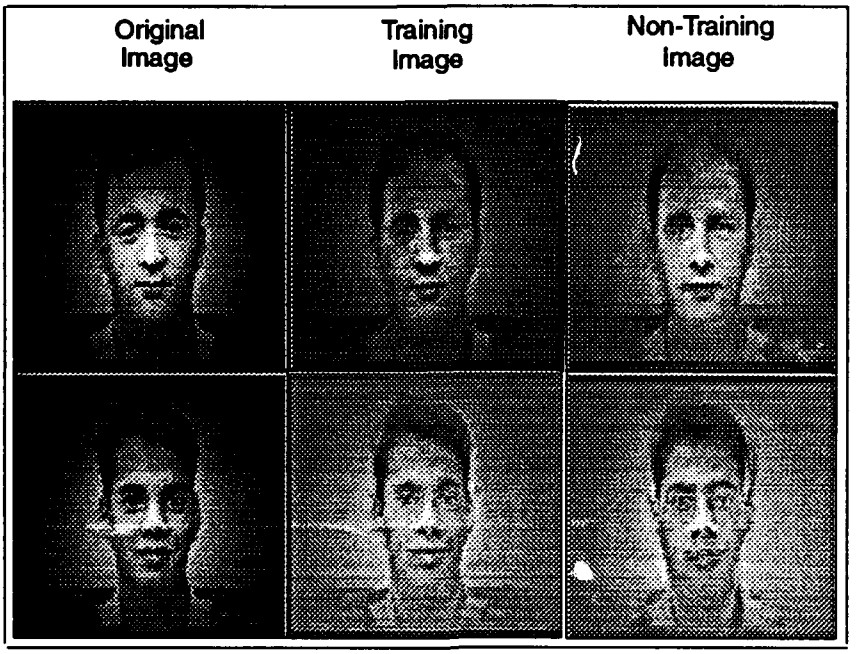


Figure 4.11. Test 4: Reconstruction with 20 eigenfaces of large population, 18 training images, and a tight Gaussian window

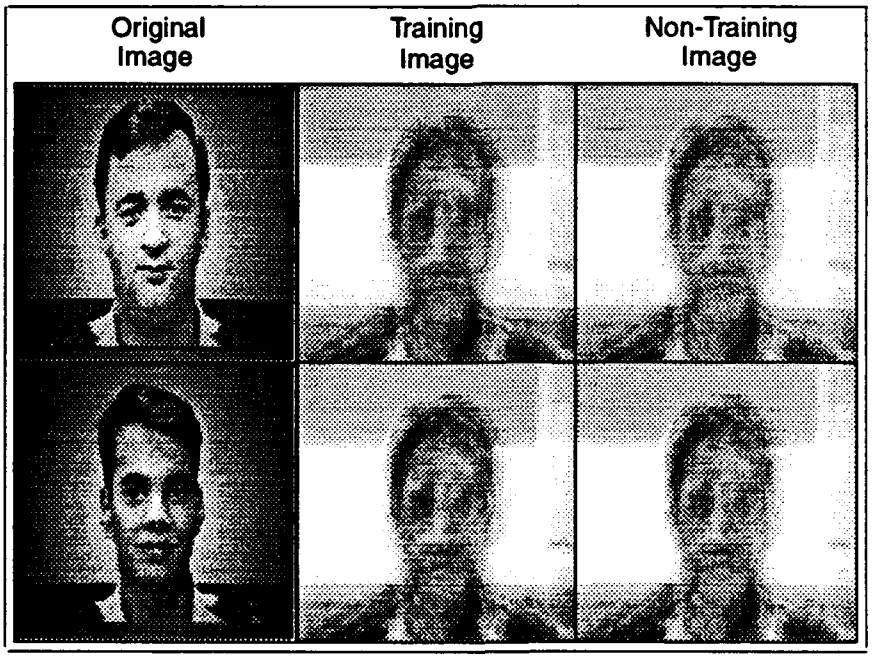


Figure 4.12. Test 5: Reconstruction of non-Gaussian windowed images

The sixth test, Figure 4.13, demonstrates a reconstruction of a subject not in the KLT training set. A training set of 81 images was used to create the KLT basis set. Note that the reconstruction has no resemblance to the original image. This result is somewhat disturbing because it indicates that the KLT did not generalize for reconstruction. In other words, subjects not in the KLT training set can not be reconstructed.

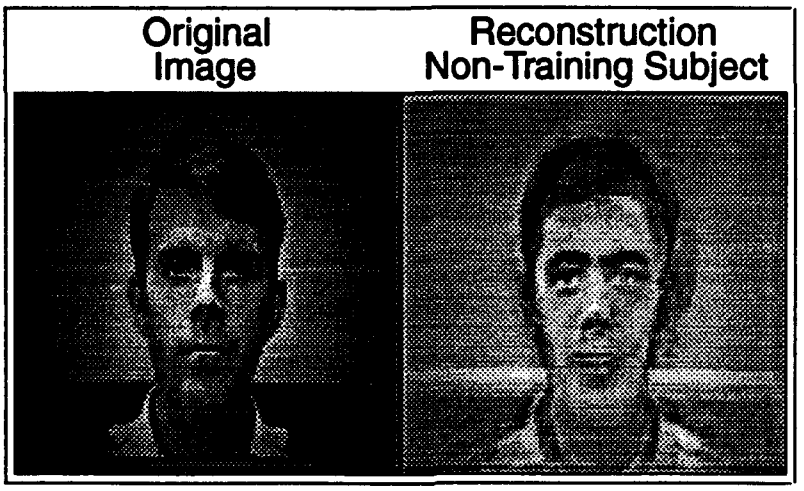


Figure 4.13. Test 6: Reconstruction using 20 eigenfaces of subject not in the KLT training set.

Overall, the best reconstruction appears to be that of test one, a large training set with three images per subject in the KLT training set. In reality, the quality of all four reconstructions is good and is extremely better than previous KLT image reconstruction of faces (27). For the first test, which is the best reconstruction, the compression is 16384:20 or approximately 800:1 for 81 training images. This is a significant image compression figure. The image compression comes at the cost of constrained population, reconstruction quality, and computational overhead.

The reconstruction results are enlightening and valuable from an image processing point of view, but the true goal of the research is face recognition. A transform that reconstructs images very well does not necessarily make a good image recognition transform, and vice versa. Therefore, the most important results of the research, face recognition and classification, are to follow.

KLT and Recognition

Single Person Verification. Single person verification consists of recognizing a persons identity. For instance, if a subject claims to be John Doe, does his face match John Doe's? This problem is a two class problem. Class one consists of the KLT coefficients of sixteen different subjects. Class two consists of the KLT coefficients of sixteen different images of the test subject. A Backpropagation Neural Network (BPN) with two layers and two hidden nodes trains to differentiate between the two classes. To test the classification accuracy of the BPN, each training vector is held out once. The results, averaged over ten trails and shown in Figure 4.14, demonstrate the accuracy versus the number of features. Five KLT coefficients resulted in 92% recognition accuracy.

Using the same images, the single person verification problem is run again with a Euclidean distance classifier. The results shown in Figure 4.14 illustrate the recognition accuracy versus the number of features. Three KLT coefficients resulted in 93% recognition accuracy.

Note that the probability of error of the Euclidean distance, also called Nearest Neighbor (NN) classifier, approximates a Bayesian classifier as the number of test and prototype vectors approaches infinity (25:83). Additionally, Ruck proves that the BPN approximates a Bayesian classifier which indicates that both the BPN and NN classifiers are Bayesian (17). The results of Figure 4.14 confirm this result.

Both tests indicate that for single person verification, a small number of coefficients provides relatively accurate verification.

Multi-person Recognition. Using fifty-five video taped subjects, six 128 by 128 pixel frames of each person are taken. One frame of each subject is used in the KLT training set.

Before feature extraction, the six images are divided into a test and prototype set. The test set consist of two images which will be compared to the remaining four in the prototype set. With the KLT coefficients of both the test and prototype sets provided by

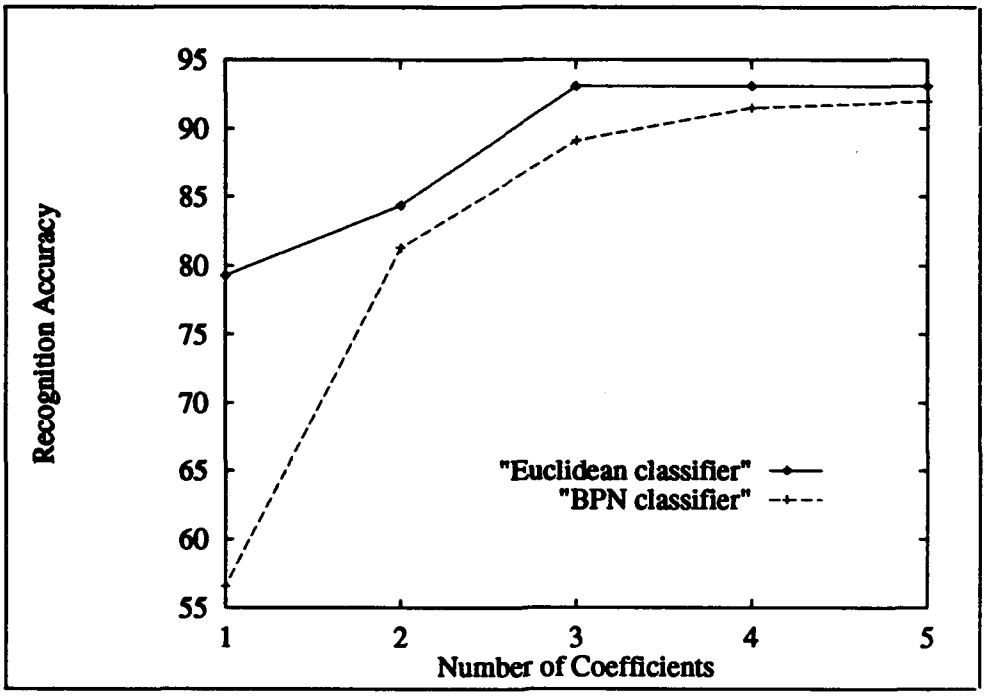


Figure 4.14. Results of single person recognition versus number of eigenfaces for both BPN and Euclidean distance classifiers

the routine recon.c, a Euclidean distance classifier, find_min.c, is employed to compare the test set to the prototype set. A minimum distance implies a match.

The multi-person test first varies the number of KLT coefficients. Figure 4.15 demonstrates the recognition accuracy versus the number of coefficients. Ninety-five percent recognition accuracy is achieved with only sixteen coefficients. Interestingly enough, increasing the number of coefficients actually decreases accuracy. This can be readily explained by pointing out that the lower eigenvalued eigenvectors have less energy and therefore these coefficients introduce more noise than information into the classification (7).

A second test is performed on the same subjects. The test set-up is identical to the previous test except forty images are used in the KLT training set. Figure 4.16 demonstrates the recognition accuracy versus the number of coefficients used. In this case, ninety-four percent accuracy is achieved using sixteen coefficients.

Using the Fast Fourier Transform (FFT), the same set of Gaussian windowed images

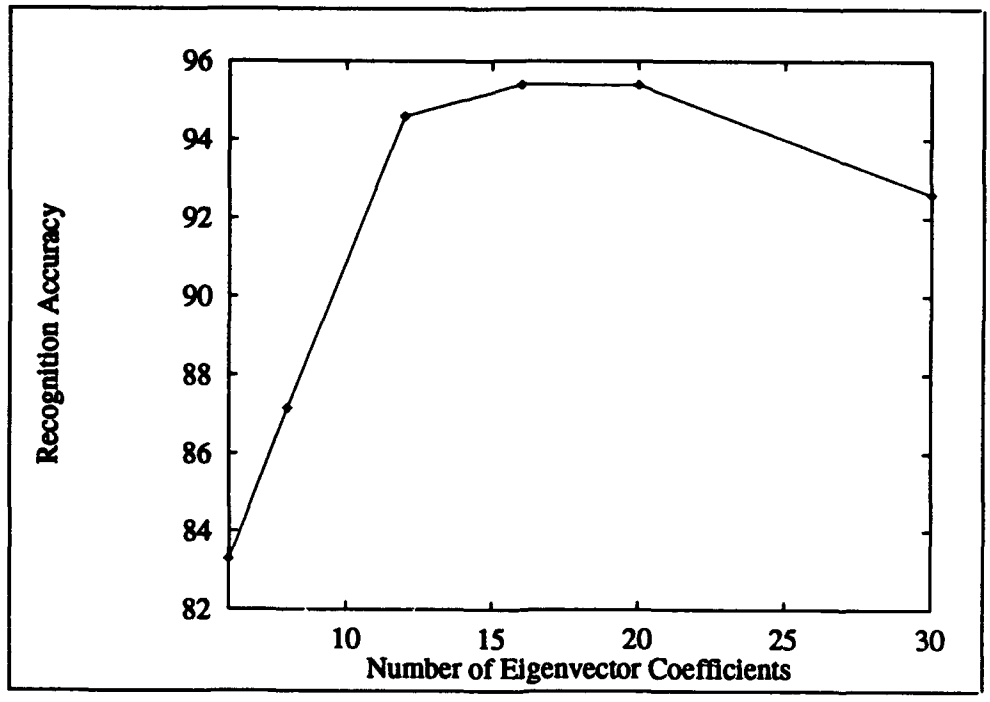


Figure 4.15. Results of multi-person recognition versus number of eigenfaces for 55 subjects where each subject is included in the KLT training set

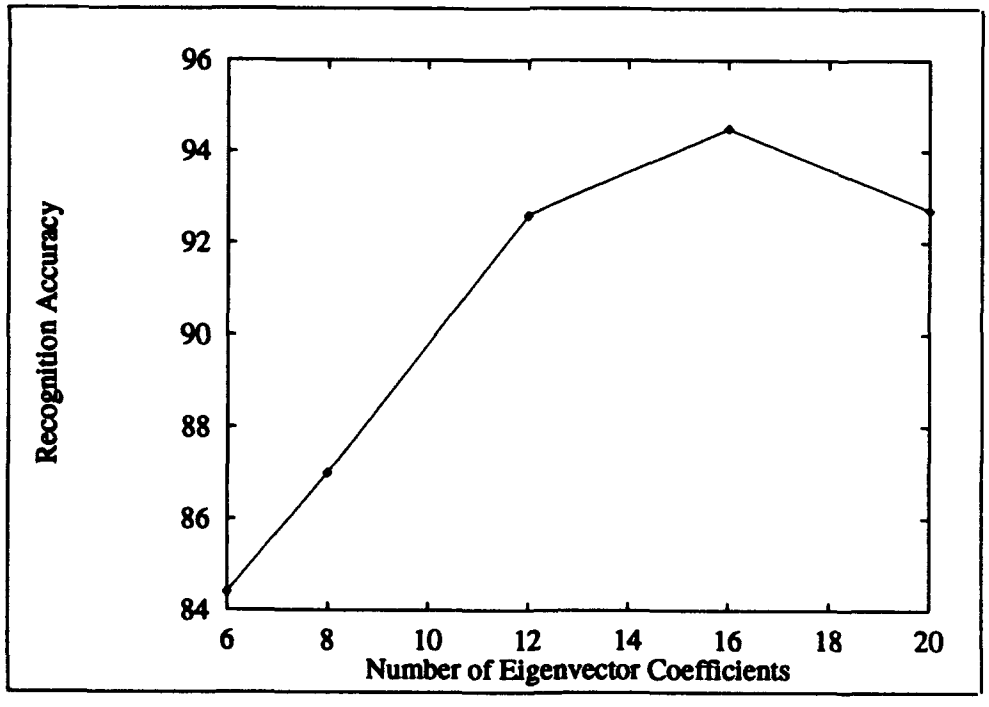


Figure 4.16. Results of multi-person recognition versus number of eigenfaces for 55 subjects where only 40 of the subjects are included in the KLT training set

are transformed, and the FFT coefficients are calculated. After Euclidean distance classification, recognition accuracy versus the number of FFT coefficients is determined, Figure 4.17.

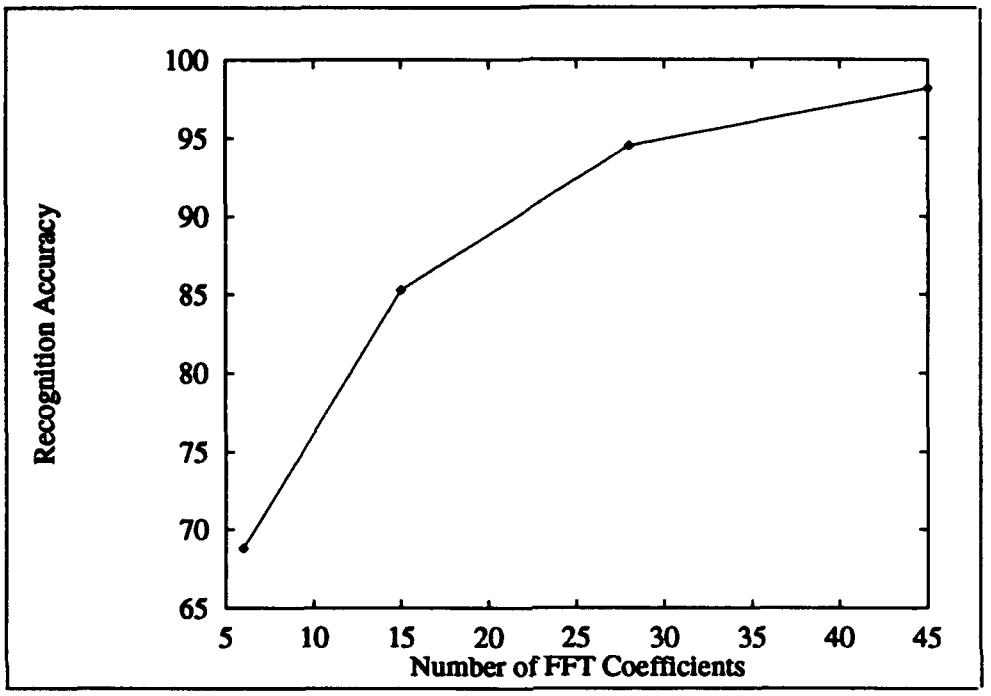


Figure 4.17. Results of multi-person recognition versus number of FFT coefficients for 55 subjects

A comparison of the KLT and FFT is shown in Figure 4.18. The KLT clearly out performs the FFT for a lower number of coefficients. For example, the FFT needs 27 coefficients to reach 95% recognition accuracy where as the KLT only requires as few as 16 coefficients. The FFT recognition performance approaches the performance of the KLT as the number of coefficients is increased.

Finding Faces in Face Space

Finding a face in a scene is the first step in any pattern recognition system. The concept of using the KLT for segmentation is evaluated. The first segmentation approach scans a scene. Each sub-image scanned is projected into face space. If the Euclidean distance of the sub-image KLT coefficients is less than a certain threshold, a face is probably

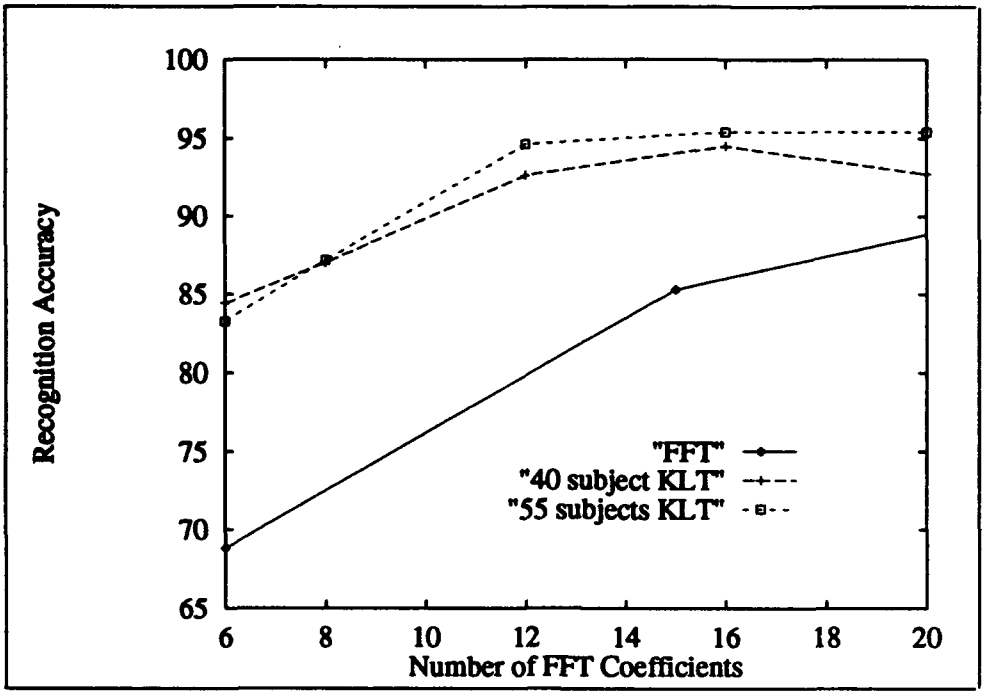


Figure 4.18. Comparison of multi-person recognition versus number of coefficients for FFT, KLT using 55 training subjects, and KLT using 40 subjects

present and the sub-image is segmented from the scene. The results of this experiment indicate that faces can not be found using this technique. The technique is very sensitive to head size and lighting. Worst of all, the technique is computationally impractical.

The second segmentation approach utilizes the FFT of the scene and of the eigenfaces. A correlation is performed beween the scene and eigenfaces to determine the location of the face in the scene. The results of this test indicate that faces can not be found with this technique. A correlation peak does occur at the location in the scene corresponding to the face, but the peak is not a global extrema and many peaks have to be tested before a face is confirmed.

Although both of the previous techniques performed poorly because of size and lighting sensitivity, the AFRM moving target segmenter has been shown to perform over various lighting and image size conditions (10). The AFRM uses motion to segment the image, then the image is normalized for brightness and size. With AFRM preprocessing, the image is properly normalized for a KLT based recognition system.

Another face segmentation technique demonstrated by Tarr, finds human eyes in a scene using a neural network (23). This neural network based segmentation can also be used instead of moving target segmentation. Tarr has also demonstrated a Karhunen-Loève Neural Network which implies that a totally neural based solution can implement many of the techniques demonstrated in this thesis (24).

Karhunen-Loève based Facial Feature Communicator for the Non-vocal

The test images are processed and classified as in Figure 3.10. The KLT coefficients of the test images are compared to the KLT coefficients of previously processed prototypes. The nearest neighbor classifier determines the class of the image, i.e. 'yes' or 'no'. The system was able to determine 'yes' tongue out or 'no' mouth open with 100% classification accuracy.

The test imposed several limitations on processing. First, the KLT is sensitive to size. So a real system would have to fix the distance between the user and camera. The second limitation is segmentation and alignment of images. This limitation can be solved in an operational system by utilizing positive feedback. Simply indicate non-alignment to the user and ignore all test images that do not fall within some KLT coefficient distance threshold. A non-alignment can easily be indicated on a computer screen and when the user is aligned properly the 'yes' or 'no' indications can be processed.

The KLT successfully determined 'yes' or 'no' in a small test set of four images. The ability to differentiate between these simple expressions indicates that the KLT may be used as a communications interface for the disabled. In other words, this concept can be used by a disabled person to manipulate a computer menu system that controls other functions. Similarly, the binary 'yes' or 'no' scheme can easily be coded into morse code permiting a simple binary to text conversion. The methodology used here should be readily employable at frame rate because the KLT coefficients are computed with only one subtraction and two dot products between the input image and each of the two eigenvectors.

Karhunen-Loève Axis System

The purpose of this test is to demonstrate correlation in Karhunen-Loève space and to develop an axis system or technique to align laser scanner head images. The first eigenvector, Figure 4.19, is the basis vector with the greatest contrast or variance among all the images. Therefore, an eigen based axis system takes into account the largest variances in all of the training set population.

Figure 4.19. An example of an eigenvector resulting from a training set consisting of head scanner data.

The graph of the projections is shown in Figure 4.20. The projections demonstrate that as the image approaches the origin of the axis system, the projection value approaches a peak. The peak corresponding to the axis center is global. After the extrema is found, the image is shifted at one pixel increments to find the precise peak. The projections at one pixel resolution about the axis origin are shown in Figure 4.21. The peak at (5,-32) corresponds to a shift of the test image 5 to the right and 32 down.

To compare the result of the Karhunen-Loève correlation with the FFT based correlation, the same image is correlated with the first eigen image which was used in the previous test. The FFT based correlation indicated a correlation peak at (5,-32). This is in agreement with the results obtained by the previous Karhunen-Loève approach.

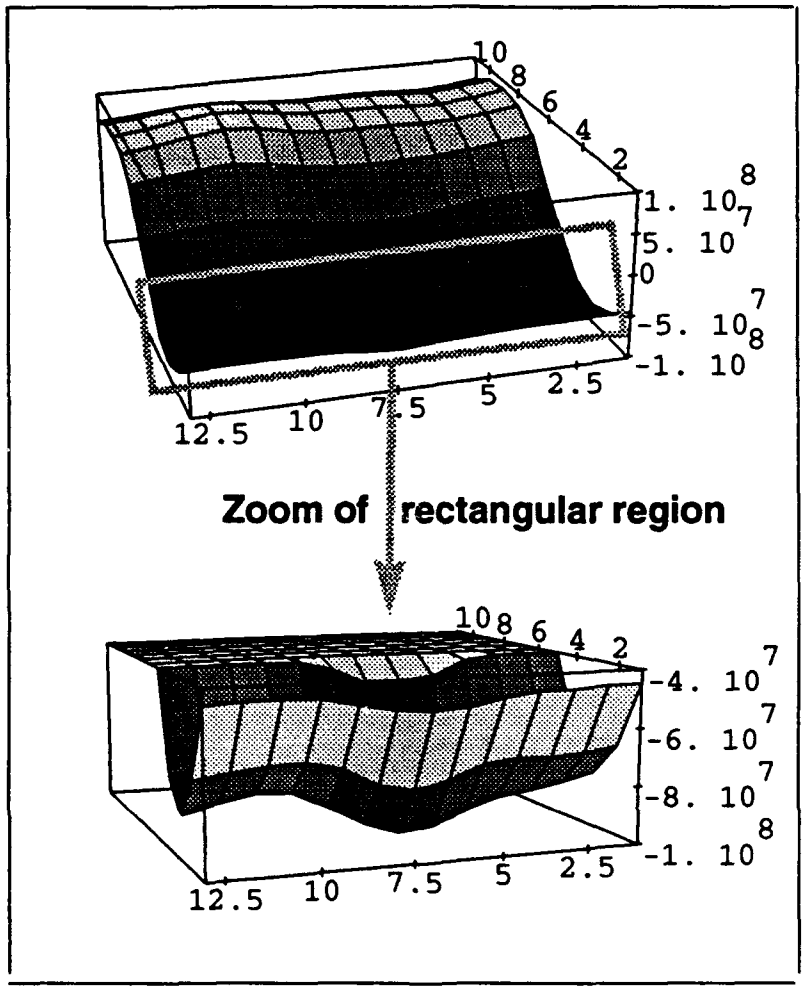


Figure 4.20. The top graph shows the projection of the shifted image into Karhunen-Loève space using only the first eigenvector. The bottom graph details the peak location which indicates alignment between the shifted test image and eigenvector.

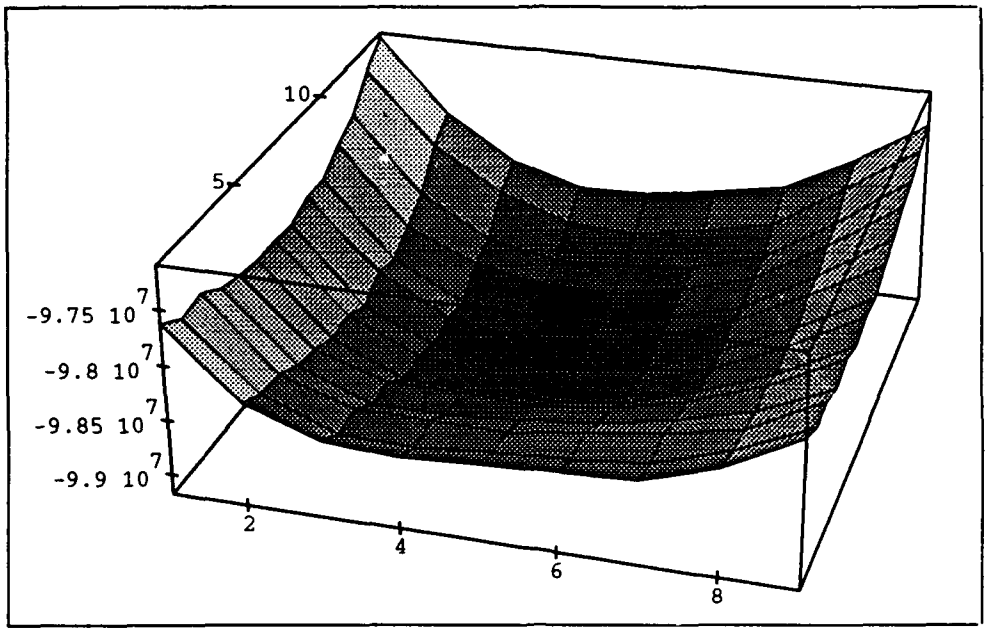


Figure 4.21. Projection of the shifted test image with one pixel resolution into Karhunen-Loève space using only the first eigenvector. The minimum represents the correlation peak indicating alignment between test image and eigen image.

The results indicate that correlation can be performed with the Karhunen-Loève coefficients, and the coefficients can be used to align images. The similarity in the location of the correlation peak with Fourier based correlation lends confidence to the result. Recall that the first eigenvector represents an image with the greatest amount of variance or information in its pixels. The eigenvector provides the best image to correlate against because the eigenvector represents the largest variance or worst case change. In short, the first eigenvector provides the best correlation with the largest population. This is the justification for the use of the eigen images as a basis for an axis system. Further evaluation needs to determine the robustness of an eigen based axis system over a larger population of test images. The use of the eigen images with either an FFT or KLT based correlator seems to be equivalent.

The significant results of this research are the demonstration of correlation in Karhunen-Loève space and its equivalence to traditional FFT correlation. Second, the use of eigen images as a basis for an axis system represents a logical choice over traditional feature based

techniques, e.g. an axis system based on the planes traversing the nose and ears. Further research will evaluate the results over a larger data set and evaluate the use of a conjugate gradient approach to Karhunen-Loève correlation techniques which would eliminate the need for as many shifts to find the global extrema.

Conclusion

The most significant result of this chapter is the demonstration of the recognition superiority of the KLT over the FFT or the AFRM feature based approach. With only 16 coefficients, the KLT provided 95% face recognition accuracy for a population of 55 subjects. The 40 subject test demonstrated KLT generalization for recognition. This is significant because the poor reconstruction of non-training subjects, Figure 4.13 , indicates that for reconstruction purposes the KLT did not generalize. In short, this demonstrates that the KLT is good for face recognition but not sufficiently adequate for reconstruction.

This chapter first presented the results of code testing and then presented the results of the thesis. The chapter then presented the reconstruction results for both large and small population training sets. Thirdly, the chapter provided results of face recognition using the KLT and FFT. The last sections presented the results the KLT based face finder, the facial feature communicator, and the Karhunen-Loève axis system. The significant results and conclusion are presented in the next chapter.

V. Conclusions

Introduction

The purpose of this thesis is to find good features for machine recognition of human faces. This thesis investigated the Karhunen-Loève Transform (KLT) and demonstrated its performance. A summary of the significant results follows.

Summary of Significant Results

This thesis developed efficient KLT algorithms in ANSI C. The algorithms, which are documented in Appendix A, implemented a Karhunen-Loève approximation that simplified the solution to a simultaneous equation from degree n^2 to M. For example, for 128 by 128 pixel images in a KLT training set of 40 images, $n^2 = 16384$ and $M = 40$. This approximation which is only valid if $n^2 >> M$ greatly simplifies and hence speeds the calculation of the KLT (27).

This thesis demonstrated the image coding and compression capabilities of the KLT. Large and small populations of subjects were used in the KLT training set. For a small population of six subjects, the reconstruction quality of the KLT encoded image was as good as the original image. The KLT was able to compress a 256 by 256 pixel image into three coefficients. This represents a compression of approximately 21,000:1. For a larger population of 55 subjects, the reconstruction of the 128 by 128 encoded images was reasonably good with only twenty coefficients. This represents a compression of approximately 800:1. Both of these dramatic compressions come at the expense of a constrained subject population and pre and post processing requirements.

The most significant achievement of this thesis is the dramatic face recognition accuracies obtained using the KLT coefficients. The KLT coefficients provided 93% single person verification with as little as three coefficients. For multi-person recognition, the KLT coefficients achieved 95% accuracy with as little as 16 coefficients and a population of 55 subjects. In comparison, the FFT with the same test data achieved 85% with 16 coefficients. These results are markedly improved over the inconsistent AFRM accuracy which varied from 56% to 90% for a population of 50 subjects and 15 coefficients. The

KLT outperforms the FFT and the AFRM. The KLT is sensitive to changes in head size and position, but the FFT based techniques also suffer from this problem.

This thesis demonstrated that a transform may be good for recognition but not adequate for reconstruction. The Karhunen-Loève Transform demonstrated very good recognition accuracy for subjects out of the KLT training set. This is significant since the KLT provided poor reconstructions of out of training set subjects.

Another significant result of this effort is that the software written for faces can also be easily used for any other image recognition application. The software written for this thesis was utilized by Singstock to recognize tanks, trucks, jeeps, and towers in forward looking infrared imagery (20). Singstock developed eigentanks, eigentrucks, eigenjeeps, and eigentowers as a basis set. The eigen images were successful in classifying tank, truck, jeep, and tower. Figure 5.1 displays a Singstock eigentank.

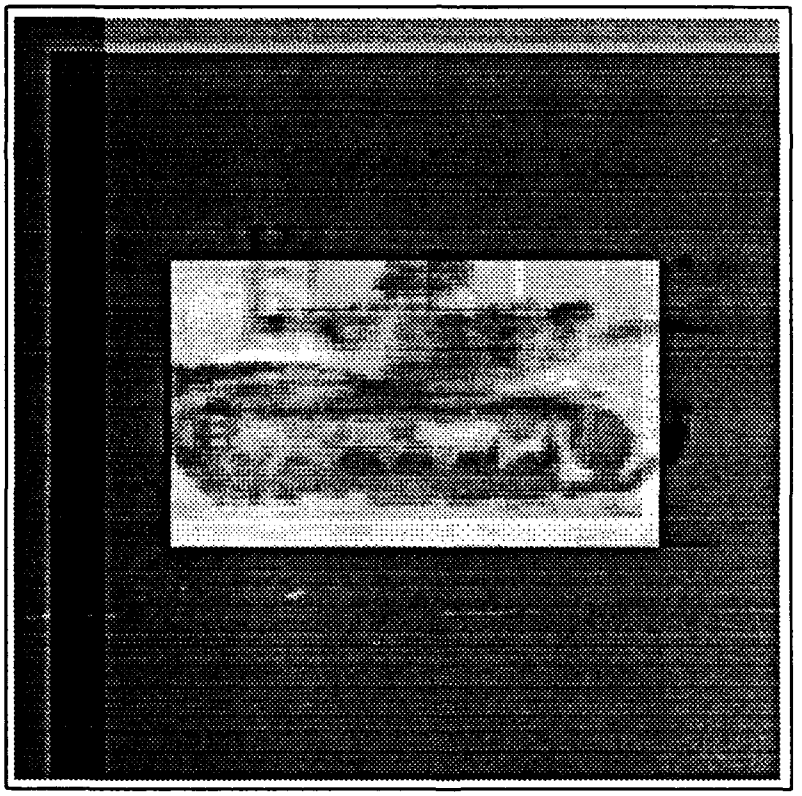


Figure 5.1. The Singstock eigentank used to recognize tanks.

A related application to face recognition is facial expression recognition. A test

was conducted to differentiate facial expressions. The motivation behind the test was to evaluate the possibility of developing a facial feature communications interface for a child with cerebral palsy. The child is severely disabled and can not speak. The child does have control of facial muscles, therefore differentation between tongue out and mouth open provides a means of communication. The Karhunen-Loève system was able to determine 'yes' tongue out or 'no' mouth open with 100% classification accuracy.

The last related application is in the anthropometric community. The anthropometric community needs an axis system for head laser scan data. The head laser scan data is basically an unwrapped picture of the head. Before anthropometric data can be extracted, the image must be centered to simplify the measurement process. This thesis evaluated the use of a Karhunen-Loève based axis system. The results indicate accurate placement of laser scanner heads with the KLT axis system.

Conclusions

In conclusion, this thesis demonstrated superior face recognition using the KLT. The KLT approximation implemented in this thesis has removed the computational burden associated with Karhunen-Loève. Also, note that the results using a simple first nearest neighbor classifier were 95%. To attain even greater accuracy a more elaborate classification scheme such as a neural network could be implemented. Additionally, a multiple look or multiple frame collection of each subject would also improve the recognition accuracy above 95%.

The primary limitation of the Karhunen-Loève is scale and position sensitivity. There are two solutions to this problem. First, the AFRM segmenter sucessfully provides scale, contrast, and lighting normalized images. Therefore, a Karhunen-Loève based recognition system could utilize the AFRM segmenter. A second approach is to transform the image into a scale, position, and rotation invariant space as demonstrated by Kobel and Martin (8).

Lastly, the software written for this thesis is flexible and the techniques developed here have been applied to problems associated with tank recognition, anthropometry, and

communications for the severely disabled.

Future Areas of Research and Recommendation

The prime recommendation for future work would be to integrate the work done on this thesis with the AFRM. This would take advantage of the AFRM camera and segmenter and utilize the Karhunen-Loève techniques developed here. Additionally, a fruitful area of research would be to implement a total neural network approach to the face recognition problem. Finally, the human face has a unique color signature and future research should investigate the use of color cameras for segmentation and recognition.

General Summary

This chapter provided a summary of the significant results as well as recommendations for future research. Chapter 1 provided background and defined the problem, objectives, questions, methodology, standards, and scope of the research. Chapter 2 provided additional background and a review of current research. The methodology used to solve the problem was presented in Chapter 3. The results of the research were presented in Chapter 4. Appendix A lists all the code utilized for this thesis.

Appendix A. Thesis Source Code

This appendix contains a listing of all code developed and or used for this thesis. This code is presented as is, and no claims are made as to suitability for other applications. Portions of this code are copyrighted by the Air Force Institute of Technology and by Pedro F. Suarez. Such portions are indicated as such.

kl_transform2.c

```
/*************************************************************************
   NAME: kl_transform1.c
   INVOKED:
   DATE: 24 June 1991
   DESCRIPTION:
  SUBROUTINES CALLED:
   FUTURE MODIFICATIONS/BUGS:
*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>

#define SQ(A)    (A*A)

main(argc,argv)
int argc;
char *argv[];
{

FILE     *train, *facein,  *fout;
FILE     *face_avg, *tempfile, *fevex, *feval;
int      i,j, N, k, M, nrot, atoi();
float      **matrix(),  *vector(), **A, **A_trans, **u, **L, **v,  *average_face;
float     *d, temp, *mag;
void     free_vector(), free_matrix(), eigsrt(), jacobi(), mat_col_mag();
char     filename[81], h,l;


if (argc ≠ 4) {
   printf("!!! The command line should be !!!:\n\n kl_transform2 trainingfile
size #_in_train_set\n\n");
   exit(0);
}
/******** Set Up Files ************************/

if ((train = fopen(argv[1], "r")) == NULL)
```

```
{
printf("I can't open the training file");
exit(-1);
}
/****

    printf("!!!  Input total image SIZE(N):");
    scanf("%d", &N);
    printf("\n");
    printf("!!!  Input the number of training faces  (M):");
    scanf("%d", &M);
    printf("\n");

****/
N=atoi(argv[2]);
M=atoi(argv[3]);


/* dynamically allocate memory */

A_trans = matrix(1,M,1,N);
A = matrix(1,N,1,M);
average_face = vector(1,N);
L = matrix(1,M,1,M);
d = vector(1,M);
v = matrix(1,M,1,M);
mag = vector(1, M);

/**** initalize matrix and vectors ****/
for(j=1;j≤M;j++)
    for(i=1;i≤N;i++) {
        A_trans[j][i]=A[i][j]=average_face[i]=0.0;
    }

for(k=1; k≤M; k++){

    fscanf(train, "%s\n", filename);
    /***printf("%s\n", filename); ***/

    facein = fopen(filename, "r");


     for(j=1;j≤N;j++){
        fscanf(facein,"%f\n",&A[j][k]);
            }
     fclose(facein);
    /*** printf("first value of file %d = %f\n", k, A[1][k]); ***/
}
```

```
/*** Normalizing Data by dividing by 255 ***/

/**** for(j=1;j≤M;j++)
   for(i=1;i≤N;i++) {
      A[i][j]=A[i][j]/255;
   }

*******/


/***********Calculate Average Face **********************/

 printf("!!! CALCULATE AVERAGE FACE\n  ");
face_avg=fopen("avg_face.dat","w");
for(i=1;i≤N;i++){
   temp=0.0;
   for(j=1;j≤M;j++)
      temp=temp+A[i][j];
   average_face[i]=temp/M;
   fprintf(face_avg,"%f\n", average_face[i]);
}
fclose(face_avg);



/*** printf("!!! SUBTRACTING OFF AVERAGE FACE \n"); ***/
for(j=1;j≤M;j++)
   for(i=1;i≤N;i++){
      A[i][j]=A[i][j]-average_face[i];
         }


/**** CREATING A TRANSPOSE ***/
 printf("!!!  CREATING TRANSPOSE MATIX  \n");

for(j=1;j≤M;j++)
   for(i=1;i≤N;i++){
      A_trans[j][i]=A[i][j];
   }



/*** fout=fopen("l.dat","w");***/
 printf("!!!  Multiplying A trans and A to get L:\n");
for(i=1;i≤M;i++)
   for(j=1;j≤M;j++) {
      temp=0.0;
      for(k=1;k≤N;k++){
```

```
            temp=temp+A_trans[i][k]*A[k][j];
        }
         /**fprintf(fout,"%f\n", temp);**/
        L[i][j]=temp;
        /** printf("!!!  Writing Output \n"); ***/
    }

/****     printf("!!!  L Matrix WRITTEN TO l2.DAT  \n");
    fclose(fout);****/


/*** ("!!!  FREE MATRIX A_TRANS \n");  ***/
free_matrix(A_trans, 1, M, 1, N);


 printf("!!! doing jacobian of L \n");

jacobi(L,M,d,v, &nrot);
/***printf("%d\n", nrot);***/

/*** printf("!!! doing eigsrt of v \n");***/
eigsrt(d,v,M);


 printf("!!! Writing eigenvvalues \n");
feval=fopen("eigen_val","w");
for(j=1;j≤M;j++) {

    fprintf(feval,"%f\n", d[j]);
    }
fclose(feval);


 printf("!!! Writing eigenvectors \n");

fevex=fopen("eigen_vec","w");
for(i=1;i≤M;i++){
    for(j=1;j≤M;j++) {
        fprintf(fevex,"%f\n", v[j][i]);
                }
    }
    fclose(fevex);



/** printf("!!! Initailizing Eigenface Matrix \n"); **/
u = matrix(1,N,1,M);
for(k=1;k≤M; k++)
    for(j=1;j≤N; j++)
        u[j][k]=0.0;
```

```
 printf("!!! Calcualting eigenface  \n");

for(k=1;k≤M; k++){
   for(i=1;i≤M; i++){
      for(j=1;j≤N; j++)
         u[j][k]=v[i][k]*A[j][i]+u[j][k];
   }
}


/***finding magnitude of eigenface ***/
/** mat_col_mag(u, N, M, mag); ***/


printf("!!! Opening train.out file for Eigenfaces \n");
tempfile = fopen("train.out", "w");

h=48;
l=48;
strcpy(filename, "eigenface");

for(k=1; k≤M; k++){

   if(l ≠ 57) l++;
         else{
            l=48;
            h++;
         }


   fprintf(tempfile,"      ilename);

   fprintf(tempfile,"%c%c",h,l);

   fprintf(tempfile,"%s\n",".dat");

}
fclose(tempfile);


 printf("!!! Writing eigenface \n");

tempfile = fopen("train.out", "r");

for(k=1; k≤M; k++){

   fscanf(tempfile, "%s\n", filename);
```

```
    facein = fopen(filename, "w");
    /*** printf("%s\n", filename);  ***/
    /*** fprintf(facein, "%f\n", mag[k]); ***/

     for(j=1;j≤N;j++){
        fprintf(facein,"%f\n", u[j][k]);

            }
     fclose(facein);
}


fclose(tempfile);

/***printf("!!!  FREEING A MATRIX  \n");****/
free_matrix(A,1,N,1,M);

free_matrix(u,1,N,1,M);

}
void mat_col_mag(u, N, M, mag)
float **u, mag[];
int N,M;
{float   b;
int k, j;
double    sqrt ();

for(k=1; k≤M; k++) {
    b=0;
    for(j=1; j≤N; j++)
        b=u[j][k] * u[j][k] + b;
    mag[k] = sqrt( (double) b);
}
}
```

recon.c

```
/*********************************************************************
    NAME:  recon.c
    INVOKED:  reconstructs faces
    DATE: 24 July 1991
    DESCRIPTION:
   SUBROUTINES CALLED:
    FUTURE MODIFICATIONS/BUGS:
*********************************************************************/
/*** #include <device.h> **/
#include <stdio.h>
#include <math.h>
#include <string.h>
```

```
#define SQ(A)   (A*A)
main(argc,argv)
int argc;
char *argv[];
{
FILE    *face1, *eigenin, *fout, *fweights, *fspectrum, *train;
FILE    *face_avg;
int     i,j, N, k, M, atoi();
float     *vector(), **matrix(), *average_face, max;
float    *d, temp, **u, *pedro, *reconface, *w, *I, temp_mag, mag;
void     free_vector();
void    rescale();
void    vec_mag();
double    sqrt(), fabs();
char    filename[81], *strcpy(), infilename[81], outfilename[81], ext[10], junk[81], ngfile[81];

 if(argc ≠ 5){
 printf("!!! The command line should be !!!:\n\n recon infile ngfile imagesize
numberrecon\n\n");
  exit(0);
  }
  /****************** Set Up Files ************************/
strcpy(ext, ".gra");
strcpy(infilename,argv[1]);
strcat(infilename, ext);

strcpy(ext, ".rec");
strcpy(outfilename, argv[1]);
strcat(outfilename, ext);




if ((face1=fopen(infilename,"r")) == NULL){
  printf("I can't open the input file");
  exit(-1);
  }



if ((fout=fopen(outfilename,"w")) == NULL){
  printf("I can't open the output file");
  exit(-1);
  }

N = atoi(argv[3]);
M = atoi(argv[4]);


/*******
```

```
   printf("!!!  Input total image SIZE(N):");
   scanf("%d", &N);
   printf("\n");
   printf("!!!  Input the number of training faces to reconstruct from  faces (M):");
   scanf("%d", &M);
   printf("\n");
********/


/* dynamically allocate memory  */

u = matrix(1,N, 1,M);
pedro = vector(1, N);
average_face = vector(1, N);
reconface = vector(1, N);
w = vector(1, M);
I =  vector(1, N);

/**** initalize and vectors  ****/
for(j=1; j≤M; j++)
   for(i=1;i≤N;i++)
   w[j]=u[i][j]=I[i]=pedro[i]=reconface[i]=average_face[i]=0.0;



/**** printf("!!!  Opening and Reading Face to be Reconstructed:\n ");  ****/

 for(j=1;j≤N;j++)
   fscanf(face1,"%f\n", &pedro[j]);
fclose(face1);


/* printf("!!!   Opening and Reading Face Average Face (avg_face.dat):\n ");*/

face_avg=fopen("avg_face.dat", "r");
for(j=1;j≤N;j++)
    fscanf(face_avg, "%f\n", &average_face[j]);

fclose(face_avg);

/** printf("!!!  Using %d eigenfaces from file train.out:\n ", M); **/

train = fopen( "train.out", "r");

for(j=1; j≤M; j++){

   fscanf(train, "%s\n", filename);
   /** printf("%s\n", filename); **/
```

```
    eigenin = fopen(filename, "r");


     for(i=1;i≤N;i++){
       fscanf(eigenin,"%f\n",&u[i][j]);
           }
     fclose(eigenin);
    /** printf("first value of file %d = %f\n", j, u[1][j]); **/
}


fclose(train);




printf("!!! SUBTRACTING OFF AVERAGE FACE \n");
    for(i=1;i≤N;i++){
       I[i]= pedro[i] - average_face[i];
           }


/* printf("!!! Calculating Weights based on %d eigenfaces\n", M); **/

for(j=1; j≤M; j++)
    for(i=1; i<N; i++){
       w[j] = u[i][j]* I[i]+ w[j];
}


fspectrum = fopen("spectrum.out", "a");

fweights = fopen(argv[2], "a");
for(i=1; i≤M; i++){
    fprintf(fweights, "%f ", w[i]);
    fprintf(fspectrum, "%d %f\n", i, w[i]);
    }

fprintf(fweights, "%s\n", argv[1]);

fclose(fweights);
fclose(fspectrum);

/*** Normalize weights by dividing by absolute max  *****/

max = 0;

for(i=1; i≤M; i++){
    /*** printf("%f\n",w[i]);***/
    if(fabs( (double) w[i]) >max){
```

```
        max= fabs( (double) w[i]);
    }
}

for(i=1; i≤M; i++)
    w[i]=w[i]/max;



/**** FACE RECONSTRUCTION  ****/


printf("!!! Reconstructing from %M faces\n", M);

for(j=1; j≤M; j++)
    for(i=1; i<N; i++)
        reconface[i] = w[j] * u[i][j]+ reconface[i];



printf("!!! Adding mean back on Reconstructing faces\n");
for(i=1; i<N; i++)
    reconface[i] = reconface[i] + (average_face[i]);

rescale(reconface, N);

 printf("!!! Writing Reconstructed Pedro  \n ");

for(j=1;j≤N;j++){
    fprintf(fout, "%4.0f\n", reconface[j]);
 /***   fwrite(&reconface[j], sizeof(reconface[j]),1,fout); ****/
    }

fclose(fout);


}


void rescale(output, N)
float output[];
int N;
{
int    NEW_MAX, NEW_MIN, i, j, count;
float   min, max;
NEW_MAX = 255;
NEW_MIN =0;
```

```
/*** printf("\n!!! FINDING MAX !!!\n\n"); ***/

/** Check for the max and min value in the data **/
min=output[1];
max=output[1];
count=0;


for(j=1; j≤N; j++){

                if(output[j]>max){
            /** printf("%f\n", max);***/
            max=output[j];
                    }
          if(output[j]<min){
                min=output[j];
            /** printf("%f\n", min); **/
          }
                count++;
      }

/** printf("\n SAMPLES = %d\n", count);
printf(" max = %f min = %f \n", max, min); **/


/*** Now translate data and write to output file ***/
for(i=1; i≤N; i++){
    output[i] = ((output[i]−min)*(NEW_MAX−NEW_MIN)/(max−min) + NEW_MIN);
 }

/** printf("!!! All Done !!!\n\n"); **/


}

void vec_mag(input, N, mag)
float input[], *mag;
int N;
{float    b;
int    i;
double    sqrt();

b = 0;



for(i=1; i<N; i++){
   b = input[i] * input[i] + b;

}
```

```
b = sqrt( (double) b);
/*** printf("mag for u = %f\n", b); ****/
*mag = b;
}
```

find_min.c

```
\* This program implements a First Nearest Neighbor  Network.
Date: Aug 1991
Written By: Pedro F. Suarez
Invoked: find_min trainfile testfile #_train  #_test #_features
where trainfile is a file containing the prototype vectors, testfile is a file containing the test
vectors, #_train number of training vectors in trainfile, #_test number of test vectors in
testfile, #_features in each vector*/

#include <device.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#define   numberblocks   300
#define knumber 4

#define SQ(A)   (A*A)
main(argc,argv)
int argc;
char *argv[];
{
FILE *ftest, *ftrain, *fout;
float distance, *average, *sd, temp, *vector(), **matrix(), number_correct;
int number_train, number_features, junk, temp2, number_test, j, i, min, k, atoi();
double sqrt();

struct    block
   {
   float   **feature_vec1;  /* container for feature vectors */
   float   *mean_vec;
   char     name[20];    /* container for name */
   float   distance;    /* contain distance between test and feature */
      };

struct block atest[numberblocks];   /* reserve test blocks */
struct block btrain[numberblocks];   /* reserve train blocks */

if(argc ≠ 6 ){

 printf("!!! The command line should be !!!:\n");
```

```
 printf("find_min trainfile testfile #_train  #_test #_features\n");
  exit(0);
  }

  printf(" open files \n");


 if ((ftrain=fopen(argv[1],"r")) == NULL){
  printf("I can't open the train file");
  exit(-1);
  }

if ((ftest=fopen(argv[2],"r")) == NULL){
  printf("I can't open the test file");
  exit(-1);
  }

printf("files opened \n");
number_train=atoi(argv[3]);
number_features=atoi(argv[5]);
number_test= atoi(argv[4]);


printf(" initialize b block \n");
temp2 = number_train/knumber; /*
for(i=1; i≤temp2; i++){
    btrain[i].feature_vec = matrix(1,knumber, 1, number_features);
    btrain[i].mean_vec = vector(1, number_features);
    btrain[i].distance = 0.0;
      }
printf(" initialize a block \n");

for(i=1; i≤number_test; i++){
    atest[i].feature_vec=vector(1, number_features);
    atest[i].distance = 0.0;
    }

printf(" create vector \n");

average=vector(1, number_features);
sd=vector(1, number_features);

printf(" init vector \n");

for(i=1;i≤number_features; i++)
        average[i]=sd[i]=0.0;


printf(" read in test vector \n");

for(i=1;i≤number_test;i++){
```

```
   for(j=1; j≤number_features;j++) {
      fscanf(ftest, "%f", &temp);
      atest[i].feature_vec[j]=temp;
      }

fscanf(ftest, "%s\n", atest[i].name);
}
printf(" read in train vector \n");

for(i=1; i≤temp2; i++)
   {
   for(j=1; j≤number_features;j++)
      {
      for(k=1; k≤knumber; k++)
         {
         fscanf(ftrain, "%f", &temp);
         btrain[i].feature_vec[j,k]=temp;
         }
      }

/* Note that the same name is used for all k training vectors per class.  I did this because
we're only interested in making sure the right class wins, and my pathetic C skills ran out
at this point. */

fscanf(ftrain, "%s\n", btrain[i].name);

}

printf(" statistically normalize \n");

for(j=1; j≤number_features; j++)
   for(i=1; i≤number_test; i++)
     average[j]+=atest[i].feature_vec[j]/(number_test + number_train);

for(j=1; j≤temp2; j++)
   for(i=1; i≤number_train; i++)
      for(k=1; k≤knumber; k++)
      {
     average[j]+=btrain[i].feature_vec[j,k]/(number_test + number_train);
      }

for(j=1; j≤number_features;j++)
   for(i=1;i≤number_test;i++){
      sd[j] += (atest[i].feature_vec[j] - average[j])*(atest[i].feature_vec[j] -average[j]);
      }

for(j=1; j≤temp2;j++)
   for(i=1;i≤number_train;i++)
      for(k=1; k≤knumber; k++)
      {
     sd[j] += (btrain[i].feature_vec[j,k] - average[j])*(btrain[i].feature_vec[j,k] - average[j]);
```

```
    }


for(j=1;j≤number_features;j++){
   sd[j] = sqrt( (double) sd[j]);
   sd[j] = ((double) 1/(number_test+number_train−1)) * sd[j];
   }

fout=fopen("data_norm.out","w");
for(j=1; j≤number_features;j++)
   fprintf(fout,"%f ",average[j]);

fprintf(fout,"average\n ");

for(j=1; j≤number_features;j++)
   fprintf(fout,"%f ",sd[j]);

fprintf(fout,"sd\n ");

for(i=1; i≤number_test; i++){
for(j=1; j≤number_features;j++){
          if(sd[j] ≠ 0)
            atest[i].feature_vec[j]= (atest[i].feature_vec[j] − average[j])/sd[j];
      else atest[i].feature_vec[j]= (atest[i].feature_vec[j] − average[j]);

     fprintf(fout,"%f ",atest[i].feature_vec[j]);
    }
   fprintf(fout,"%s\n ",atest[i].name);
}

 for(i=1; i≤number_train; i++){
 for(j=1; j≤number_features;j++){

      if(sd[j]≠ 0)
          btrain[i].feature_vec[j]= (btrain[i].feature_vec[j] − average[j])/sd[j];
      else btrain[i].feature_vec[j]= (btrain[i].feature_vec[j] − average[j]);

     fprintf(fout,"%f ",btrain[i].feature_vec[j]);
     }
   fprintf(fout,"%s\n ",btrain[i].name);
}


fclose(fout);

number_correct=0;

for(k=1;k≤number_test;k++){

for(i=1; i≤number_train; i++){
   temp=0;
```

```
   for(j=1; j≤number_features;j++) {
        temp = (atest[k].feature_vec[j] − btrain[i].feature_vec[j]) * (atest[k].feature_vec[j] −
btrain[i].feature_vec[j]) + temp;
       btrain[i].distance = sqrt ( (double) temp);
       }
   }

temp = btrain[1].distance;
min =1;


for(i=1; i≤number_train; i++){
   if(btrain[i].distance < temp){
        temp = btrain[i].distance;
        min = i;
        }

   }

printf("the matching face is for %s is %s with distance of %f\n",
atest[k].name,btrain[min].name,btrain[min].distance);
}

}
```

gwind.c

```
/*********************************************************************
This routine takes an image by a guassian window.


written by: Pedro F. Suarez
         29 July 91


*********************************************************************/
#include <device.h>
#include <stdio.h>
#include <math.h>
#define pi    3.1416
#define SQ(a)     a*a

main(argc,argv)
int argc;
char *argv[];
{
```

```
FILE *fin, *fout, *fo;
float **p, **w, **matrix(),test, xmean, xvar, ymean, yvar, normal;
int     row, col;
int i, j, k, count;
int outval, temp_i, temp_j;
float inval, max, min;
double      exp();
void    rescale();

if(argc ≠ 3){
   printf("!!! The command line should be !!!:\n\n gauswind infile   outfile\n\n");
    exit(0);
    }


if ((fin=fopen(argv[1],"r")) == NULL){
    printf("I can't open the input file");
    exit(-1);
   }

if ((fout=fopen(argv[2],"w")) == NULL){
    printf("I can't open the output file");
    exit(-1);
   }

 row = col =128;
xmean = 65;
xvar = 25;
ymean = 55;
yvar = 30;

/******

printf("Input the number of rows and cols in Row Col :");
scanf("%d %d", &row, &col);

printf("%d %d\n", row, col);

printf("Input the guassian window xmean and xvariance :");
scanf("%f %f", &xmean, &xvar);
printf("%f %f\n", xmean, xvar);
printf("Input the guassian window ymean and yvariance :");
scanf("%f %f", &ymean, &yvar);
printf("%f %f\n", ymean, yvar);

*****/

/** Allocate Memory for Arrays **/
 printf("Allocating Memory \n");
```

```
p = matrix(1, row, 1, col);
w = matrix(1, row, 1, col);


printf(" reading image of %d row and %d col\n", row, col);


for(j=1; j≤row; j++)
   for(i=1; i≤col; i++)
      fscanf(fin, "%f\n",&p[i][j]);


printf(" Calcualting window \n");

normal =  1/(2 * pi * xvar * yvar);

for(i=1; i≤row; i++)
   for(j=1; j≤col; j++){
      w[i][j] =  exp ((double) −0.5 *( SQ((i−xmean)/xvar)+ SQ((j−ymean)/yvar)));
      p[i][j] = p[i][j] * w[i][j];
   /**printf("%d, %d, %g\n", i, j, w[i][j]);**/
   }
rescale(p, row, col);
rescale(w, row, col);

for(j=1; j≤row; j++)
   for(i=1; i≤col; i++)
            fprintf(fout, "%4.0f\n ", p[i][j]);

fo = fopen("wind.dat", "w");

for(j=1; j≤row; j++)
   for(i=1; i≤col; i++)

            fprintf(fo, "%4.0f\n ", w[i][j]);
fclose(fo);
fclose(fout);
fclose(fin);
}
void rescale(output, row, col)
float **output;
int row, col;
{
int     NEW_MAX, NEW_MIN, i, j, count;
float   min, max;
NEW_MAX = 255;
NEW_MIN =0;


/*** printf("\n!!! FINDING MAX !!!\n\n"); **/
```

```
/** Check for the max and min value in the data **/
min=max=output[1][1];

count=0;


for(j=1; j≤row; j++)
      for(i=1; i≤col; i++){
                        if(output[i][j]>max){
                        /** printf("%f\n", max); **/
                         max=output[i][j];
                      }
               if(output[i][j]<min){
                         min=output[i][j];
                        /** printf("%f\n", min);**/
                   }
                   count++;
      }

/*** printf("\n SAMPLES = %d\n", count);
printf(" max = %f min = %f \n", max, min);  ***/


/*** Now translate data and write to output file ***/
for(j=1; j≤row; j++)
      for(i=1; i≤col; i++){
   output[i][j] = ((output[i][j]-min)*(NEW_MAX-NEW_MIN)/(max-min) + NEW_MIN);
 }

/**** printf("!!! All Done !!!\n\n"); ****/

}
```

C_late5.c

```
/*********************************************************************
   NAME: C_late5.c
   INVOKED: C_late5_n  image1 image2 outfile
   DATE: July 1991
    DESCRIPTION:  This program centers image2 on image1 and places the centered
image2 in outfile.
  SUBROUTINES CALLED:  Correlate(), max_find(), C_late3(), shift(), rescale()
   FUTURE MODIFICATIONS/BUGS:  none
*********************************************************************/
#include <stdio.h>
#include <math.h>
#define skip_line gets(junk)
#define IDX(a,b,c) (a)*(c)+(b)
```

```
#define SQR(a) (a)*(a)
#define loopi(A) for(i=0;i<(A);i++)
#define loopj(A) for(j=0;j<(A);j++)
#define loopij(A,B) for (i=0; i<(A); i++)\
for (j=0; j<(B); j++);

float *vector();
void  fourn();

/*** void        doflip(); ***/
void      Correlate();
void       max_find();
void       C_late3();
void       shift();
void    rescale();

main(argc,argv)
int argc;
char *argv[];
{
int    x,y,i,j, Row, Col,sub_row, sub_col,  downsample;
FILE       *fout, *dat_file1, *dat_file2, *out_file;
char       filename[81], in_string[81], junk[256];
float    *x1, *x2,  *vector(), **matrix(), **image1, **image2;
signed int location[3];


if (argc ≠ 4) {
  printf("!!! The command line should be !!!:\n\n C_late5_n  image1 image2 outfile
\n\n");
   exit(0);
}
/****************** Set Up Files ************************/

if ((dat_file1 = fopen(argv[1], "r")) == NULL) {
printf("I can't open the image1 file");
exit(-1);
}

if ((dat_file2 = fopen(argv[2], "r")) == NULL) {
printf("I can't open the output file");
exit(-1);
}

if ((fout = fopen(argv[3], "w")) == NULL) {
printf("I can't open the image2 file");
exit(-1);
}

/*****printf("Input the number of rows and cols in Row Col :");
scanf("%d %d", &Row, &Col);
```

```
**************************************/
Row = 128;
Col = 128;

/* dynamically allocate memory */
image1 = matrix(0, Row-1, 0, Col-1);
image2 = matrix(0, Row-1, 0, Col-1);

/* Initialize matrix **/
for(y=0; y<Row; y++)
 for(x=0; x<Col; x++){
     image1[x][y]= image2[x][y] = 0.0;
       }
 /****** read the training faces in as columns ****/
 printf("!!!  Opening and Reading Ref Face 1:\n ");

for(y=0; y<Row; y++)
 for(x=0; x<Col; x++){
     fscanf(dat_file1, "%f\n",&image1[x][y]);
   }
fclose(dat_file1);

printf("!!!   Opening and Reading Face to be Centered:\n ");
 for(y=0; y<Row; y++)
 for(x=0; x<Col; x++){
     fscanf(dat_file2, "%f\n",&image2[x][y]);
       }
fclose(dat_file2);



/** ************** START FACE CENTERING ****/
downsample = 1;   /*** downsample allows for faster centering by desampling the image
**/

sub_row = Row/downsample;   /**** down sample image for faster correlation **/
sub_col = Col/downsample;

printf("init vectors\n");

x1 = vector(0,2*sub_row*sub_col-1);
 x2 = vector(0,2*sub_row*sub_col-1);

printf("initialize vectors\n");

loopi(sub_row*sub_col){
 x1[2*i] = x1[2*i+1] = 0.0;
 x2[2*i] = x2[2*i+1] = 0.0;
   }

/********************** PUT MATRIX INTO VECTOR FOR FFT OPS *******/
```

```
loopi(sub_row) {
    loopj(sub_col) (float)  x1[2*(i*sub_col+j)] = image1[(j*downsample)][(i*downsample)];
     loopj(sub_col)  (float) x2[2*(i*sub_col+j)] = image2[j*downsample][i*downsample];
 }

/***** C_late3 return the amount of horizontal(i) or vertical(j) shift needed ****/

C_late3(x1, x2, sub_row, sub_col, location);

 printf(" i location %d    j location %d \n", location[0], location[1]);

/***Some Error Correction ****/
if(location[0]>(sub_col/2)) location[0] = -(sub_col - location[0]);
if(location[1]>(sub_row/2)) location[1] = -(sub_row - location[1]);
if(location[0]<(sub_col/2)) location[0] =  location[0];
if(location[1]<(sub_row/2)) location[1] =  location[1];


location[0] = location[0] * downsample;
location[1] = location[1] * downsample;


/**** ************************** Actually Shift Image with shift  ****/

shift(image2, Row, Col, location);

/***************************** Make 0 to 255 *************/
rescale(image2, Row, Col);

for (y = 0; y < Row; y++)
   for (x = 0; x < Col; x++)
      fprintf(fout, "%4.0f\n ",image2[x][y]);
/**** fclose(fout); ****/

printf(" END OF MAIN\n");

}

/*********************************************************************
  Subroutine C_late3
   INVOKED: C_late3(x1, x2, Row, Col, location)
   DATE: July 1991
  FUTURE MODIFICATIONS/BUGS:  none
*********************************************************************/
void C_late3(x1, x2, Row, Col, location)
int Row, Col;
signed int location[];
float x1[], x2[];
{
 FILE       *out_file;
 int        n[2], i, j;
```

```
 float      *output, temp, **matrix(), **mat_output, *vector();

/** Allocate Memory for Arrays **/
/*** printf("Allocating Memory \n"); ****/

 output = vector(0,2*Row*Col-1);
 mat_output = matrix(0, Row-1, 0, Col-1);

/** Assign Initial Array Values **/

n[0] = Col;
n[1] = Row;


printf("Corellating\n");
Correlate(x1, x2, output, n, Row, Col);


printf("Storing results\n");

/** Store The Magnitude Results **/
 loopi(Row)
   loopj(Col) {
  temp=sqrt((double)SQR(output[2*(i*Col+j)]) +(double)SQR(output[2*(i*Col+j)+1]));
   /**temp=temp/sqrt((double)SQR(output[0]) +(double)SQR(output[1]));**/
   /*** fprintf(out_file,"%4.2f\n", temp);  ****/
   mat_output[j][i] =  (float) temp;
      }

   max_find(mat_output, Row, Col, location);

/*** fclose(out_file);  ****/

}          /** **/


/*********************************************************************
  Subroutine Correlate
   INVOKED: Correlate(input1, input2, output, n, Row, Col)
   DATE: July 1991
  FUTURE MODIFICATIONS/BUGS:  none
*********************************************************************/

void Correlate(input1, input2, output, n, Row, Col)
float input1[],input2[], output[];
int n[], Row, Col;
{
int i;
float *temp1, *temp2;

temp1 = vector(0,2*Row*Col-1);
```

```
temp2 = vector(0,2*Row*Col-1);

loopi(2*Row*Col){
  temp1[i] = input1[i];
  temp2[i] = input2[i];
  }
/** Take Fourier Transform of Input Functions **/

fourn(temp1-1, n-1, 2, 1);
fourn(temp2-1, n-1, 2, 1);

/** Conjugate One of The Fourier Transforms **/

loopi(Row*Col)
   temp2[2*i+1] = -temp2[2*i+1];

/** Multiply Fourier Transforms Together **/

loopi(Row*Col){
  /** Real Component **/
  output[2*i] = temp1[2*i]*temp2[2*i] - temp1[2*i+1]*temp2[2*i+1];

  /** Imaginary Component **/
  output[2*i+1] = temp1[2*i]*temp2[2*i+1] + temp2[2*i]*temp1[2*i+1];
  }

/** Take Inverse Transform to obtain Correlation **/

fourn(output-1, n-1, 2, -1);

/** Rescale to get proper magnitude **/

loopi(2*Row*Col)
  output[i] /= Row*Col;

/*************************************************************************
 The result of the correlation is that first element of the  output
matrix is for zero shift, the next element for shift one to the right and
so on.  This puts results into a format which  humans can understand.
*************************************************************************/


/** Free up the memory when finished **/

free_vector(temp1,0,2*Row*Col-1);
free_vector(temp2,0,2*Row*Col-1);

} /******************** End of Correlate Routine **********************/
```

```
/*****************************************************************
  Subroutine max_find  finds maximum value in a matrix and returns location
   INVOKED:  max_find(mat_output, row, col, location)
   DATE: July 1991
  FUTURE MODIFICATIONS/BUGS:  none
*****************************************************************/


void max_find(mat_output, row, col, location)
float **mat_output;
int row, col;
signed int location[];
{
int i, j, count, temp_i, temp_j;
float   max;

/***printf("\n!!! FINDING MAX !!!\n\n");***/

/** Check for the max and min value in the data **/

max=mat_output[0][0];
count=0;

for(j=0; j<row; j++)
   for(i=0; i<col; i++){

         if(mat_output[i][j]>max) {

            max=mat_output[i][j];
             temp_i=i; temp_j=j;
         }


      count++;
   }
   location[0]=temp_i;   location[1]=temp_j;
   if(location[0]>col) location[0]=0;
   if(location[1]>row) location[1]=0;
/****
printf("\n\n SAMPLES = %d\n", count);
 printf(" max = %f \n", max);

printf("i location = %d   j location = %d \n", location[0], location[1]);

printf("\n!!! All Done !!!\n\n");
****/
}


void shift(image, Row, Col, location)
float   **image;
```

```
int Row, Col;
signed int location[];
{

/*************************************************************************
   NAME: shift
   DESCRIPTION: This routine takes a image shifts
*************************************************************************/
int      **temp_image, i,x,y, xshift, yshift, abs();
float     **matrix(), **shifted_image;
int     n,k, new_row, new_col;

xshift = location[0];
yshift = location[1];
printf(" xshift = %d  yshift = %d \n", xshift, yshift);

printf(" Allocating Memory\n");

new_row= (int) Row+ 2 * abs(yshift);
new_col=(int)  Col+ 2 * abs(xshift);
printf("%d %d %d %d\n", -abs(xshift), new_col, -abs(yshift), new_row);
shifted_image = matrix( -abs(xshift), new_col, -abs(yshift), new_row);

printf(" initialize matrix\n");

/*** initialize matrix ***/

for(y= - abs(yshift); y<new_row; y++)
   for(x= - abs(xshift); x< new_col; x++)
           shifted_image[x][y]=127.0;

printf(" shifting image\n");

for(y=0; y<Row; y++)
     for(x=0; x<Col; x++){
       shifted_image[x+xshift][y+yshift] = image[x][y];
   }


for (y = 0; y < Row; y++)
   for (x = 0; x < Col; x++)
      image[x][y]=shifted_image[x][y];

}


/*************************************************************************
   NAME: rescale
   DESCRIPTION: This routine takes a rescale an range to 0 to 255
*************************************************************************/
```

```
void rescale(output, row, col)
float **output;
int row, col;
{
int     NEW_MAX, NEW_MIN, i, j, count;
float   min, max;
NEW_MAX = 255;
NEW_MIN =0;


printf("\n!!! FINDING MAX !!!\n\n");

/** Check for the max and min value in the data **/
min=max=output[0][0];

count=0;


for(j=0; j<row; j++)
       for(i=0; i<col; i++){
                          if(output[i][j]>max){
                          /* printf("%f\n", max); */
                           max=output[i][j];
                       }
                 if(output[i][j]<min){
                          min=output[i][j];
                          /** printf("%f\n", min);**/
                    count++;
          }
}

/** printf("\n SAMPLES = %d\n", count);
printf(" max = %f min = %f \n", max, min); ****/


/*** Now translate data and write to output file ***/
for(j=0; j<row; j++)
       for(i=0; i<col; i++){
   output[i][j] = ((output[i][j]-min)*(NEW_MAX-NEW_MIN)/(max-min) + NEW_MIN);
 }

}
```

fft_trunc.c

```
/*********************************************************************
NAME:  fft_trunc.c
INVOKED: fft_trunc   infile outfile imsize order
DATE: June 1991
```

```
DESCRIPTION:  This program takes the FFT of an image and if order is greater than
zero it provides the magnitude of the FFT upto order.
SUBROUTINES CALLED:  truncate, fourn, doflip
FUTURE MODIFICATIONS/BUGS:  Ugly code
*****************************************************************/
#include <device.h>
#include <stdio.h>
#include <math.h>

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

#define loopi(A) for(i=0;i<(A);i++)
#define loopj(A) for(j=0;j<(A);j++)
#define loopij(A,B) for (i=0; i<(A); i++)\
for (j=0; j<(B); j++);
#define SQ(A)    (A*A)

char outname[80];

/************************************************************
subroutine: truncate
************************************************************/

void truncate(data,size,order,out_data,name)
float data[],out_data[];
int size,order;
char name[];
{
/** truncate an FFT to order desired  ***/

   FILE *fout;
   int i,j, p, row_offset, col_offset, Row,Col;
   float tempr;
   Row=size;
   Col=size;

   p=0;
   row_offset = (int) Row/2-1;
   col_offset = (int) Col/2;

/*   printf("%s\n", name);    */
   printf("%f\n",data[8128]);
   fout=fopen(outname, "a");

    for(j=(-1*order);j<=0;j++)
    {
     for(i=(-1*order);i<=order;i++)
     {
      out_data[p] = data[(col_offset-i)+((row_offset-j)*Col)];
       fprintf(fout,"%.10f ",out_data[p]);
       p++;
```

```
        }
      }
    fprintf(fout, "%s\n", name);

      fprintf(fout,"\n");

    fclose(fout);
}

/************************************************************
subroutine:doflip   - flips the fourn fft
************************************************************/

void doflip(data,size)
float data[];
int size;
{
/***convert fourn format to format normal humans can use **/
    int i,j, row_offset, col_offset, Row,Col;
    float tempr;

    Row=size;
    Col=size;
    row_offset = (int) Row/2 * Col;
    col_offset = (int) Col/2;

    for(j=0;j<(Row*Col/2);j+=Col)
     for(i=0;i<Row;i++)    /* top half to botttom half swazp */
      {
      tempr = data[i+j];
      data[i+j] = data[i+j+row_offset];
      data[i+j+row_offset] = tempr;
      }

    for(j=0; j<((Row-1)*Col);j+=Col)
     for(i=0; i<Col; i++)   /* left half to right half swap */
      {
      tempr = data[i+j];
      data[i+j] = data[i+j+col_offset];
      data[i+j+col_offset] = tempr;
      }

}

/************************************************************
subroutine:fourn   - Numerical Rec in C  FFT routine
************************************************************/


void fourn(data,nn,ndim,isign)
float data[];
```

```
int nn[],ndim,isign;
{
   int i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
   int ibit,idim,k1,k2,n,nprev,nrem,ntot;
   float tempi,tempr;
   double theta,wi,wpi,wpr,wr,wtemp;

   ntot=1;
   for (idim=1;idim≤ndim;idim++)
      ntot *= nn[idim];
   nprev=1;
   for (idim=ndim;idim≥1;idim--) {
      n=nn[idim];
      nrem=ntot/(n*nprev);
      ip1=nprev ≪ 1;
      ip2=ip1*n;
      ip3=ip2*nrem;
      i2rev=1;
      for (i2=1;i2≤ip2;i2+=ip1) {
         if (i2 < i2rev) {
            for (i1=i2;i1≤i2+ip1-2;i1+=2) {
               for (i3=i1;i3≤ip3;i3+=ip2) {
                  i3rev=i2rev+i3-i2;
                  SWAP(data[i3],data[i3rev]);
                  SWAP(data[i3+1],data[i3rev+1]);
               }
            }
         }
         ibit=ip2 ≫ 1;
         while (ibit ≥ ip1 && i2rev > ibit) {
            i2rev -= ibit;
            ibit ≫= 1;
         }
         i2rev += ibit;
      }
      ifp1=ip1;
      while (ifp1 < ip2) {
         ifp2=ifp1 ≪ 1;
         theta=isign*6.28318530717959/(ifp2/ip1);
         wtemp=sin(0.5*theta);
         wpr = -2.0*wtemp*wtemp;
         wpi=sin(theta);
         wr=1.0;
         wi=0.0;
         for (i3=1;i3≤ifp1;i3+=ip1) {
            for (i1=i3;i1≤i3+ip1-2;i1+=2) {
               for (i2=i1;i2≤ip3;i2+=ifp2) {
                  k1=i2;
                  k2=k1+ifp1;
                  tempr=wr*data[k2]-wi*data[k2+1];
                  tempi=wr*data[k2+1]+wi*data[k2];
```

```
                    data[k2]=data[k1]-tempr;
                    data[k2+1]=data[k1+1]-tempi;
                    data[k1] += tempr;
                    data[k1+1] += tempi;
                }
            }
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
        }
        ifp1=ifp2;
    }
    nprev *= n;
  }
}


/*************************************************************
MAIN

*************************************************************/
main(argc,argv)
int argc;
char *argv[];
{   FILE *fin, *fout;
   float *output,*input,*trunc_out;
   float norm;
   float  *vector();
   void *free_vector();
   char name[30];
   int i,j, nn[1], ndim, isign, new_order, order, image_size;
    if(argc ≠ 5){

printf("!!! The command line should be !!!:\n\n  fft_trunc   infile outfile imsize
order \n\n");
 exit(0);
 }

image_size=atoi(argv[3]);
order=atoi(argv[4]);

sprintf(outname, "%s%d","netfft.ng",order);

/*******************set up dynamic allocation*****************/
input = vector(0,2*image_size*image_size-1);
output = vector(0,image_size*image_size-1);



/****************** Set Up Files ************************/
```

```
if ((fin=fopen(argv[1],"r")) == NULL){
  printf("I can't open the input file");
  exit(-1);
  }


if ((fout=fopen(argv[2],"w")) == NULL){
  printf("I can't open the output file");
  exit(-1);
  }

/**************Read File ********************/

loopi(2*image_size*image_size-1)    /* initialize array to zero */
     input[i] = 0.0;

     loopi(image_size*image_size-1) /*read data in the fourn format */
     {                          /* see numerical recipes in c */
       fscanf(fin, "%f\n", &input[i*2]);
     }


   fclose(fin); /*close input file */


/* Initialization parameters for FFT   */

   nn[0]=image_size;  /* size of mput IAW fourn()   */
   nn[1]=image_size;

   ndim=2;            /* two dim FFT  */
   isign=1;          /* FFT */

   fourn(input-1, nn-1, 2, 1);



/*********** Find Fourier Magnitude  ***********/
   j=0;
   for(i=0;i<(2*image_size*image_size-1); i+=2)
   {

      output[j]=sqrt( (double) SQ(input[i]) +
            (double) SQ(input[i+1]));
            j++;
    }
   norm=output[0];    /* d.c component used for normalization ***/

   printf("%4.0f\n",norm);
```

```
/***** doflip*********************************/

   doflip(output,image_size);   /* converts fourn format to human format */
   printf("%4.4f\n",output[8128]);

   loopi(image_size*image_size){
    output[i]= 1000*output[i]/norm;
   fprintf(fout, "%f\n", output[i]);
   }

/*****   normalize and write output of FFT in argv[2] file   *****/

/******* truncate   ************************************************************/
/* truncate takes fft(output) of size(image_size) and truncates the FFT to  **/
/* order specified plus d.c.  the array is returned in trunc_out, the argv[2]*/
/* is used as a header when truncate writes the output in netfft.dat     */



   if(order ≠ 0){
   new_order = 2*order+1;
   trunc_out = vector(0,image_size*image_size-1);
   truncate(output,image_size,order,trunc_out, argv[2]);


   /*free_vector(trunc_out,0,image_size*image_size-1);*/
}

   free_vector(input,0,2*image_size*image_size-1);
   free_vector(output,0,image_size*image_size-1);
fclose(fout);



}
```

noise.c

```
#include <stdio.h>
#include <math.h>
#define M1 259200
#define IA1 7141
```

```
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349

main(argc,argv)
int argc;
char *argv[];
{
FILE    *dat_file1, *dat_file2;
float gasdev();
float   x,inval, outval;
int    i,var, dum;

if (argc ≠ 5) {
  printf("!!! The command line should be !!!:\n\n noise  inimage1 outimage seed
var\n\n");
   exit(0);
}
printf("Input the reandom seed and var :");
scanf("%d %d", &dum, &var);
dum = atoi(argv[3]);
var = atoi(argv[4]);

if ((dat_file1 = fopen(argv[1], "r")) == NULL)
{
printf("I can't open the image1 file");
exit(-1);
}
if ((dat_file2 = fopen(argv[2], "w")) == NULL)
{
printf("I can't open the output file");
exit(-1);
}


   while(fscanf(dat_file1, "%f\n", &inval) ≠ EOF){


      outval = inval + (var * (float) gasdev( (int)&dum));
      fprintf(dat_file2, "%4.0f\n", outval);

     }

}
```

```
float gasdev(idum)
int *idum;
{
        static int iset=0;
        static float gset;
        float fac,r,v1,v2;
        float ran1();

        if (iset == 0) {
                do {
                        v1=2.0*ran1(idum)-1.0;
                        v2=2.0*ran1(idum)-1.0;
                        r=v1*v1+v2*v2;
                } while (r ≥ 1.0);
                fac=sqrt(-2.0*log(r)/r);
                gset=v1*fac;
                iset=1;
                return v2*fac;
        } else {
                iset=0;
                return gset;
        }
}


float ran1(idum)
int *idum;
{
        static long ix1,ix2,ix3;
        static float r[98];
        float temp;
        static int iff=0;
        int j;
        void nrerror();

        if (*idum < 0 || iff == 0) {
                iff=1;
                ix1=(IC1-(*idum)) % M1;
                ix1=(IA1*ix1+IC1) % M1;
                ix2=ix1 % M2;
                ix1=(IA1*ix1+IC1) % M1;
                ix3=ix1 % M3;
                for (j=1;j≤97;j++) {
                        ix1=(IA1*ix1+IC1) % M1;
                        ix2=(IA2*ix2+IC2) % M2;
                        r[j]=(ix1+ix2*RM2)*RM1;
                }
                *idum=1;
        }
        ix1=(IA1*ix1+IC1) % M1;
        ix2=(IA2*ix2+IC2) % M2;
```

```
        ix3=(IA3*ix3+IC3) % M3;
        j=1 + ((97*ix3)/M3);
        if (j > 97 || j < 1) nrerror("RAN1: This cannot happen.");
        temp=r[j];
        r[j]=(ix1+ix2*RM2)*RM1;
        return temp;
}

#undef M1
#undef IA1
#undef IC1
#undef RM1
#undef M2
#undef IA2
#undef IC2
#undef RM2
#undef M3
#undef IA3
#undef IC3
```

angle.c

```
/*************************************************************************
   NAME: angle.c
   INVOKED: angle weightoutfile totalsize numberofvectors
   DATE: 24 June 1991
   DESCRIPTION: This program calculates the angles between eigenfaces 1 - 8
  SUBROUTINES CALLED: none
   FUTURE MODIFICATIONS/BUGS: change file i/o method to be compatible with
recon.c
*************************************************************************/
#include <stdio.h>
#include <math.h>
#define SQ(A)   (A*A)
main(argc,argv)
int argc;
char *argv[];
{
FILE    *face1, *face2, *face3, *face4, *face5, *face6, *face7, *face8,  *fout;
FILE    *face_avg, *fevex, *feval;
int     i,j, N, k, M, nrot;
float     **matrix(),  *vector(), **A, **w, *mag;
float    *d, temp;
void     free_vector(), free_matrix(), eigsrt(), jacobi();
double    sqrt(), acos();
char     atoi();

 if(argc ≠ 4){
```

```
 printf("!!! The command line should be !!!:\n\n    angle weightoutfile totalsize
numberofvectors\n\n");
  exit(0);
  }

  /****************** Set Up Files ************************/
if ((fout=fopen(argv[1],"w")) == NULL){
  printf("I can't open the angle output file");
  exit(-1);
  }
N = atoi(argv[2]);
M = atoi(argv[3]);


/* dynamically allocate memory */

A = matrix(1,N,1,M);
w = matrix (1,M, 1,M);
mag = vector(1, M);

/**** initalize matrix and vectors ****/
for(j=1;j≤M;j++)
    for(i=1;i≤N;i++)
        A[i][j]=0.0;


for(k=1; k<M; k++)
    for(j=1; j<M; j++)
        w[j][k] = mag[k] = 0.0;


/****** read the eigen faces in as columns  ****/

printf("!!!  Opening and Reading EigenFace 1:\n ");
face1=fopen("eigenface01.dat", "r");
 for(j=1;j≤N;j++){
    fscanf(face1,"%f\n",&A[j][1]);
    /*printf("%4.0f\n", A[j][1]);*/
     }
fclose(face1);


printf("!!!    Opening and Reading Eigenface 2:\n ");
face2=fopen("eigenface02.dat", "r");
for(j=1;j≤N;j++){
     fscanf(face2, "%f\n", &A[j][2]);
    }
fclose(face2);


printf("!!!     Opening and Reading Eigenface 3:\n");
```

```
face3=fopen("eigenface03.dat","r");
for(j=1;j≤N;j++){
    fscanf(face3, "%f\n", &A[j][3]);
   }
fclose(face3);


printf("!!!      Opening and Reading Eigenface 4:\n ");
face4=fopen("eigenface04.dat", "r");
for(j=1;j≤N;j++){
    fscanf(face4, "%f\n", &A[j][4]);

}
fclose(face4);

printf("!!!      Opening and Reading Eigenface 5:\n");
face5=fopen("eigenface05.dat", "r");
for(j=1;j≤N;j++){
    fscanf(face5, "%f\n", &A[j][5]);

}
fclose(face5);


printf("!!!      Opening and Reading Eigenface 6:\n ");
face6=fopen("eigenface06.dat", "r");

for(j=1;j≤N;j++){
    fscanf(face6, "%f\n", &A[j][6]);

}
fclose(face6);


printf("!!!      Opening and Reading Eigenface 7:  \n");
face7=fopen("eigenface07.dat", "r");

for(j=1;j≤N;j++) {
    fscanf(face7, "%f\n", &A[j][7]);
   }
fclose(face7);


printf("!!!      Opening and Reading Eigenface 8:\n ");
face8=fopen("eigenface08.dat", "r");
 for(j=1;j≤N;j++) {
    fscanf(face8, "%f\n", &A[j][8]);

}
fclose(face8);
```

```
printf("!!!      Normalizing    :\n ");

/*** Normalizing Data by dividing by 255 ***/

for(j=1;j≤M;j++)
   for(i=1;i≤N;i++)
      A[i][j]=A[i][j]/255;


printf("!!! Calculating magnitudes\n", M);
for(j=1; j≤M; j++){
   for(i=1; i≤N; i++)
      mag[j] = A[i][j]* A[i][j]+ mag[j];
   mag[j] = sqrt( (double) mag[j] );
/***   printf(" Wait!!\n");   ****/
}

printf("!!! Calculate dot products \n", M);
for(k=1; k≤M; k++){
   for(j=1; j≤M; j++)
      for(i=1; i≤N; i++)
         w[j][k] = A[i][j] * A[i][k] + w[j][k];
   }

printf("!!! Calculate cos alpha \n", M);
for(k=1; k≤M; k++)
   for(j=1; j≤M; j++)
      w[j][k] = w[j][k]/(mag[j]*mag[k]);

printf("!!! Write cos alpha \n", M);

for(k=1; k≤M; k++)
   for(j=1; j≤M; j++)
    fprintf(fout,"Angle in degrees between u%d and u%d is: %3.2f\n",j, k,57.2957*acos(
(double) w[j][k]));

fclose(fout);
printf("!!! ALL DONE \n", M);
}
```

kl_med

```
/*********************************************************************
NAME:  fft_trunc.c
INVOKED: fft_trunc   infile outfile imsize order
DATE: June 1991
DESCRIPTION:  This program takes the FFT of an image and if order is greater than
zero it provides the magnitude of the FFT upto order.
SUBROUTINES CALLED:  truncate, fourn, doflip
FUTURE MODIFICATIONS/BUGS:  Ugly code
```

```
**********************************************************************/
#include <device.h>
#include <stdio.h>
#include <math.h>

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

#define loopi(A) for(i=0;i<(A);i++)
#define loopj(A) for(j=0;j<(A);j++)
#define loopij(A,B) for (i=0; i<(A); i++)\
for (j=0; j<(B); j++);
#define SQ(A)    (A*A)

char outname[80];

/************************************************************
subroutine: truncate
************************************************************/

void truncate(data,size,order,out_data,name)
float data[],out_data[];
int size,order;
char name[];
{
/** truncate an FFT to order desired   ***/

   FILE *fout;
   int i,j, p, row_offset, col_offset, Row,Col;
   float tempr;
   Row=size;
   Col=size;

   p=0;
   row_offset = (int) Row/2-1;
   col_offset = (int) Col/2;

/*   printf("%s\n", name);     */
   printf("%f\n",data[8128]);
   fout=fopen(outname, "a");

    for(j=(-1*order);j<=0;j++)
    {
     for(i=(-1*order);i<=order;i++)
     {
      out_data[p] = data[(col_offset-i)+((row_offset-j)*Col)];
       fprintf(fout,"%.10f ",out_data[p]);
       p++;
           }
     }
   fprintf(fout, "%s\n", name);
```

```
    fprintf(fout,"\n");

   fclose(fout);
}

/************************************************************
subroutine:doflip  - flips the fourn fft
************************************************************/

void doflip(data,size)
float data[];
int size;
{
/***convert fourn format to format normal humans can use **/
   int i,j, row_offset, col_offset, Row,Col;
   float tempr;

   Row=size;
   Col=size;
   row_offset = (int) Row/2 * Col;
   col_offset = (int) Col/2;

   for(j=0;j<(Row*Col/2);j+=Col)
    for(i=0;i<Row;i++)    /* top half to botttom half swazp */
     {
     tempr = data[i+j];
     data[i+j] = data[i+j+row_offset];
     data[i+j+row_offset] = tempr;
     }

   for(j=0; j<((Row-1)*Col);j+=Col)
    for(i=0; i<Col; i++)   /* left half to right half swap */
     {
     tempr = data[i+j];
     data[i+j] = data[i+j+col_offset];
     data[i+j+col_offset] = tempr;
     }

}

/************************************************************
subroutine:fourn   - Numerical Rec in C  FFT routine
************************************************************/


void fourn(data,nn,ndim,isign)
float data[];
int nn[],ndim,isign;
{
   int i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
   int ibit,idim,k1,k2,n,nprev,nrem,ntot;
```

```
float tempi,tempr;
double theta,wi,wpi,wpr,wr,wtemp;

ntot=1;
for (idim=1;idim≤ndim;idim++)
   ntot *= nn[idim];
nprev=1;
for (idim=ndim;idim≥1;idim--) {
   n=nn[idim];
   nrem=ntot/(n*nprev);
   ip1=nprev ≪ 1;
   ip2=ip1*n;
   ip3=ip2*nrem;
   i2rev=1;
   for (i2=1;i2≤ip2;i2+=ip1) {
      if (i2 < i2rev) {
         for (i1=i2;i1≤i2+ip1-2;i1+=2) {
            for (i3=i1;i3≤ip3;i3+=ip2) {
               i3rev=i2rev+i3-i2;
               SWAP(data[i3],data[i3rev]);
               SWAP(data[i3+1],data[i3rev+1]);
            }
         }
      }
      ibit=ip2 ≫ 1;
      while (ibit ≥ ip1 && i2rev > ibit) {
         i2rev -= ibit;
         ibit ≫= 1;
      }
      i2rev += ibit;
   }
   ifp1=ip1;
   while (ifp1 < ip2) {
      ifp2=ifp1 ≪ 1;
      theta=isign*6.28318530717959/(ifp2/ip1);
      wtemp=sin(0.5*theta);
      wpr = -2.0*wtemp*wtemp;
      wpi=sin(theta);
      wr=1.0;
      wi=0.0;
      for (i3=1;i3≤ifp1;i3+=ip1) {
         for (i1=i3;i1≤i3+ip1-2;i1+=2) {
            for (i2=i1;i2≤ip3;i2+=ifp2) {
               k1=i2;
               k2=k1+ifp1;
               tempr=wr*data[k2]-wi*data[k2+1];
               tempi=wr*data[k2+1]+wi*data[k2];
               data[k2]=data[k1]-tempr;
               data[k2+1]=data[k1+1]-tempi;
               data[k1] += tempr;
               data[k1+1] += tempi;
```

```
                }
            }
            wr=(wtemp=wr)*wpr-wi*wpi+wr;
            wi=wi*wpr+wtemp*wpi+wi;
          }
          ifp1=ifp2;
        }
        nprev *= n;
    }
}


/*************************************************************
MAIN

*************************************************************/
main(argc,argv)
int argc;
char *argv[];
{   FILE *fin, *fout;
    float *output,*input,*trunc_out;
    float norm;
    float  *vector();
    void *free_vector();
    char name[30];
    int i,j, nn[1], ndim, isign, new_order, order, image_size;
     if(argc ≠ 5){

printf("!!! The command line should be !!!:\n\n  fft_trunc   infile outfile imsize
order \n\n");
  exit(0);
  }

image_size=atoi(argv[3]);
order=atoi(argv[4]);

sprintf(outname, "%s%d","netfft.ng",order);

/******************set up dynamic allocation*****************/
input = vector(0,2*image_size*image_size-1);
output = vector(0,image_size*image_size-1);



/****************** Set Up Files ************************/

if ((fin=fopen(argv[1],"r")) == NULL){
  printf("I can't open the input file");
  exit(-1);
  }
```

```
if ((fout=fopen(argv[2],"w")) == NULL){
  printf("I can't open the output file");
  exit(-1);
  }

/**************Read File ********************/

loopi(2*image_size*image_size-1)    /* initialize array to zero */
     input[i] = 0.0;

     loopi(image_size*image_size-1) /*read data in the fourn format */
     {                              /* see numerical recipes in c */
       fscanf(fin, "%f\n", &input[i*2]);
     }


   fclose(fin); /*close input file */


/* Initialization parameters for FFT  */

   nn[0]=image_size;  /* size of mput IAW fourn()  */
   nn[1]=image_size;

   ndim=2;            /* two dim FFT  */
   isign=1;          /* FFT */

   fourn(input-1, nn-1, 2, 1);



/*********** Find Fourier Magnitude ***********/
   j=0;
   for(i=0;i<(2*image_size*image_size-1); i+=2)
   {

      output[j]=sqrt( (double) SQ(input[i]) +
            (double) SQ(input[i+1]));
            j++;
    }
   norm=output[0];    /* d.c component used for normalization ***/

   printf("%4.0f\n",norm);

/***** doflip*******************************/

   doflip(output,image_size);  /* converts fourn format to human format */
   printf("%4.4f\n",output[8128]);
```

```
    loopi(image_size*image_size){
     output[i]= 1000*output[i]/norm;
    fprintf(fout, "%f\n", output[i]);
    }

/*****  normalize and write output of FFT in argv[2] file  *****/

/******* truncate  ************************************************************/
/* truncate takes fft(output) of size(image_size) and truncates the FFT to  **/
/* order specified plus d.c.  the array is returned in trunc_out, the argv[2]*/
/* is used as a header when truncate writes the output in netfft.dat     */



    if(order ≠ 0){
    new_order = 2*order+1;
    trunc_out = vector(0,image_size*image_size-1);
    truncate(output,image_size,order,trunc_out, argv[2]);


    /*free_vector(trunc_out,0,image_size*image_size-1);*/
}

    free_vector(input,0,2*image_size*image_size-1);
    free_vector(output,0,image_size*image_size-1);
fclose(fout);



}
```

jacobi.c

```
/*****************************************************************************
NAME:  jacobi.c
DESCRIPTION:  Numerical Recipies routine that finds eigenvectors and eigenvalues of a
matrix
SUBROUTINES CALLED:
WRITTEN BY: Numerical Recipies in C

*****************************************************************************/
```

```
#include <math.h>

#define ROTATE(a,i,j,k,l) g=a[i][j];h=a[k][l];a[i][j]=g-s*(h+g*tau);\
   a[k][l]=h+s*(g-h*tau);

void jacobi(a,n,d,v,nrot)
float **a,d[],**v;
int n,*nrot;
{
   int j,iq,ip,i;
   float tresh,theta,tau,t,sm,s,h,g,c,*b,*z,*vector();
   void nrerror(),free_vector();

   b=vector(1,n);
   z=vector(1,n);
   for (ip=1;ip≤n;ip++) {
      for (iq=1;iq≤n;iq++) v[ip][iq]=0.0;
      v[ip][ip]=1.0;
   }
   for (ip=1;ip≤n;ip++) {
      b[ip]=d[ip]=a[ip][ip];
      z[ip]=0.0;
   }
   *nrot=0;
   for (i=1;i≤50;i++) {
      sm=0.0;
      for (ip=1;ip≤n-1;ip++) {
         for (iq=ip+1;iq≤n;iq++)
            sm += fabs(a[ip][iq]);
      }
      if (sm == 0.0) {
         free_vector(z,1,n);
         free_vector(b,1,n);
         return;
      }
      if (i < 4)
         tresh=0.2*sm/(n*n);
      else
         tresh=0.0;
      for (ip=1;ip≤n-1;ip++) {
         for (iq=ip+1;iq≤n;iq++) {
            g=100.0*fabs(a[ip][iq]);
            if (i > 4 && fabs(d[ip])+g == fabs(d[ip])
               && fabs(d[iq])+g == fabs(d[iq]))
               a[ip][iq]=0.0;
            else if (fabs(a[ip][iq]) > tresh) {
               h=d[iq]-d[ip];
               if (fabs(h)+g == fabs(h))
                  t=(a[ip][iq])/h;
               else {
                  theta=0.5*h/(a[ip][iq]);
```

```
                t=1.0/(fabs(theta)+sqrt(1.0+theta*theta));
                if (theta < 0.0) t = -t;
            }
            c=1.0/sqrt(1+t*t);
            s=t*c;
            tau=s/(1.0+c);
            h=t*a[ip][iq];
            z[ip] -= h;
            z[iq] += h;
            d[ip] -= h;
            d[iq] += h;
            a[ip][iq]=0.0;
            for (j=1;j<=ip-1;j++) {
                ROTATE(a,j,ip,j,iq)
            }
            for (j=ip+1;j<=iq-1;j++) {
                ROTATE(a,ip,j,j,iq)
            }
            for (j=iq+1;j<=n;j++) {
                ROTATE(a,ip,j,iq,j)
            }
            for (j=1;j<=n;j++) {
                ROTATE(v,j,ip,j,iq)
            }
            ++(*nrot);
        }
      }
    }
    for (ip=1;ip<=n;ip++) {
        b[ip] += z[ip];
        d[ip]=b[ip];
        z[ip]=0.0;
    }
  }
  nrerror("Too many iterations in routine JACOBI");
}

#undef ROTATE
```

eigsrt.c

```
/*************************************************************************
NAME: eigsrt.c
DESCRIPTION: Numerical Recipies routine that sorts eigenvalues and vectors into decr-
easing order.
SUBROUTINES CALLED:
WRITTEN BY: Numerical Recipies in C

*************************************************************************/
void eigsrt(d,v,n)
```

```
float d[],**v;
int n;
{
   int k,j,i;
   float p;

   for (i=1;i<n;i++) {
      p=d[k=i];
      for (j=i+1;j≤n;j++)
         if (d[j] ≥ p) p=d[k=j];
      if (k ≠ i) {
         d[k]=d[i];
         d[i]=p;
         for (j=1;j≤n;j++) {
            p=v[j][i];
            v[j][i]=v[j][k];
            v[j][k]=p;
         }
      }
   }
}
```

fourn.c

```
/*************************************************************************
NAME: fourn.c
DESCRIPTION:  Numerical Recipies multi dimensional FFT routine.
Requires a complex column vector  as follows:
/ real a(1)/
/ complex a(1)/
/ real a(2)/
/ complex a(2)/
/ etc/
SUBROUTINES CALLED:
WRITTEN BY: Numerical Recipies in C

*************************************************************************/
#include <math.h>

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void fourn(data,nn,ndim,isign)
float data[];
int nn[],ndim,isign;
{
   int i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
   int ibit,idim,k1,k2,n,nprev,nrem,ntot;
   float tempi,tempr;
   double theta,wi,wpi,wpr,wr,wtemp;
```

```
ntot=1;
for (idim=1;idim≤ndim;idim++)
   ntot *= nn[idim];
nprev=1;
for (idim=ndim;idim≥1;idim--) {
   n=nn[idim];
   nrem=ntot/(n*nprev);
   ip1=nprev ≪ 1;
   ip2=ip1*n;
   ip3=ip2*nrem;
   i2rev=1;
   for (i2=1;i2≤ip2;i2+=ip1) {
      if (i2 < i2rev) {
         for (i1=i2;i1≤i2+ip1-2;i1+=2) {
            for (i3=i1;i3≤ip3;i3+=ip2) {
               i3rev=i2rev+i3-i2;
               SWAP(data[i3],data[i3rev]);
               SWAP(data[i3+1],data[i3rev+1]);
            }
         }
      }
      ibit=ip2 ≫ 1;
      while (ibit ≥ ip1 && i2rev > ibit) {
         i2rev -= ibit;
         ibit ≫= 1;
      }
      i2rev += ibit;
   }
   ifp1=ip1;
   while (ifp1 < ip2) {
      ifp2=ifp1 ≪ 1;
      theta=isign*6.28318530717959/(ifp2/ip1);
      wtemp=sin(0.5*theta);
      wpr = -2.0*wtemp*wtemp;
      wpi=sin(theta);
      wr=1.0;
      wi=0.0;
      for (i3=1;i3≤ifp1;i3+=ip1) {
         for (i1=i3;i1≤i3+ip1-2;i1+=2) {
            for (i2=i1;i2≤ip3;i2+=ifp2) {
               k1=i2;
               k2=k1+ifp1;
               tempr=wr*data[k2]-wi*data[k2+1];
               tempi=wr*data[k2+1]+wi*data[k2];
               data[k2]=data[k1]-tempr;
               data[k2+1]=data[k1+1]-tempi;
               data[k1] += tempr;
               data[k1+1] += tempi;
            }
         }
         wr=(wtemp=wr)*wpr-wi*wpi+wr;
```

```
            wi=wi*wpr+wtemp*wpi+wi;
        }
        ifp1=ifp2;
    }
    nprev *= n;
  }
}

#undef SWAP
```

nrutil.h

```
float *vector();
float **matrix();
float **convert_matrix();
double *dvector();
double **dmatrix();
int *ivector();
int **imatrix();
float **submatrix();
void free_vector();
void free_dvector();
void free_ivector();
void free_matrix();
void free_dmatrix();
void free_imatrix();
void free_submatrix();
void free_convert_matrix();
void nrerror();
```

Bibliography

1. Agawa, Hiroshi and others. "A Stereo-Based Approach to Face Modeling for the ATR Virtual Space Conferencing System." In *SPIE, Visual Communication and Image Processing '90*, Volume 1360, pages 1184–1197, 1990.
2. Fleming, Michael K. and Garrison W. Cottrell. "Categorization of Faces Using Unsupervised Feature Extraction," *IEEE International Joint Conference on Neural Networks*, pages 65–70 (1990).
3. Fu, King Sun. *Sequential Methods in Pattern Recognition.* New York: Academic Press, 1968.
4. Gonzalez, Rafael C. and Paul Wintz. *Digital Image Processing* (Second Edition). Reading MA: Addison-Wesley Publishing Company, Inc., 1987.
5. Harmon, L.D. and others. "Machine Identification of Human Faces," *Pattern Recognition*, *13*:97–110 (1981).
6. Kabrisky, Matthew. *A Proposed Model for Visual Information Processing.* Urbans, IL: University of Illinois Press, 1966.
7. Kabrisky, Matthew, et al. "Interpretation of the Karhunen-Loève Transform." Discussion, July 1991.
8. Kobel, William G. and Timothy Martin. *Distortion-Invariant Pattern Recognition in Non-Random Noise.* MS thesis, AFIT/GE/ENG/86D-20. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1986.
9. Kreysig, Erwin. *Advanced Engineering Mathematics* (Fifth Edition). New York: John Wiley and Sons, 1983.
10. Lambert, Laurence C. *Evaluation and Enhancement of the AFIT Autonomous Face Recognition Machine.* MS thesis, AFIT/GE/ENG/87D-35. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.
11. Luria, Aleksander Romanovich. *Higher Cortical Functions in Man.* New York: Basic Books, 1962.
12. Mahler, Frank A. "A Correlation of Human and Machine Pattern Discriminator." In *NAECON '70*, pages 260–264, 1970.
13. Press, William H. and others. *Numerical Recipes In C.* Cambridge: Cambridge University Press, 1988.
14. Robb, Barbara C. *Autonoumous Face Recognition Machine Using a Fourier Feature Set.* MS thesis, AFIT/GE/ENG/89D-44. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.
15. Rogers, Steven K. *An Introduction to Biological and Artificial Neural Networks.* P.O. Box 10, Bellingham Washington, 98227-0010: SPIE Press, 1991.

16. Routh, Richard L. *Cortical Thought Theory: A Working Model of the Human Gestalt Mechanism*. PhD dissertation, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985.

17. Ruck, Dennis W. and others. "The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function," *IEEE Transactions on Neural Networks*, *1*:296–298 (December 1990).

18. Russel, Robert L. *Performance of a Working Face Recognition Machine Using Cortical Thought Theory*. MS thesis, AFIT/GE/ENG/85D-37. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

19. Sander, David D. *Enhanced Autonomous Face Recognition Machine*. MS thesis, AFIT/GE/ENG/89D-19. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1989.

20. Singstock, Brian D. *Infrared Target Recognition*. MS thesis, Air Force Institute of Technology, Wright–Patterson Air Force Base, December 1991.

21. Smith, Edward J. *Development of an Autonomous Face Recognition Machine*. MS thesis, AFIT/GE/ENG/86D-36. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986.

22. Tarr, Gregory L. *NeuralGraphics User's Guide*. Defense Technical Information Center (DTIC), 1989.

23. Tarr, Gregory L. "LANT Software." Unpublished software developed for Air Force Institute of Technology, Wright-Patterson AFB OH, 1991.

24. Tarr, Gregory L. *Multi-Layered Feedforward Neural Networks for Image Segmentation*. PhD dissertation, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.

25. Tou, J. T. and R.C. Gonzalez. *Pattern Recognition Principles*. Reading, MA: Addison-Wesley Publishing, 1974.

26. Turk, Matthew A. and Alex P. Pentland. "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience*, pages 1 - 28 (September 1990).

27. Turk, Matthew A. and Alex P. Pentland. "Recognition in Face Space," *SPIE Intelligent Robots and Computer Vision IX: Algorithm and Techniques*, pages 43–54 (1990).