(1)

AFIT/GE/ENG/93D-20

IDENTITY VERIFICATION THROUGH THE FUSION OF FACE AND SPEAKER RECOGNITION

THESIS

John Gregory Keller
Captain, USAF

AFIT/GE/ENG/93D-20

AFIT/GE/ENG/93D-20

Identity Verification Through the Fusion of Face and Speaker Recognition

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

John Gregory Keller, B.S.E.E., M.A.

Captain, USAF

December, 1993

Acknowledgements

I would first like to thank my committee members, Doctors Steve Rogers and Mark Oxley, and Captain Dennis Ruck. A special word of gratitude goes to Doctor Rogers for his inspiration and guidance throughout this research. His help made this effort into an incredible learning experience, as well as a pleasurable one.

I also wish to thank "Mother" Dan Zambon for his unceasing efforts in keeping the computer systems in the Signal Processing Laboratory up and running. His skill and patience in keeping up with our endless demands and whining proved invaluable to this entire undertaking.

Lastly, many thanks go to my fellow students. Thanks to the Pattern Recognition crew, Captains Gary Shartle, Bob MacDonald, and Kim McCrae, Flight Lietenant Neale Prescott, and Lieutenant Curtis Martin for keeping me on the right track. And thanks to Captains Jay Cossentine and Warren Gool and their families for proving to be super friends and keeping me sane through some trying times.

John Gregory Keller

AFIT/GE/ENG/93D-20

Abstract

In this research, face recognition and speaker identification systems are each converted into verification systems. The two verification systems are then fused to form a single identity verification system. Finally, the use of the Karhunen-Loéve Transform (KLT) for dimensional reduction is examined for suitability in the verification task.

The base face recognition system used the KLT for feature reduction and a backpropagation neural net for classification. Verification involved training a net for each individual in the database for two classes of outputs, 'Joe' or 'not Joe.' The base speaker identification system used Cepstral analysis for feature extraction and a distortion measure for classification. Verification in this case involved performing the KLT on the Cepstral coefficients and then classifying using a two-class neural net for each individual, similarly to the face verifier implementation.

KLT feature reduction is compared to alternative linear and non-linear methods, and the KLT is found to provide superior performance. The fusion of the two base verification systems is shown to provide superior performance over either system alone.

Table of Contents

Page

Page

List of Figures

List of Tables

Identity Verification Through the Fusion of Face and Speaker Recognition

I. Introduction

The automated recognition of individuals is an area of great interest to both the military and the commercial communities. Instilling such a capability into a machine would benefit many diverse activities, such as validating the identity of an Automatic Teller Machine user or distinguishing a terrorist within a bustling airport crowd. A recognition system could serve as vital a function as protecting our national security by ensuring only authorized personnel are granted access to restricted data in government computer systems. It could also perform as trivial a function as recognizing and greeting the user upon startup of the latest computer game. Table 1.1 provides a listing of possible applications (42).

Though humans are able to perform individual recognition with relative ease, the challenge of automating this function has been daunting researchers for over four decades. This is not to say that all work in the area has been fruitless; in fact, there has been considerable success in finding solutions to certain elements of the problem. Unfortunately, even with those successes, there is currently no automated, autonomous system capable of accurately and consistently identifying individuals in real time. Such a system would find a ready market in today's world.

In the remainder of this chapter, some background on past and ongoing research in the fields of face and speaker recognition will be provided, as well as a statement of the problem to be investigated in this research. Objectives, assumptions, and known limitations will then be outlined, followed finally by a description of the methodology which will be observed in the performance of this research.

1.1 Background

This section contains a brief discussion of the application of pattern recognition techniques to face recognition and speaker identification and outlines some past and present

Table 1.1 Applications for Positive Identification Systems (42)

ACCESS TO RESTRICTED AREAS
Airport Cargo, Ticket and Baggage Areas
Airport Control Towers, Refueling and Maintenance Areas
Embassies and Corporate Offices in Foreign Countries
Nuclear Facilities, Conventional Power Stations and Grid Control Stations
Munitions and Hazardous Materials Storage Areas
Corporate Archives and Computer Centers
Engineering Labs
Blood Banks, Tissue Banks and Forensic Labs
Research and Development Facilities
ACCESS TO DISTRIBUTION OF GOODS AND SERVICES
Automatic Teller Machines - Cash
Point of Sale Terminals - Goods and Credit
Welfare Agencies - Food Stamps and Cash
Drug and Other Clinics - Medication
Computer Networks - Electronic Fund Transfers
ACCESS TO RESTRICTED INFORMATION
Company Proprietary Data, Plans and Forecasts
Government Reports and Regulations in Progress
Classified Government Files
Financial Securities Transactions
R & D Technical and Business Data
Medical and Personnel Records
Patent Applications
Wills and Personal Papers
Competitive Proposals

research in the field. Further details regarding this research will be provided in the review of the literature.

Pattern recognition is an area of research pertaining to the ability of a system (biological or mechanical) to perceive and identify certain characteristics of some target, be the target an image, a sound, or any other piece of data, and determine the classification of the target from those characteristics. The human brain is an excellent example of a biological pattern recognition system. For instance, we find it very easy to differentiate between the letter 'Z' and the number '2,' though both these symbols contain several common characteristics. Our brains provide us the capability of selecting those characteristics that make the symbols different. Such characteristics are classically know as features, and the success of a pattern recognition system depends on its ability to extract appropriate and sufficient features to perform robust classification. As with other recognition systems, both face and voice recognizers must use features that allow discrimination of one target from another, yet will also allow recognition of different occurrences of the same target.

The problem of pattern recognition has historically been broken down into three subproblems: segmentation, feature extraction, and classification (63). Segmentation involves determining the area of interest in a collection of data; that is, determining what region in the data space may contain targets. Feature extraction involves determining which features within (or derived from) the data set will be used to perform the classification. In the classification step the target is identified as belonging to a certain class (or in some recognition systems may be identified as not belonging to a certain, or any, class). Figure 1.1 pictorially illustrates an elementary pattern recognition system.

1.1.1 Face recognition. The segmentation problem in face recognition has traditionally attracted the least attention, and in most research efforts the target faces have been manually segmented; in other words, target faces within images were manually centered before presentation to the feature extraction mechanism. In an Air Force Institute of Technology (AFIT) thesis, Kevin Gay tackled the segmentation problem, and developed a system which used two consecutive frames from a videotape of a subject to determine the location of the face within an image (17). His technique was based on the fact that the

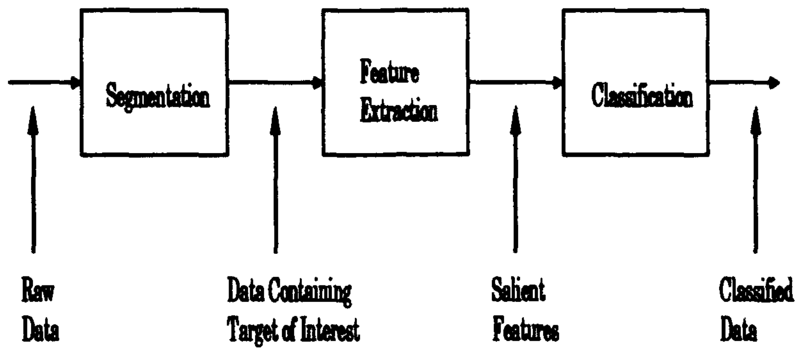

Figure 1.1 A basic pattern recognition system.

head will always be undergoing some relative motion, and subtracting one frame from the other will specify that motion. That information can then be used to determine where the head, and thus the face, can be found within the image.

Figure 1.2 presents an example of the segmentation process. The two photographs are consecutive frames from a videotape sequence, and the "motion" image is the difference of the two frames. The motion image is filtered to remove extraneous noise, and then presented to an algorithm that fills in all pixels below any detected motion. The result is a mapping in space that defines where the face lies. Note that this application depends on a cooperative target with face turned toward the camera, and the images obtained can only contain the single target. These are not limitations if trying to verify an individual's identity in a cooperative situation, but adaptation would be required to use such a system with a non-cooperative target or with a specific target within a crowd.

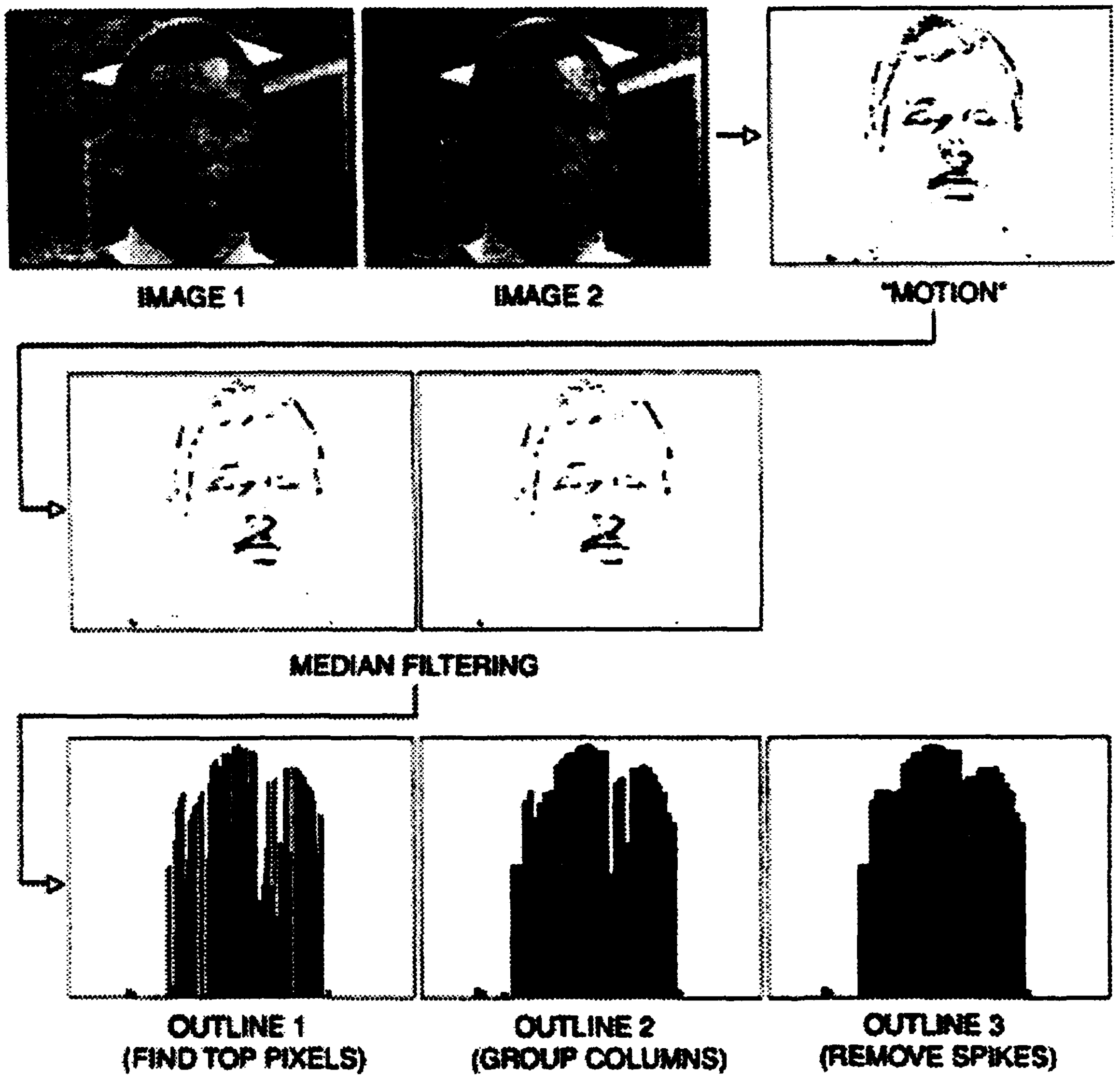

Figure 1.2 Example of segmentation process using Gay's system (17).

The feature extraction problem has been attacked in two general ways. The first methodology relies on extraction and examination of specific facial features to determine the individual to whom the face belongs. Mannaert and Oosterlinck proposed a representative implementation of this method, basing their feature discriminants on geometric proportions, surface properties, and iconic features of the face (34).

The second method uses a holistic approach, in which the faces are examined as a whole. With this technique, it may be appropriate to think of the face as being a feature in itself, as specific features within the face image are not extracted. Turk and Pentland, at the Massachusetts Institute of Technology, have been strong proponents of this methodology, contending that "individual facial features such as the eyes or nose may not be as important to human face recognition as the overall pattern capturing a more holistic encoding of the face" (65). They have developed the concept of "eigenfaces," which are the recognition system's representations of the variations in a set of face images. This information is used to encode and compare the facial features of individuals. Researchers at AFIT have implemented a neural network-based, holistic recognition system relying on the eigenface approach. In a thesis by Ken Runyon, we find that the system performs fairly well for some tests, but does have limitations, such as a marked degradation in performance when presented with a target image that was obtained one or more days after the image on which the system was trained (52).

Gordon recently proposed a relatively unique approach to face recognition: using depth and curvature features of the face to determine identity (20). The strategy is similar to the standard feature extraction of Mannaert and Oosterling, but relies on a completely different feature set that is not restricted to two dimensions in space. Gordon developed and implemented a system at Harvard University based on the use of a rotating laser scanner system to obtain range (depth and curvature) data from subjects. She reports excellent classification capability when the three-dimensional features are presented to the recognizer. Range data has often been used in other pattern recognition problems, but only recently has equipment with the required accuracy become available at acceptable cost.

1.1.2 Speaker Identification. Automated speaker recognition is dependent on features found within acoustic speech signals for classification. Many speaker recognition systems use linear predictive analysis, a method in which the speaker's speech patterns are used to develop a parametric model; new instances of speech can then be compared against this model to determine whether they match the model to some degree of accuracy.

The major deficiency of this approach is the model's poor performance in noise. This occurs because some of the primary assumptions under which the model was developed are violated when the speech signal is corrupted by noise (39:165). Another approach is to use an auditory model, in which speech is converted to a representation of the auditory nerve firing patterns found within the human aural system. In other words, a cochlear model is developed based on the workings of the human cochlea, and acoustic speech presented to this model is recoded to simulate the firing of nerves along the basilar membrane within the auditory system. Because this approach attempts to emulate a system that is known to work well (human hearing), the hope is that more robust recognition can be performed. In an AFIT thesis by John Colombi, such a cochlear model is implemented and compared to the traditional Linear Predictive Coding approach (10).

1.2 Problem Statement

In the course of this research, an implementation will be developed that fuses derivations of the existing AFIT face recognition and speaker identification systems to provide a user verification capability. The bulk of the research will be concentrated on implementing this fusion and determining whether the technique currently being used to extract features from faces is appropriate for the task. Alternate methods for feature extraction will be explored. The system performance will be measured using two metrics: classification accuracy, and identification speed. The classification accuracy will be a function of the robustness of the system, and will be based not only on identification accuracy, but also on rejection inaccuracy; that is, both mistaken recognition and mistaken rejection will be considered. Recognition speed will be based on the perception of "acceptable" time for recognition. This is a somewhat subjective measure, imposed by the probability that a system user will only accept a recognition system if it performs with a certain relative speed. Of these two measurement metrics, classification accuracy will be considered the more important.

1.3 Research Objectives

The primary objectives of this research are to fuse two single-sensor recognition systems (face and voice) into a single identity verification system and to attempt to improve the classification performance of the face identification portion of the fused system. The former will require that methods be developed to convert the individual recognition systems into verification systems, and then that the outputs of these verification systems be fused in some probabilistic space. Improvements to the system will be attempted by determining the appropriateness of the current feature extraction method and developing new ones for testing.

1.4 Assumptions

- Methods can be developed to convert the existing recognition systems (identifying an individual as a member of a data base) into verification systems (verifying that the individual is who he/she claims to be).
- The outputs of the individual verification systems will be, or will be directly related to, post (posteriori) probabilities. This will allow simple probabilistic fusing.
- This research will depend on a cooperative subject. The orientation of the head in the image will be face forward, and the subject will be alone and close enough to the camera to allow either manual or automatic segmentation. The subject will speak into a microphone when prompted, and the acoustic environment will not be excessively noisy.

1.5 Scope and Limitations

The scope of this research will be limited to exploring methods of converting and fusing the two recognition systems and analyzing the use of an alternate feature set for presentation to the face verifier. The performance may be limited by the accuracy of the computational and image-capturing hardware provided.

1.6 Approach/ Methodology

Existing software will be modified or new software will be developed and implemented on a Sun SPARCstation2 . The existing face segmentation software will be retained in whole or in part, as will classification algorithms developed during previous AFIT face recognition research efforts. The face recognition system used will extract features using the Karhunen-Loeve Transform (KLT) technique as outlined in Chapter 2 of this thesis, and will train a back-propagation, multi-layer perceptron to perform the classification task. Speaker recognition algorithms developed at AFIT will also be retained in whole or in part; Cepstral analysis will be used for for feature extraction and a distortion metric for classification. The development software used will be a UNIX implementation of ANSI C. The following tasks will be accomplished during the course of the research effort:

1. Convert the existing AFIT holistic face recognition system into a holistically-based face verification system. Perform tests using this system to develop a baseline against which to measure modified or newly developed systems and capabilities.
2. Convert the existing AFIT auditory model-based speaker identification system into a speaker verification system. As with the face verifier, perform tests using this system to develop a baseline against which to measure modified or newly developed systems and capabilities.
3. Develop the algorithms necessary to fuse the two verification systems into a single user verification system.
4. Develop the algorithms for testing and analyzing the use of an alternate feature set for face verification based on a calculated Figure of Merit (FoM).
5. Develop the algorithms for testing and analyzing the use of an alternate feature set for face verification based on nonlinear dimensional transformation.
6. Measure the performance of each of the verification systems alone and the fused verification system KLT/FoM- based face feature sets.

1.7 Conclusion

The systems developed during this research effort will help further understanding of the face recognition and speaker identification processes, bringing us one step closer to our goal of constructing a reliable and accurate mechanism for autonomous identity verification. Some uses for such a mechanism were briefly mentioned in the introduction to this chapter, but there exist many other activities which will also benefit; indeed, it should be expected that an identity verification system will serve purposes of which we have not yet conceived. But until the day comes that such a capability exists, researchers will continue the quest for increased knowledge about the mechanics of recognition through efforts similar to the one to be undertaken here.

In the next chapter, we shall review recent and current research into the areas of face recognition, speaker identification, and multi-sensor fusion. Methodologies and motivations for user verification will be presented, as well as a brief survey of current implementations.

II. Literature Review

2.1 Introduction

Security issues play an increasingly important role in both the long- and short-term operation of the government. The ability to protect data within the electronic confines of a computer system becomes vital as one considers the threat to national security that could ensue if unauthorized agents were given improper access. The main thrust in the area of computer security has thus been to develop procedures to ensure only authorized users are permitted access to important data. These procedures have traditionally ranged from implementing simple password protection schemes to verifying some physical device, such as an access card, and even to requiring manual identification and verification of users by security guards. The problem with the first method is that passwords can be "broken," and the second method carries with it the assumption that the access device will always be with the authorized user. The third method requires human security guards, who by nature are extremely adept at providing accurate individual authentication, but will tend to perform increasingly poorly as more persons are added to the database of authorized users (9). Therefore, it is natural that we look for some method of automating the user verification process, thereby providing increased security.

In this review, current research into automated identification and verification of individuals will be examined. Research being performed in the areas of recognition and identification of faces will first be surveyed, followed by a review of efforts in the speaker identification arena. Finally, methods used to fuse the outputs of multiple sensor systems will be presented.

2.2 Face Recognition

The recognition of a familiar face is something we humans take for granted. We perform this task with admirable precision, and only seldom is any effort involved. But for all that natural skill, no one really understands precisely how we perform recognition. Recent research indicates the process of face recognition involves at least two major steps (21). The first is known as segmentation, in which we are alerted to the fact that there is a face within our visual field. At this stage, recognition of the person as an individual has not occurred; we simply notice the face. In the next step we actually recognize the face as being familiar to us. Efforts to mimic these biological processes with machines have met with varying degrees of success, and in the following sections some of these efforts will be outlined.

2.2.1 Segmentation of Faces. The problem of determining whether or not a face is present in the visual field has not been extensively addressed in the literature. Most face recognition research has assumed the face is known to be there, and is already pre-processed (scaled, rotated, and positioned) for introduction to the recognition mechanism itself. Govindaraju, Sher, et al, studied the problem of locating faces in newspaper photographs, developing a geometric model of the prototypical human face and scanning photographs for images approximately representative of that model (21:p551). The technique was quite successful, but constraints placed on the problem included:

1. Frontal face view required in photograph.
2. Face must be upright with negligible tilt.
3. Faces must not be occluded by other objects.
4. Face must be at least some minimum size.
5. Image must have some minimum resolution.
6. Number of faces to be found must be known.

Though these restrictions may be well adapted to finding faces in newspaper photographs, they are not likely to be adequate for real-world "face in a crowd" capturing.

A more complex pattern-matching scheme has been proposed by Seitz and Bichsel (57). Their approach involves performing a hierarchical search for features in progressively finer resolution images (images containing progressively higher spatial frequency content), with the assumption that at different levels of resolution, different features will be more important. For instance, at low resolutions (about two Hz), only the broad outline of the head is searched for, a horizontal oriented line in the upper part of the image and two vertical lines at the left and right sides of the image. At finer resolutions, the nose is localized, and then the eyes and pupils. This proved to be a relatively robust method, and was somewhat invariant to rotation; once the pupils were found, the planar rotation of the head could be calculated, and the face could be rotated into a standard position for further processing. The particular application presented did not account for scaling differences, but the authors state that in principle, the information was available to scale the face for an unknown size.

In an Air Force Institute of Technology thesis by Kevin Gay, we are shown that the use of motion analysis may be a suitable technique for face segmentation (17). His approach was to capture two images of a subject from a fixed camera in rapid succession, then perform a frame-to-frame subtraction to determine any motion. Based on the fact that humans cannot keep their heads perfectly still, he hypothesized that movement in the image could correspond to the presence of a face. After enhancing the motion image (the difference between the two frames), it was analyzed for detection of a face, and if one was found it was cut from the image and resized to create a standard size vector for input to the face recognition system. This approach proved to be fairly successful, but not flawless. Gay found that the outlines of the motion images were not consistent, but could not determine the cause of the inconsistency. He felt that given a better method of finding the motion image, standardization and face discrimination capability could likely be improved.

2.2.2 Recognition of Faces. There are currently two major approaches being examined by researchers into face recognition techniques. The first relies on extraction and examination of specific facial features to determine the individual to whom the face belongs, while the second extols a holistic approach, examining the face as a whole.

2.2.2.1 Feature Extraction. **Facial Geometric Proportion** Mannaert and Oosterlinck proposed a representative implementation of the first method, basing their feature discriminants on geometric proportions, surface properties and iconic features (34). The use of geometric proportions is based on the idea that certain distances within the human face may vary between people, but are quite invariant for the same person.

Examples include the vertical distance from the eyes to the upper side of the mouth and the horizontal width of the face at the nose.

Surface, or texture, characteristics depend on extraction of "smoothness" information about certain areas of the face, such as the forehead or the cheek, and are calculated by relating the mean intensity gradient at a particular region to the entropy of the histogram in that region. Iconic features depend on characteristics of certain subimages of the original image. The shape of the chin, for example, could be a valid, discriminable feature. Another textural proposed by the authors, but not applied during the study described, is a measure of the standard deviation of the fractal dimension, to determine the presence of hair.

Once the features were extracted from the image, they were correlated with the features of all faces present in the database. Measures of similarity and Euclidean distance were accomplished, and the system identified the face in the image by selecting the best match. Preliminary results from the use of this system have been promising, and the authors intend to continue development with a larger database of faces, more different sensors for feature detection, and various other enhancements.

Biologically Motivated Feature Extraction A somewhat different method of feature extraction is presented by Manjunath, whereby features are extracted without any assumptions concerning face structure (33). His work is biologically motivated, in the sense that it attempts to emulate the human visual system's ability to recognize images that don't necessarily lend themselves to simple geometrical representations.

> The development of the feature detection model is motivated by the early processing stages in the visual cortex of mammals. The cells in the visual cortex can be classified into three broad functional categories: simple, complex, and hypercomplex. Of particular interest here is the end-inhibition property exhibited by the hypercomplex cells. This property refers to the response of these cells to short lines and edges, line endings, and sharp changes in curvature (e.g., corners). Since these correspond to some of the low level salient features in an image, these cells can be said to form in some sense a low level feature map of the intensity image (33:p374).

Similarly to Seitz and Bichsel's face segmentation work, Manjunath proposes the extraction of oriented feature information at different scales. He obtains the information by using Gabor wavelet transformations on the original intensity image, where Gabor functions are simply Gaussians modulated by complex sinusoids. The wavelet transformation decomposes the original signal into a linear combination of basis functions, which are obtained from simple dilations and translations of a "mother" wavelet (for an in-depth treatment of wavelet transform theory, see (8)). The decomposed signals represent different spatial resolutions, and the information contained within the different levels can be used to localize curvature changes. An example of such features found by this method is seen in Figure 2.1, where the input image is a hand-drawn hammer, and the processed image shows a star at each location of changing curvature. Figure 2.2 shows the same technique applied to two face images. The curvature information at those points represents the features within the image, and an appropriate cost function is used to determine whether two different feature maps represent the same face.

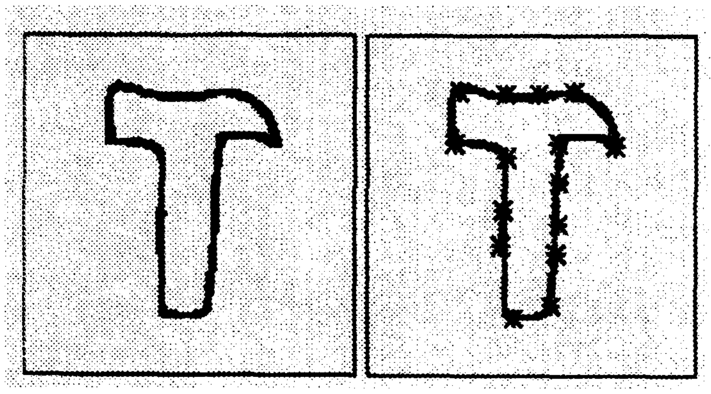

Figure 2.1 Curvature changes found via the wavelet transformation of a hand-drawn image (33).

Facial Thermographic Feature Extraction Prokoski, et al, have presented an identification system based on the extraction of facial thermographic features (42). They claim that

> . . . the thermal measurements of individuals under repeated conditions are highly repeatable. The mean and standard deviation temperature of a group of individuals over a period of several months were 30.8°, +/- 0.032° C with a coefficient of variation of 0.1

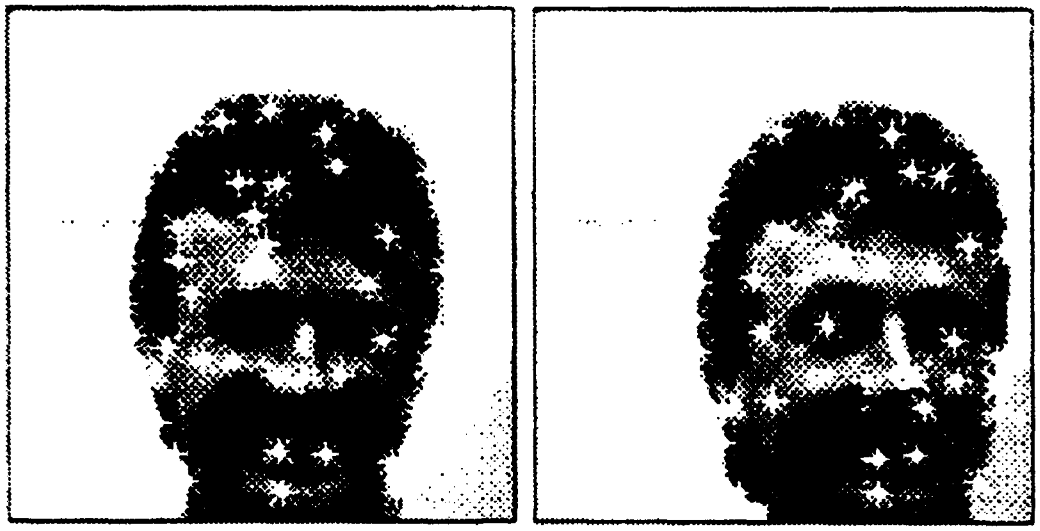

Figure 2.2 Manjunath's technique applied to faces (33).

The amount of information contained within a thermographic image of a face is quite large, and though the authors have not performed extensive testing, they feel the information is sufficient to discriminate between and identify individuals.

Three-dimensional Feature Extraction Jia and Nixon propose a method of extracting profile information from a two-dimensional, front-view of a face, and using that information as additions to a standard geometrically based feature set (25). The authors assume the position of the eyes within the image have been located with a high degree of accuracy, and thus the center, vertical line of the face can be found. They then calculate the intensity projection along the direction of that line, where the intensity projection $p_w(z)$ of image $f(x,z)$ along the line z in direction w is simply

$$p_w(z) = \int_z f(x,y)dw$$

Though not precisely the profile of the face, this projection represents the relation between the intensity peaks and valleys along the center line of the face. Figure 2.3 shows an example of this intensity extraction applied to a face. Seven series of feature data were derived from this projection:

1. the resampled projection. $p(i)$.
2. the autocorrelation of $p(i)$.
3. the Dyadic autocorrelation function of $p(i)$.
4. the Fourier transform of $p(i)$.
5. the Walsh transform of $p(i)$.
6. the Fourier power spectrum of $p(i)$.
7. the Walsh power spectrum of $p(i)$.

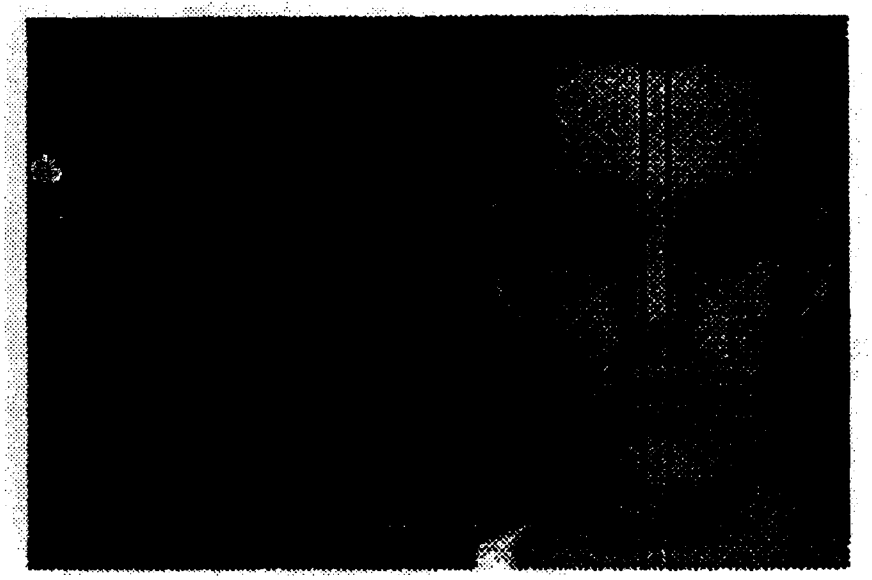
Figure 2.3 Intensity projection of a face (25).

The authors feel that these features, combined with traditional geometrical measurements, provide superior discrimination capability to feature sets consisting of geometrical measurements alone. Their results support this conclusion, though the effect of hair falling on the forehead, beard and moustache growth, and other physical changes to the pseudo-profile were not addressed.

Gordon has proposed a face recognition methodology based on the extraction of depth and curvature features from a face. The major difference between this method and most other feature extraction methods is that features here are actually based on depth measurements, and not on intensity values. Because intensity based image descriptions depend on intensity variations, low contrast features such as cheeks and foreheads are very difficult, if not impossible, to describe. This approach is also different from biologically motivated ones, because "although it is unlikely that humans base their representation or comparison of shape on the accurate perception of depth, we propose the use of depth data because at our current state of technology it is the most straight forward way to input or record complex shape information for machine analysis."(20:p235) A rotating laser scanner was used to extract the depth information, which in turn generated a surface embedded in a three-dimensional space. Curvature measurements across the surface were then obtained and templates were produced corresponding to specific physical facial features. See Figure 2.4 for an example of the curvature maps obtained, and Figure 2.5 for local maxima/minima plots derived from the maps. Face identification could then be accomplished by simple template matching or by measuring the volumetric difference between a test surface and a known surface when both were normalized with respect to a small set of common feature points.

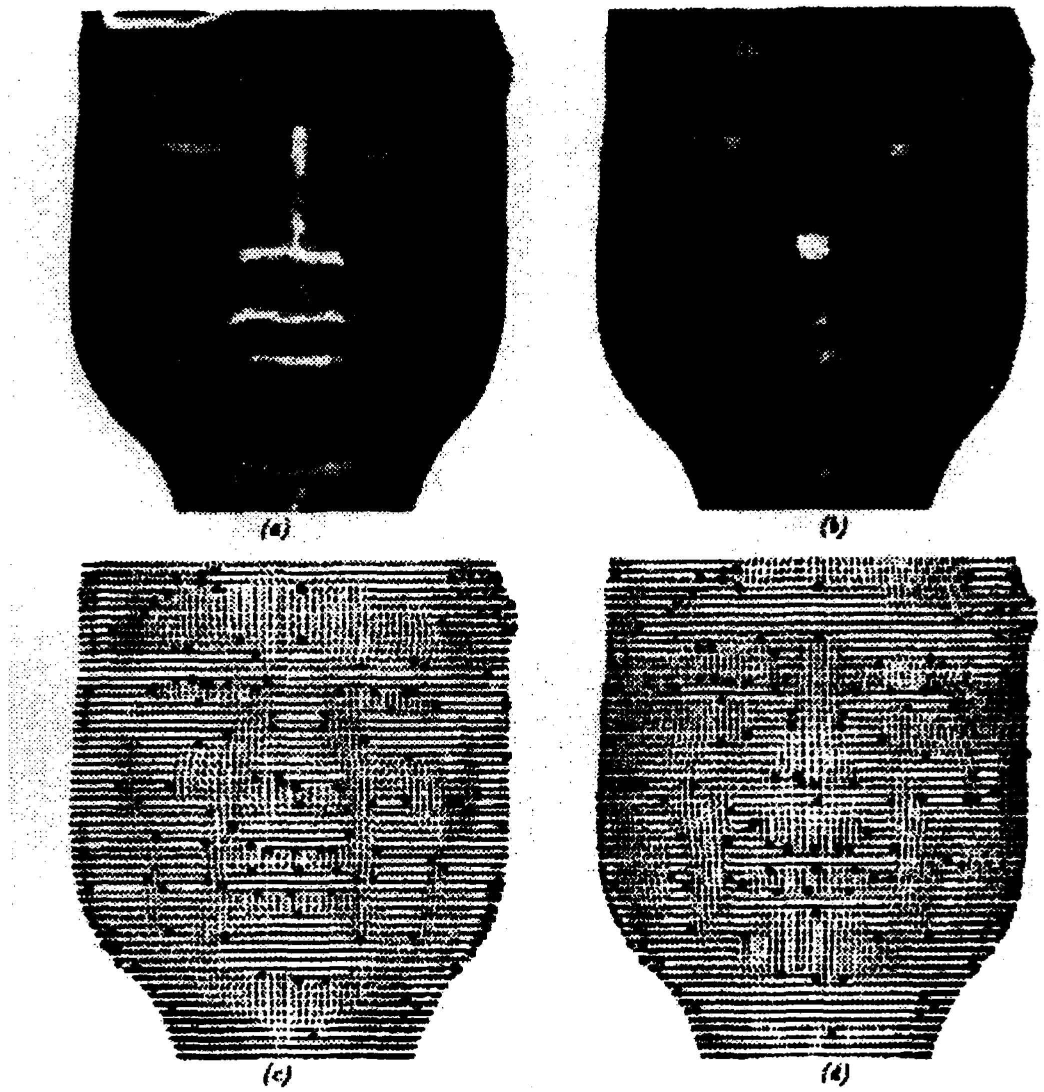

Figure 2.4 Principle curvatures for a single face: magnitude (a) and direction (c) of maximum curvature, magnitude (b) and direction (d) of minimum curvature. Umbilic points are marked in (c) & (d); filled circles are points with positive index and open circles are points with negative index (20).

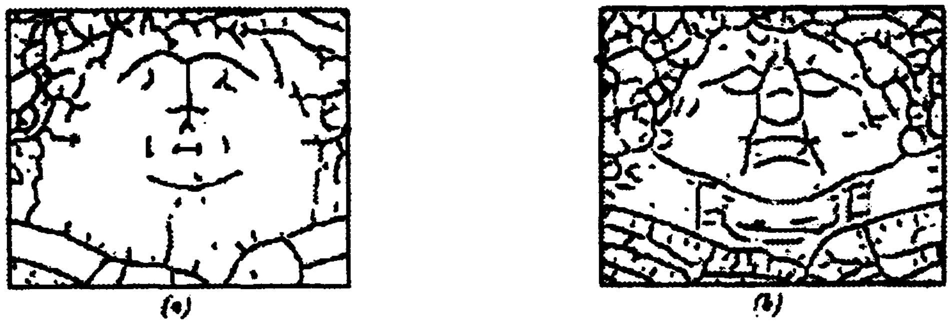

Figure 2.5 (a) Ridge lines: local maxima of ($k_{max} > thresh_r$), and (b) valley lines: local minima of ($k_{min} < thresh_r$). (20).

2.2.2.2 Holistic Recognition. Turk and Pentland, at the Massachusetts Institute of Technology, have been quite active in pursuit of the application of holistic recognition techniques to the face recognition problem. They contend that "individual facial features such as the eyes or nose may not be as important to human face recognition as the overall pattern capturing a more holistic encoding of the face" (65). This approach leads to less dependency on detailed geometries, and may lead to simpler computational models well suited to use in certain constrained environments such as offices. In their scheme, face images were decomposed into characteristic feature images, called eigenfaces. These eigenfaces occupy unique positions in what is known as face-space, and identification is made by projecting the image to be identified into the face-space, then determining the eigenface that is closest to the test face. Turk and Pentland claim the approach has advantages over other face recognition methodologies in speed and simplicity, learning capacity, and insensitivity to small or gradual changes in the face image.

Fleming and Cottrell developed a similar system, training a back-propagation neural network to automatically extract holistic features from face images and save them as "holons," similar to the eigenfaces of Turk and Pentland (14). The resultant network could recognize new images of familiar faces, categorize unknown images as to their "faceness," and to a degree categorize faces as to their gender.

Researchers at AFIT have also implemented a neural network based, holistic recognition system (52, 27, 17). Because the face verification portion of the system to be developed for this thesis depends on the AFIT implementation, details of its operation will be provided in Chapter 3. In a thesis by Ken Runyon, we find that the system performs fairly well for some tests, but does have limitations (52). The major one was in recognition accuracy over multiple days of testing. That is, recognizing an image of a face taken at a different time than the image the system was trained on. This was overcome to a large extent by training the system with images taken over multiple days, which allowed the net to capture a more general "view" of what the face looked like.

2.3 Speaker Identification

As with the recognition of human faces, we tend to take for granted our ability to recognize the voice of a familiar person, even if the voice has been altered in some way (changes in pitch or changes due to illness, for example). Indeed, according to Levinson and Roe, humans are unable to appreciate the difficulties that speaker recognition poses for a computer, since humans comprehend speech so easily (29). Two general schemes are generally used for speaker identification, one based on extraction of features within an utterance, and the other on the use of a model of the mammalian auditory system.

2.3.1 Feature Extraction. Most feature-based speaker identification systems rely on some preprocessing of the acoustic speech signal, selecting features which attempt to model the physical makeup of an individual's vocal tract and using those features to build a database of known individuals (39). Many preprocessing methodologies have been used in the past, and Colombi provides an excellent synopsis of those techniques in his 1992 thesis (9), reproduced here in Table 2.1.

Once the features have been extracted from the acoustic signal, it remains to classify them, thereby determining with what probability a test subject belongs to some class. Again, Colombi has provided a summary of different classification techniques used over the last several years (Table 2.2).

The speaker verification system to be implemented as part of this thesis effort will be based on the recognition system developed at AFIT by Colombi. Details of that system,

Table 2.1 Preprocessing Techniques for Speaker Recognition Feature Extraction (9)

Feature	Author (Date)	Comments
Filterbanks	Pruzansky (1963, 1964)	100Hz - 10KHz, various averages of (and between several) filterbank outputs over time were examined (39).
Spectral Characteristics	Wolf(1972)	Nasal consonants, fricatives, v owels, pitch and vowel duration (39).
Pitch Contours	Atal(1972)	Karhunen-Loève transform on pitch con tours (39).
Filterbank Correlation	Li and Hughes(1974)	Correlations among filterbank en ergies (39).
LPC Cepstral	Atal(1974, 1976)	Comparison to log-area ratios, correlation coefficients, LPC coefficients (2, 1).
Spectral Characteristics	Sambur(1975)	Formant frequencies, LPC Poles, pitch, some temporal patterns (39).
Formants	Goldstein (1976)	Vowels, 199 ranked features (39).
Linear Prediction	Sambur (1976)	LPC, reflection, log-area ratios, found orthogonal reflection coefficients best (least significant projections) (39).
Long-Term Statistics	Markel (1977, 1979)	Mean and standard deviation of pi tch, reflection coefficients (39).
Mel Cepstral	Davis and Mermelstein (1980)	Cosine expansion of the spectrum, comparison to linear and LPC cepstral (13).
Delta Cepstral	Furui(1981)	Polynomial expansion over time (15).
Log Area Ratios	Schwartz(1982)	Examined different classifiers using spectral log area ratios (56).

Table 2.1 (cont'd) Preprocessing Techniques for Speaker Identification Feature Extraction

Feature	Author (Date)	Comments
LPC Cepstral	Oglesby and Mason (1990)	10th order LPC derived cepstral (37).
Line Spectral Pair	Liu(1990)	Several variants of LSP - Even, Odd, Mean and Difference of LSPs (31).
Mel Cepstral and LPC	Bennani (1990)	12th order LPC and Mel Frequency Cepstral, based on 24 triangular filters (4).
LPC Cepstral	Gaganelis and Frangoulis (1990)	10th order LPC (16).
Delta LPC Cepstral	Furui (1991)	LPC cepstral, first order regression every 88 msec period (35).
Delta Cepstral /Cepstral	Rosenburg (1990, 1991)	12th order cepstral and delta- cepstral coefficients, weighted using a sinusoidal "lifter" (48, 49).
Mel Cepstral	Oglesby and Mason (1991)	12 filterbanks, Mel frequency spaced (38).
Eigenvector Analysis	Bennani (1991)	LPC and Mel cepstrum covariance, mean and two eigenvectors (3).
Filterbanks	Higgins (1991)	Power output of 14 uniformly spaced frequency banks (23).
Auditory Model	Hattori (1992)	Seneff auditory model mean rate response, 40 channels (22).
Delta Cepstral /Cepstral	Tseng et al (1992)	Linear combination of cepstral and delta cepstral. Found cepstral alone performed better recognition (64).
LPC Cepstral	Savic and Sorensen (1992)	20th order cepstral derived from only 12th order LPC (54).

Table 2.2 Classification Techniques for Speaker Recognition (9)

Classifiers	Author (Date)	Speakers, ID %, Comments
Distortion	Atal (1974)	10 speakers, 98% identification, Mahalonobis Distance using pooled intra speaker covariance (2).
DTW	Furui (1981)	20, Dynamic Time Warp distortion measurement on fixed sentences (15).
K-means, Gaussian Estimation	Schwartz (1982)	Compared Gaussian classifiers to K-means and Mahalonobis Distance, non-parametric outperformed (56).
HMM	Poritz (1982)	Application of 5 state ergodic HMM to speaker verification (40).
VQ	Soong (1985)	First Speaker dependent codebooks, voiced and unvoiced speech (59).
VQ	Soong (1988)	2 Codebooks, 1 instantaneous and 1 temporal (60).
MLP	Oglesby and Mason (1990)	10, 92%, Backprop learning, single layer with 16 - 128 hidden nodes, Equal recognition to VQ s5.1.10.
K-means/ LVQ	Bennani et al (1990)	10, 95 - 97% (4).
HMM	Rosenburg et al (1990)	20, 98.8 - 99.1%, Used k-means to segment the utterance into acoustic segment units, also examined phonetically labeled speech (48).
HMM	Savic and Gupta (1990)	43, 97.8%, 5 HMM models representing broad classes (55).
GMM	Rose and Reynolds (1990)	12, 89%, Only 1 sec of test speech (46).

Table 2.2 (cont'd) Classification Techniques for Speaker Recognition

Classifiers	Author (Date)	Speakers, ID %, Comments
Binary Partition	Rudasi and Zahorian (1991)	47, 100%, TIMIT corpus, need N(N-1)/2 binary MLP classifiers. (51)
RBF NN	Oglesby and Mason (1991)	40 , 89% true talker, different mannerisms of speech. (38)
GMM	Rose et al (1991,1992)	10, 77.8%, Integrated noise model into GMM, GMM on Original clean speech - 99.5 (46, 47) %.
Discriminator Counting	Higgins and Bahler (1991)	24, 80% true talker, KING corpus, multivariate gaussian, count wins/speaker summed over frames.
VQ	Matsui and Furui (1991)	9, 98.5 - 99.0 %, Voice/Unvoiced or 2-state HMM, New Distortion measure (DIM), Talker variability normalization (TVN) individually weights features. (35)
HMM	Rosenburg (1991)	20, 96.5 - 99.7%, Whole word L-to-R *HMM, text* dependent (digits), compared to VQ.(49)
Time Delay NN	Bennani and Gallinari (1991)	20, 98%, First a Male / Female TDNN, then a 10 output (speakers) TDNN using 2 hidden layers (hierarchical). (3)
HMM, VQ, ANN	Hattori (1992)	24, 100 %, TIMIT corpus (females), Predictive NN (recurrent) within HMM, compared to VQ and MLP classifiers. (22)
CPAM (GMM)	Tseng et al (1992)	20, 98.3% identification, CPAM - Continuous Probability Acoustic Map, mixtures of Gaussian kernels with and without HMM. (64)
MLP	Gong and Haton (1992)	72, 89 - 100%, Trained MLP to interpolate between speaker utterances (phoneme), needs labeled speech (vowels).
VQ	Kao et al (1992)	26 (51), 93.3% (67.6), KING corpus, 11 broad class codebooks of 10 vectors, Needs labeled speech. (26)

as well as its modification to perform the verification function, will be provided in Chapter 3 of this document.

2.4 Multiple Sensor Fusion

Humans are able to quite easily integrate information from different senses (hearing and sight, for example) and make decisions based on that integration. Combining information in machines, however, can be somewhat more problematic. This task of automating the integration of multiple sensors is commonly known as sensor fusion, and is defined by Thomoupoulous as

> . . . the process of integrating raw and processed data into some form of meaningful inference that can be used intelligently to improve the performance of the system, measured in any convenient and quantifiable way, beyond the level that any one of the components of the system separately or any subset of the system components partially combined could achieve. (62)

Three general schools of thought exist for the fusion of information from multiple sensors, and are described in the following sections and illustrated in Figure 2.6 (28).

Fusion of Observations Each individual sensor i provides an observation vector s_i to a centralized decision unit D that determines the decision probability

$$q = P(\omega = 1|s_1, \cdots, s_n) = D(s_1, \cdots, s_n),$$

where ω is the class being considered. q is provided to a decider within D that will make the classification decision.

Fusion of Decisions With this method, each sensor i is provided with its own forecaster, H_i, that determines the single sensor-based probability

$$x_i = P(\omega = 1|s_i) = H_i(s_i).$$

x_i is then mapped by a decider D_i into a binary decision vector $(a_i, \cdots, a_n)$, which is presented to a fusion rule prescribing the final decision. Reibman and Nolte (44, 43) and Chair and Varshney (7) have proposed methods to optimize both the fusion rule and the individual sensor decision unit rules.

Fusion of Probabilities As with the second method, each sensor i is provided with it's own forecaster H_i that maps an observation vector into a classification probability $x_i = P(\omega = 1|s_i) = H_i(s_i)$. The resultant vector of the individual probabilities is then input into a fusion rule H that provides the posteriori fused classification probability

$$p = P(\omega = 1|x_1, \cdots, x_n) = H(H_1(s_1), \cdots, H_n(s_n)).$$

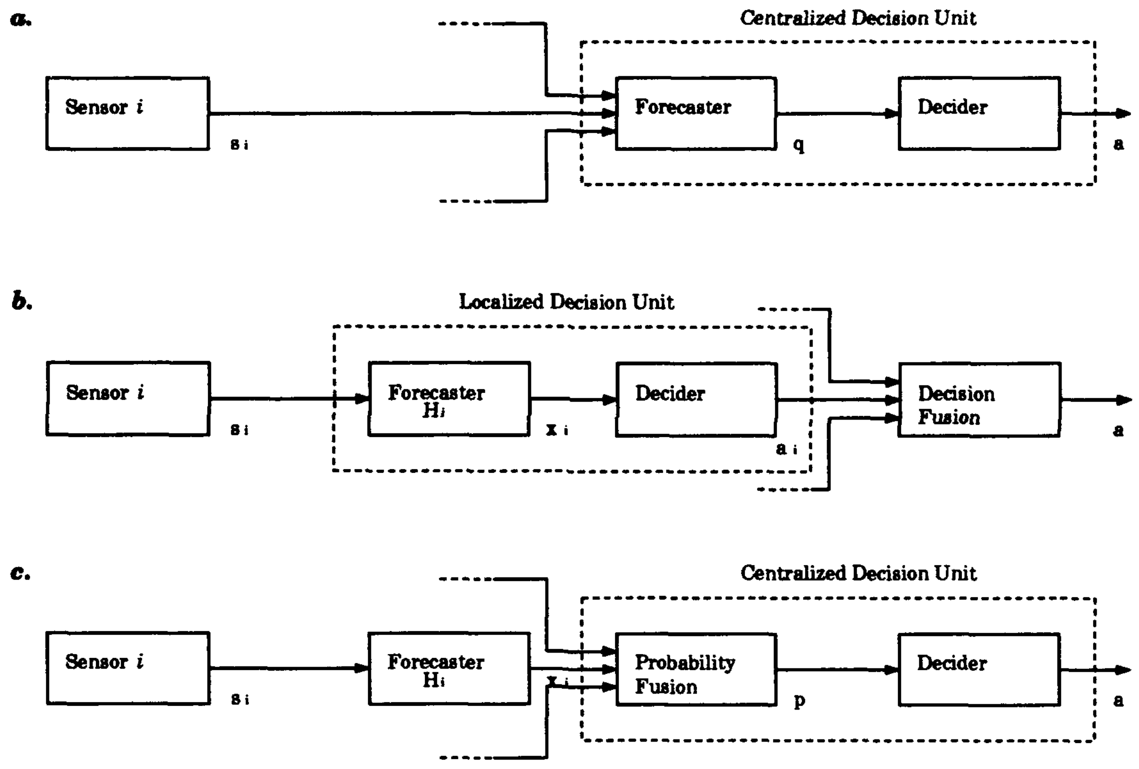

Figure 2.6 Three methods of fusing information from multiple sensors: a) Fusion of Observations, b) Fusion of Decisions, c) Fusion of Probabilities

2.5 Conclusion

This search of the current literature has briefly outlined research efforts into the areas of face recognition, speaker identification, and multi-sensor integration or fusion. The latter two areas have received significant attention over the last few years, and research into the first is becoming increasingly prevalent. We have found that identification systems based on either faces or speech have seen considerable success, but that all of the problems inherent to such a task have not yet been solved. We have not found any attempt to combine the capabilities of face and speech recognition systems into a single, cohesive unit, but have seen that the general problem of fusing information from multiple sources has been addressed and successful solutions have been developed; such methodologies should be applicable to the identity verification problem being addressed by this thesis.

The next chapter shall present the methodology to be followed in the performance of this research, and Appendix A provides the actual implementation techniques used and the software developed and modified for this effort.

III. Methodology

3.1 Introduction

The system to be developed in this research is based to a large degree on previous and concurrent AFIT thesis efforts. The fundamental building blocks include a neural-net based face recognizer, detailed in theses by Krepp, Runyon, and Gay (27, 52, 17), and a distortion based speaker identifier, described by Colombi (9). These systems will be modified to perform the verification task, and will then be fused to form a multiple sensor verification mechanism. Techniques to enhance the operation of the face verifier portion of the system will be explored in this thesis; efforts to improve the speaker verifier will be addressed in a collateral thesis by Prescott (41).

The remainder of this chapter will be organized as follows: First, the basic identification systems will be described, followed by a description of the efforts needed to modify the systems to function as verifiers. Next, the technique to be used to fuse the disparate verification systems will be presented, and finally methods to improve the performance of the overall system will be addressed.

3.2 Verification Building Blocks

3.2.1 Face Recognizer. The face recognizer used in this research is based on the system developed by Turk and Pentland at the Massachusetts Institute of Technology, as well as work conducted at AFIT by Suarez, Goble, et al. (65, 61, 18, 19, 27, 52). An $n \times n$ image of a face is converted into a vector of length n^2, and this high dimensional vector is then projected into a lower dimensional space via the Karhunen-Loéve Transform (KLT). The coefficients describing this new, reduced vector are then presented to a back-propagation neural network for classification.

3.2.1.1 Segmentation. The segmentation portion of the system has been approached in two fundamental ways. In the first, the faces within the image are manually segmented, producing a very constant placement of the face within the image. When using this method, the assumption is that a technique will be made available at some point automate this placement, a non-trivial task. The second method addresses this subject, and involves a frame differencing technique explored by Gay (17). Using the notion that no one is able to keep his or her face perfectly still, two successive image frames of the target are captured, and one is subtracted from the other. The resultant motion image is scanned for head shapes, and if one is found, it will be used as a template to locate the face within the image. The located face will then be enlarged and moved to a standard position in the image.

McCrae has explored an additional segmentation methodology in a concurrent thesis (36). She uses a neural net based color segmentation scheme to detect faces within a color image by discriminating between 'face' color and 'non-face'color; she is then able to detect the eyes within the faces using a similar approach. An example of face segmentation and eye detection using the color discrimination system is provided in Figure 3.1.

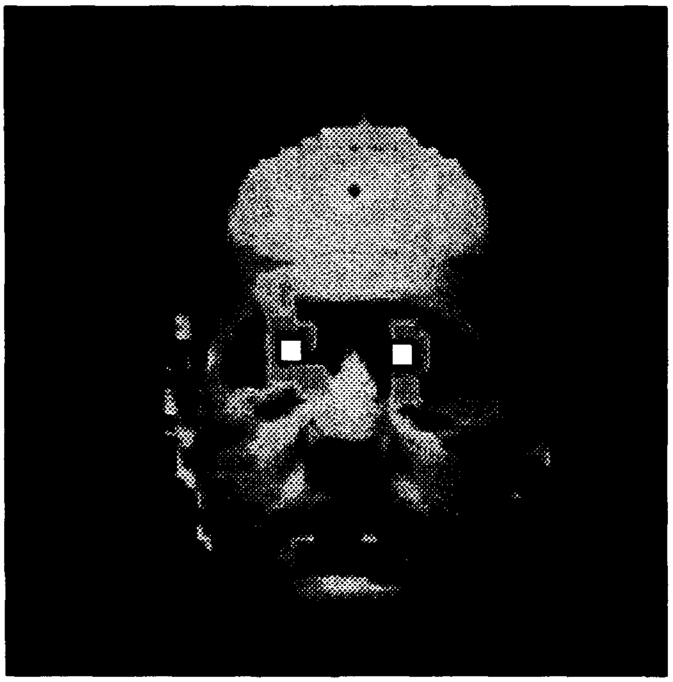

Figure 3.1 Eye point detection using McCrae's system (36).

In the course of this research we also briefly examined face segmentation using a three-dimensional temporal wavelet introduced by Burns in his Doctoral dissertation (6). Such a wavelet is able to detect motion within a sequence of two-dimensional images by analyzing the frequency content over time (the succession of images) and space, rather than strictly space. Figure 3.2 shows one of the original source images from a sequence showing a person nodding their head, and Figure 3.3 demonstrates the result of performing one level of a wavelet decomposition. 3.3a shows the approximation image, and 3.3b, 3.3c, and 3.3d show , respectively, the horizontal, vertical, and diagonal high frequency components of the images over time. This technique appears to do a good job of decreasing the importance of the background, and holds great promise in the motion segmentation arena.

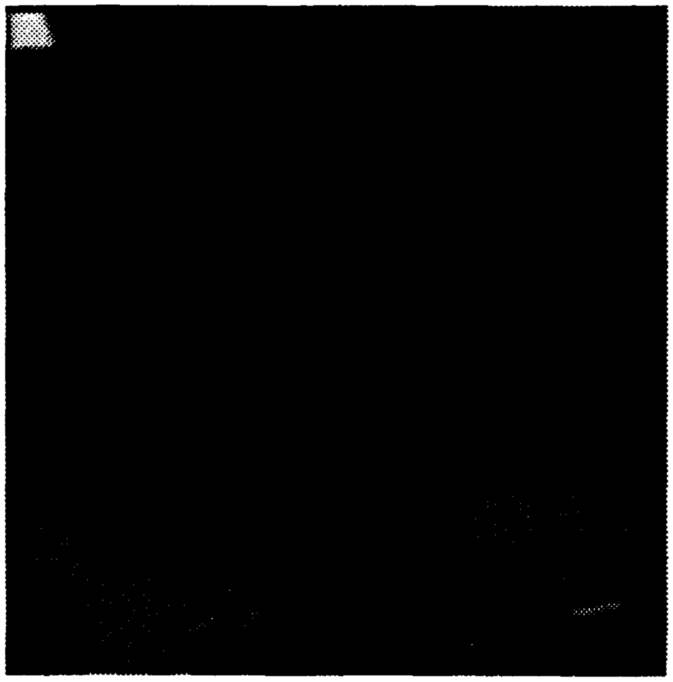

Figure 3.2 Sample of source image before temporal wavelet decomposition.

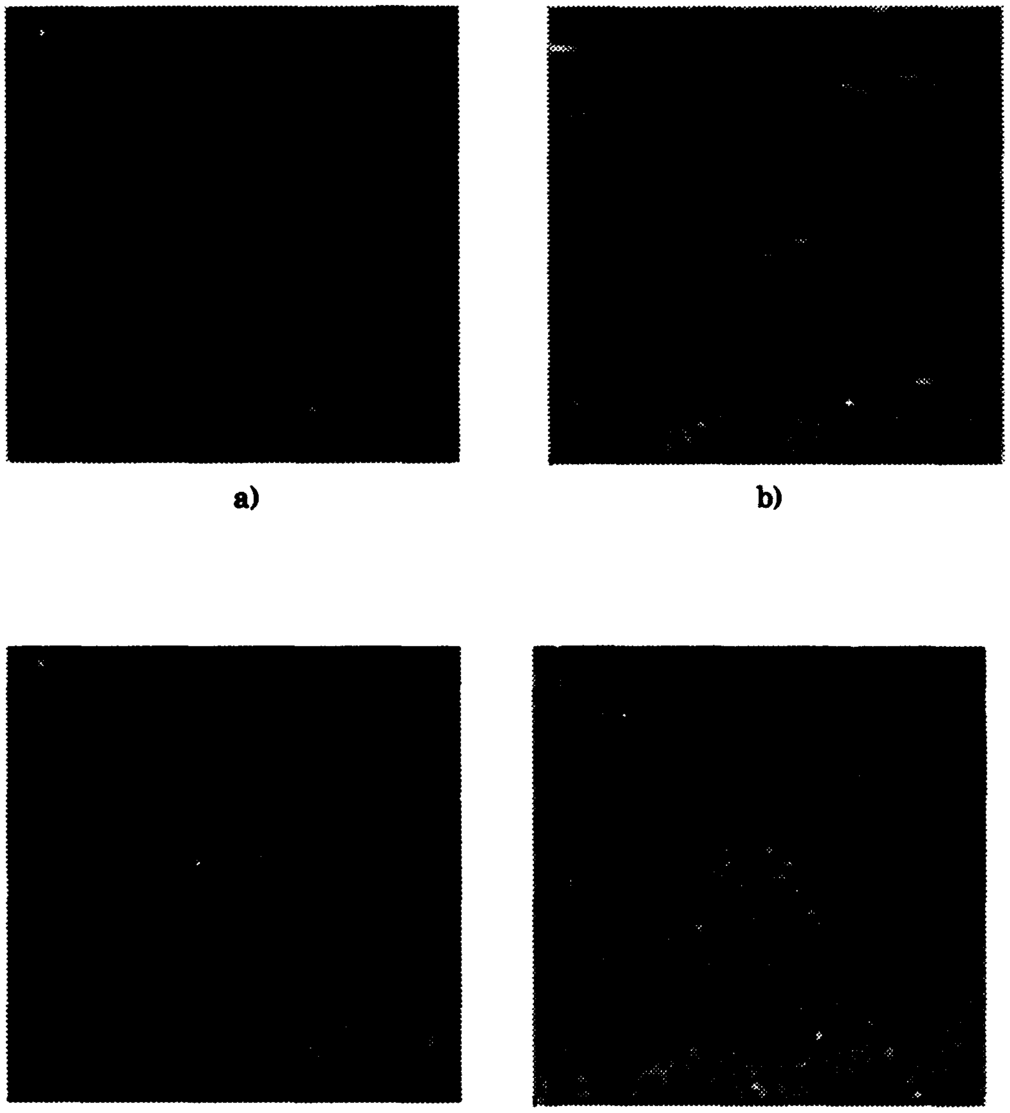

Figure 3.3 Result of one level of temporal wavelet decomposition.

3.2.1.2 Feature Reduction via the Karhunen-Loève Transform. When working with an $n \times n$ image, the holistic recognition approach has led to a traditional feature set consisting of the intensity value of each of the n^2 pixels within the image. A problem with this approach, however, is that we are left with a feature vector of dimensionality n^2, which may become extremely unwieldy as we attempt to perform some training/classification process. Therefore, it behooves us to find an intelligent method to reduce that dimensionality without sacrificing the information inherent to the original feature set. Of equal importance, we would like to find some way to determine which of these reduced features are most important; that is, which features best differentiate images of different targets[1].

Certain data transformations have been developed that provide us with methods of identifying which feature, or combination of features, best allow such differentiation. In classification problems where one does not know the various probability densities of the classes being studied, methods of orthogonal expansion have proven quite useful in providing a new feature space in which to project the existing features (63:269). A Fourier series expansion allows such a projection of periodic processes, but certain conditions must be met to use them with nonperiodic processes such as our faces[2]. The Karhunen-Loéve Transform (KLT), on the other hand, allows a non-periodic random process to be expressed as a series of orthogonal functions with uncorrelated coefficients, and will prove useful in our classification problem.

While the basis set of the Fourier Transform consists of sines and cosines, the basis set of the KLT is made up of the eigenvectors of the covariance matrix of the source data. The derivation is relatively straightforward, and can be stated in words as follows:

1. Compute the mean and covariance functions from the population of training images.
2. Compute the eigenvalues of the statistically normalized covariance matrix, where statistically normalized simply means the mean image has been subtracted from each individual image in the training population.
3. Calculate the eigenvector corresponding to each eigenvalue computed in the previous step.
4. To reduce the dimensionality of the original images to some value k, select the k eigenvectors corresponding to the k largest eigenvalues and matrix multiply each

[1]We should mention at this point that if we are performing classification using neural networks (to be discussed in the next section), Ruck, et al, have developed a *saliency measure* to determine the input features with the greatest influence on the output (50). Our purpose here is to determine these features before attempting classification.

[2]If an assumption is made that the non-periodic sequence under study is actually one period of a periodic sequence, Fourier techniques can be used. Goble showed that under such an assumption, the Discrete Cosine Transform was indeed effective in providing a recognition capability (18).

image vector by the eigenvectors. The k eigenvectors form the new KLT basis set, and any n^2 dimensional image may now be reduced to k coefficients of the basis set. Let's now look at the specifics of the KLT process. First define the n^2 dimensional image as a feature vector

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

where $\mathbf{x}^i$ is the ith image within the population. The mean vector of the training population is defined as

$$\mathbf{m}_x = \frac{1}{M}\sum_{i=1}^{M}\mathbf{x}^i$$

and the covariance matrix as

$$\mathbf{c}_x = \frac{1}{M}\sum_{i=1}^{M}\mathbf{x}^i\mathbf{x}^{iT} - \mathbf{m}_x\mathbf{m}_x{}^T$$

After subtracting the mean image from all the source images, we are left with a covariance matrix of the form

$$\mathbf{c}_x = \frac{1}{M}\begin{pmatrix} \sum_{i=1}^{M} x_1^i x_1^i & \sum_{i=1}^{M} x_1^i x_2^i & \cdots & \sum_{i=1}^{M} x_1^i x_{n^2}^i \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{M} x_{n^2}^i x_1^i & \sum_{i=1}^{M} x_{n^2}^i x_2^i & \cdots & \sum_{i=1}^{M} x_{n^2}^i x_{n^2}^i \end{pmatrix}$$

This covariance matrix provides us with a measure of the importance of each dimension within the original set of images. A high variance in a single dimension across all images indicates a large amount of differentiability information in that dimension; conversely, a small variance in a dimension indicates only a small amount of information. Our goal, then, is to maximize the values in the diagonal of the covariance matrix (corresponding to single-dimension variance), and minimize the variation in all other locations (corresponding to co-dimensional variance), thereby orthogonalizing the multi-dimensional space. This can be accomplished by finding the eigenvectors (and eigenvalues) of the covariance matrix, thus transforming the matrix into an orthogonal space and ensuring an optimal distance between dimensions (optimality is implied by orthogonality in this case).

We can then rank the eigenvectors in descending eigenvalue order, placing them in the eigenmatrix

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

This matrix is our new basis set, with k orthogonal dimensions into which any n^2 image can be transformed. This transformation is accomplished by the simple process

$$\mathbf{y}_k = (\mathbf{x}_k - \mathbf{m}_x)\mathbf{u}_k,\ k = 1 \ldots M$$

where $\mathbf{y}_k$ represents the projection of the original image $\mathbf{x}_k$ into our new eigenspace. Note that the selection of k is usually dependent upon the desired reconstruction accuracy. Krepp found that 32 × 32 face images were sufficiently represented by twenty KLT coefficients for classification to an accuracy of 97.0% (based on 300 images of 10 different people); this implementation out-performed the system when using all pixel values, where an accuracy of 93.3% was attained. Therefore, he was able to reduce an $n^2 = 1024$ dimensional feature vector into one of only 20 dimensions and still obtain excellent classification performance. This is the feature extraction approach used for the base face recognizer building block, but because we are not necessarily looking for optimal face verification performance (we are instead more interested in the performance resulting from the fusion of two verification systems) we will use a reduced number of KLT coefficients. That number will be determined as part of the testing process, to be elaborated on in the next chapter.

3.2.1.3 Neural Net Classification. The use of artificial neural networks for face recognition is not uncommon in recent research, and networks have indeed been found to function quite effectively in the face classification task (5, 53, 14, 11, 27, 52). An excellent treatment of the history, development, and function of artificial neural networks is provided by Rogers and Kabrisky, and should be referenced for an in-depth examination of the subject (45). We shall provide a brief look here at the reasons for using a neural net for classification, and describe the network developed for the face recognizer used in this research.

The artificial neural network is meant to emulate, in some sense, the function of the biological brain. Nodes are developed that model the neurons within the brain, permitting some functional transformation on the inputs, and weighted interconnections are established to provide communication between the nodes, somewhat analogously to the way in which the biological neural system provides communication between neurons via axons (neural outputs) and dendrites (neural inputs). Figure 3.4 illustrates the implementation of a single node of a neural network and shows the multiple-input, single-output nature of the artificial (and biological) neuron. This particular implementation is also known as the single layer perceptron, and provides as its output a function of a linear combination of the weighted inputs, with weights indicated by w_1 through w_n, and a threshold bias θ. Figure 3.5 shows a complete neural network, with an input layer, an output layer, and one 'hidden' layer.

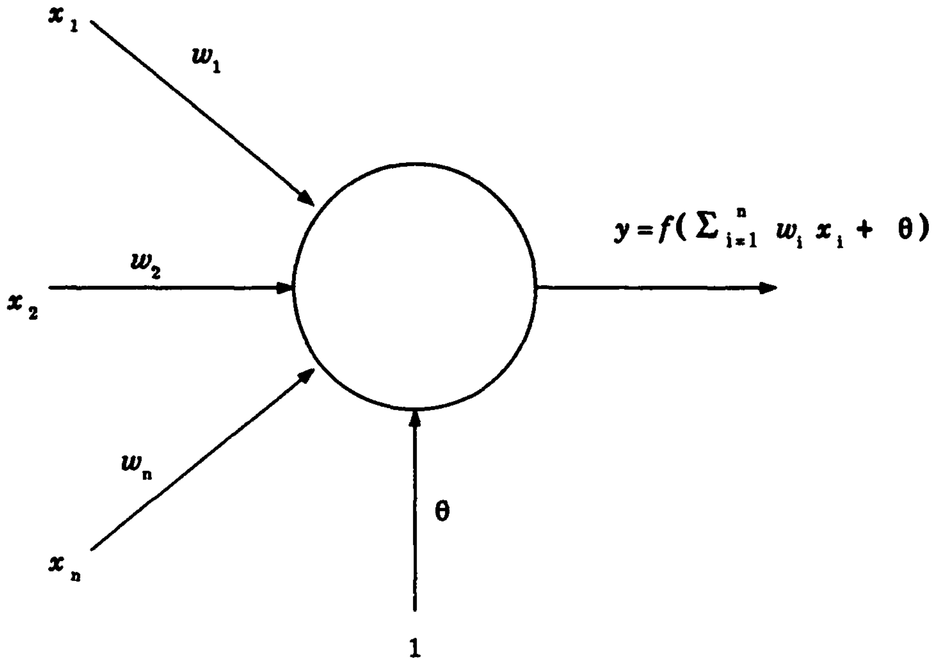

Figure 3.4 A single artificial neuron (perceptron).

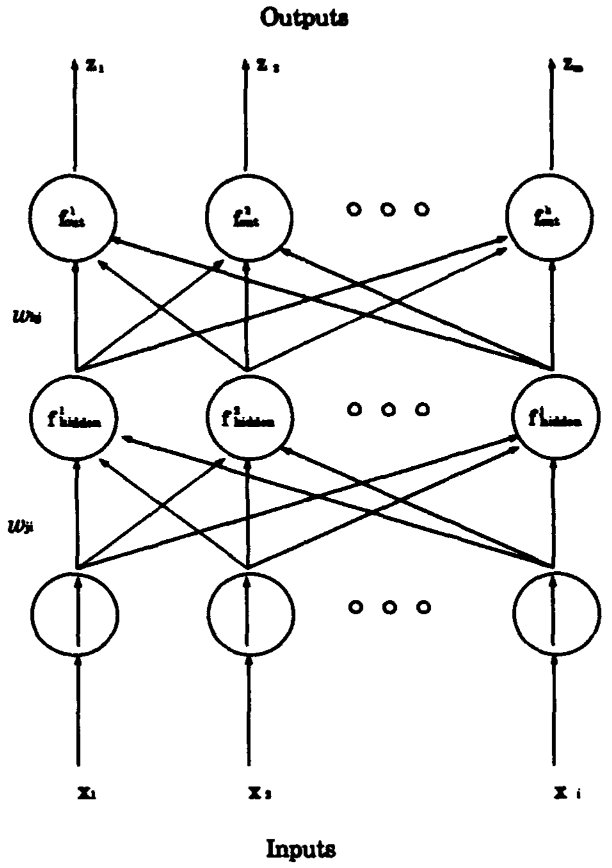

Figure 3.5 A fully connected, three layer neural net.

A major advantage of artificially implementing such a system is that the transformation, or activation, within each node can be selected by the user, allowing one to effectively implement any mathematical function desired. This ability becomes vital as one attempts to perform classification of data that is not linearly separable; Cybenko has shown that, given sufficient nodes in a single hidden layer and non-linear activations within those nodes, any continuous function can be approximated (12). This implies that, given sufficient resources, any separable data set may be properly classified by such an artificial neural network.

The basic function of a neural network is conceptually fairly straightforward, and generally relies on a training process to determine the appropriate setting of the interconnection weights. Once the net has been trained to produce a specific result when presented some family, or class, of data, our goal is to be able to present new data to the net, and, if this new data belongs to one of the classes previously trained on, properly classify the new pattern based on the net output.

During the training process, pattern vectors are presented one at a time to the input nodes of the net and allowed to propagate through the interconnection weights, any hidden nodes (and their associated transformations), and finally through the set of output nodes (and their transformations). The output values produced for each vector are compared to a desired set of output values (determinable because we know the classes to which our training vectors belong) and the net weights are updated based on the difference between the desired and actual outputs. The update may occur after each vector is presented, or in a batch manner, after all the training data are presented. This process is repeated until the error between the actual and desired outputs is reduced to some desired level. At this point, all the interconnection weights are saved, and when we wish to classify a new data pattern, we rebuild the net using the saved weights, propagate the unknown pattern through the net, and examine the output.

A simple example would be illustrative at this point. Consider training a net on two classes of data using multiple sample vectors from each class. Because we only have two classes to differentiate, we can establish an output layer consisting of just two nodes. The input layer will be composed of the same number of nodes as there are features (also known as dimensions) in the input pattern, and the hidden layer(s) may contain varying numbers of nodes [3]. If the net is presented a pattern from Class 1, we wish to produce an output value of 1 from output node 1, and 0 from output node 2. When a pattern from Class 2 is presented we wish to see the converse: a value of 0 from output node 1 and 1 from output node 2. The difference between these desired outputs and the actual outputs are calculated, and the weights are updated using some learning rule.

Once we have reduced the differences (across all training patterns) to a satisfactory level, we can present a previously unseen pattern from one of the two classes to the net; a new pattern belonging to a particular class should produce outputs similar to those

[3]Determination of the appropriate number is not always apparent; a discussion of the subject is presented in Appendix A of (45)

produced by the training data of the same class. If this is not the case, either the net was not allowed to train to a sufficiently low error value, or the test vector presented was not representative of the class of data on which the net was trained.

The face recognition neural network used here is a three-layer (input, output, and one hidden) network from the class of multi-layer perceptrons as shown in Figure 3.5. The inputs to the net are the KLT coefficients calculated from the original face images, and the outputs represent the identities of the persons to be recognized. The learning rule for weight updates on this particular net is based on the common backward error propagation (or back-prop) algorithm, with which the output error is used to propagate a correction to the weights back through the net . The hidden and output layer nodes all use sigmoid functions as their activations, providing non-linear separation capability as well as an easily implemented update rule. Given some error measure E as the output of the net, the generic gradient descent weight update rule is

$$w^+ = w^- - \eta \frac{\partial E}{\partial w}$$

and the rule we shall use can be stated simply as

$$w_{ji}^+ = w_{ji}^- + \eta \delta_j Y_i$$

where w_{ji} represents the weight connecting node j to node i in the previous layer, δ_j is an error factor associated with the outputs of node j, Y_i is the output of node i, and η is generally a variable learning rate. w_{ji}^+ represents the updated weight value, and w_{ji}^- represents the weight value before the update. The δs are easily calculated for a net with k output nodes, j hidden layer nodes, and i input nodes:

Define the error output as

$$E = \sum_k \frac{1}{2}(D_k - Z_k)^2$$

where D_k is the desired output from node k and Z_k is the actual output from node k. We wish to minimize this error function, and thus will use the derivative of the function to perform a gradient descent search for the update.

$$\begin{aligned}\frac{\partial E}{\partial w_{kj}} &= \frac{\partial}{\partial w_{kj}} \sum_k \frac{1}{2}(D_k - Z_k)^2 \\ &= (D_k - Z_k)\frac{\partial}{\partial w_{kj}}(D_k - Z_k)\end{aligned}$$

Notice the derivative is being taken with respect to a specific weight connecting a particular hidden node j with a particular output node k, so the summation over k reduced to only a single value. Let's now rewrite the output Z, recalling that the activations for our hidden and output nodes are sigmoids:

$$Z_k = \frac{1}{1 + e^{-\sum_j w_{kj} Y_j}}$$

where Y_j is the output of hidden node j. Then we can say

$$\begin{aligned}\frac{\partial}{\partial w_{kj}}(D_k - Z_k) &= \frac{\partial}{\partial w_{kj}}\left(D_k - \frac{1}{1 + e^{-\sum_j w_{kj} Y_j}}\right) \\ &= -\left(\frac{1}{\left(1 + e^{-\sum_j w_{kj} Y_j}\right)^2}\right)\left(e^{-\sum_j w_{kj} Y_j}\right) Y_j \\ &= -\left(\frac{1}{1 + e^{-\sum_j w_{kj} Y_j}}\right)\left(\frac{e^{-\sum_j w_{kj} Y_j}}{1 + e^{-\sum_j w_{kj} Y_j}}\right) Y_j\end{aligned}$$

After some slight mathematical manipulation, we can rewrite the derivative as

$$\frac{\partial}{\partial w_{kj}}(D_k - Z_k) = -Z_k(1 - Z_k)Y_j$$

Therefore

$$\frac{\partial E}{\partial w_{kj}} = -(D_k - Z_k)Z_k(1 - Z_k)Y_j$$

If we now define $\delta_k = (D_k - Z_k)Z_k(1 - Z_k)$, then we can state the update rule for the weights between hidden layer and output as

$$w_{kj}^{+} = w_{kj}^{-} - \eta_{out}\delta_k Y_j$$

We can derive the update rule for the weights between the input and hidden layers in a similar manner:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_k \frac{1}{2}(D_k - Z_k)^2 = \sum_k (D_k - Z_k) \frac{\partial}{\partial w_{ji}} (D_k - Z_k)$$

In this case, the summation over k is retained, as the derivative is being taken with respect to a weight connecting nodes i and j.

$$\begin{aligned} \frac{\partial}{\partial w_{ji}}(D_k - Z_k) &= -\left(\frac{1}{\left(1+e^{-\sum_j w_{kj}Y_j}\right)^2}\right)\left(e^{-\sum_j w_{kj}Y_j}\right)\left(\frac{\partial}{\partial w_{ji}}Y_j\right) \\ &= -Z_k(1 - Z_k)(w_{kj})\left(\frac{\partial}{\partial w_{ji}}Y_j\right) \end{aligned}$$

$$\begin{aligned} \frac{\partial}{\partial w_{ji}}Y_j &= \frac{\partial}{\partial w_{ji}}\left(\frac{1}{1+e^{-\sum_i w_{ji}X_i}}\right) \\ &= \left(\frac{1}{(1+e^{-\sum_j w_{ji}X_i})^2}\right)\left(e^{-\sum_j w_{ji}X_i}\right)X_i \\ &= Y_j(1 - Y_j)X_i \end{aligned}$$

So

$$\frac{\partial E}{\partial w_{ji}} = -\sum_k (D_k - Z_k)(Z_k)(1 - Z_k)(w_{kj})(Y_j)(1 - Y_j)X_i$$

We can now denote an error factor δ_j in terms of the output delta calculated previously, δ_k: $\delta_j = \sum_k \delta_k (w_{kj})(Y_j)(1 - Y_j)$. This allows us to state our update rule for the weights between the input and hidden layer as

$$\begin{aligned} w_{ji}^{+} &= w_{ji}^{-} - \eta_{in} \sum_k \delta_k (w_{kj})(Y_j)(1 - Y_j) X_i \\ &= w_{ji}^{-} - \delta_j X_i \end{aligned}$$

Note that we may also have implemented our neural net with linear outputs while retaining the sigmoidal hidden nodes. For that case, the update rule for the weights between the input and hidden layers will remain the same, but the rule will change for updating the weights between the hidden and output layers. The error function will still be

$$E = \sum_k \frac{1}{2}(D_k - Z_k)^2,$$

but the output Z at each node will now be represented by

$$Z_k = \sum_j w_{kj} Y_j$$

leading to the derivative with respect to w_{kj}

$$\begin{aligned} \frac{\partial}{\partial w_{kj}}(D_k - Z_k) &= (D_k - Z_k)\frac{\partial}{\partial w_{kj}}(D_k - Z_k) \\ &= (D_k - Z_k)\frac{\partial}{\partial w_{kj}}(D_k - w_{kj} Y_j) \\ &= -(D_k - Z_k) Y_j \end{aligned}$$

With the new definition $\delta_k = -(D_k - Z_k)$, our update rule for the net with linear outputs is still

$$w_{kj}^{+} = w_{kj}^{-} - \eta_{out} \delta_k Y_j$$

Figures 3.6 and 3.7 illustrate the function of the complete face recognition system used as a building block for our identity verification implementation. Figure 3.6 shows the process of determining the appropriate lower-dimensionality feature space for the given training population, and Figure 3.7 demonstrates the actual recognition process. The faces are first manually centered in the input images[4], then an eigen-analysis is performed on the training population to determine an orthogonal feature space into which to project the images. The dimensions to be retained become the new basis set, and are known as eigenfaces, because they represent the eigenvectors of the original training population. Figure 3.8 shows examples of eigenfaces extracted from a training set consisting of six people. Using this basis set, coefficients are extracted from each image and presented to the neural net for training. Each node of the output represents one person to be recognized, and once the net is trained, presentation of a new instance of one of the faces the system was trained on should result in the 'firing' of the appropriate node.

[4]Recall that a collateral thesis by McCrae is solving the automatic segmentation problem (36).

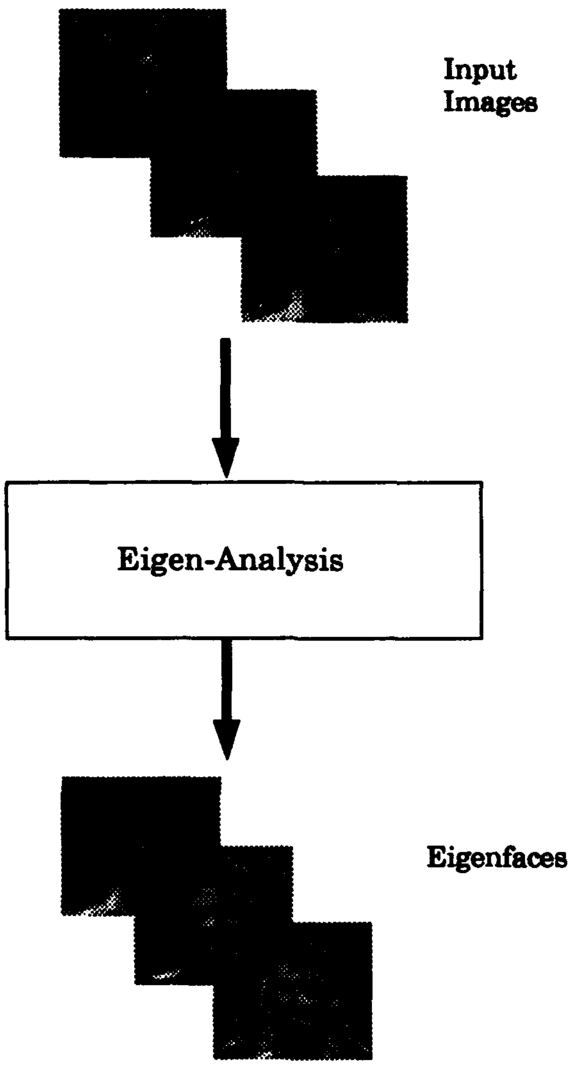

Figure 3.6 Determination of the orthogonal feature space.

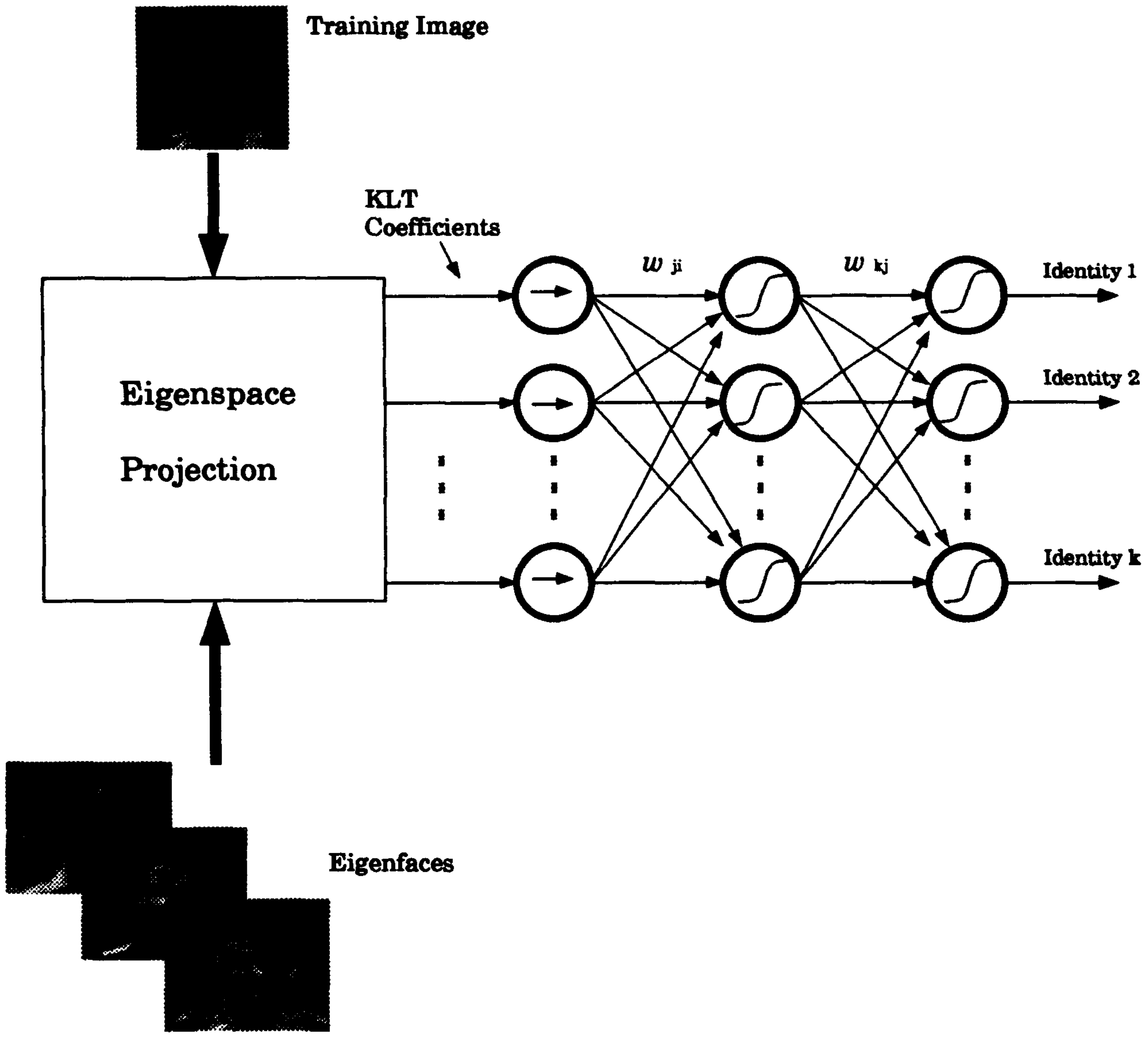

Figure 3.7 The face recognition system used in this research.

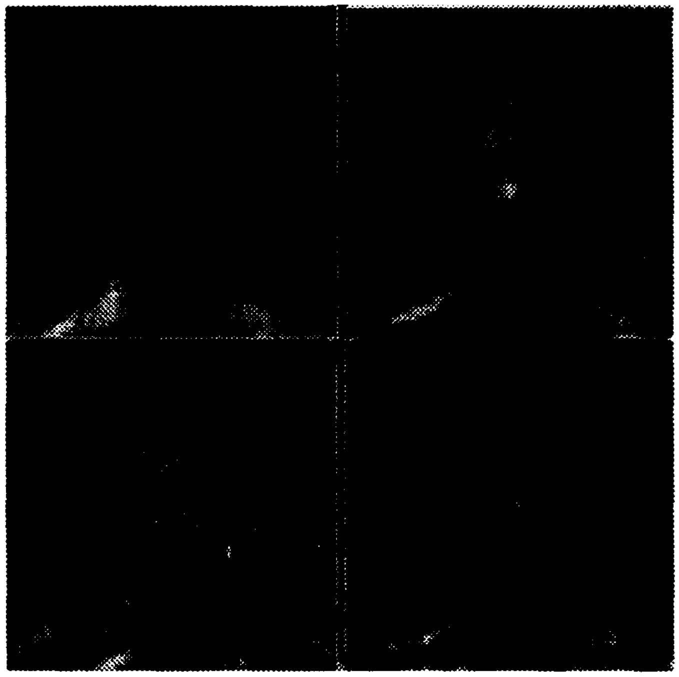

Figure 3.8 Four example eigenfaces produced by a training set consisting of six examples each of six different people.

3.2.2 Speaker Identifier. The base speaker identifier used for this verification system was developed at AFIT by Colombi in a 1992 thesis effort, and is based on the extraction of cepstral coefficients from individual utterances (9)[5]. These coefficients are used to build a codebook containing a set of code vectors (coefficient vectors) representative of the vocal frequency range of the person to be identified. When that person presents new utterances to the system for recognition, cepstral coefficients are extracted from the new speech and a distortion metric is used to measure the distance from the new vectors and those in the codebook. An overview of the system is provided in the following section.

3.2.2.1 Training the System. Figure 3.9 illustrates the procedure followed to train the system for a particular individual. Using the audio input capabilities of the Sun SPARCstation, frames of approximately 30 milliseconds of speech (overlapped by 50%, as illustrated in Figure 3.10) are digitally sampled at 8000 samples/second, and then

[5]Furui provides an excellent discussion of the theory and application of Cepstral coefficient-based speech processing in (15).

processed through a pre-emphasis network to boost the high-frequency components of the speech.

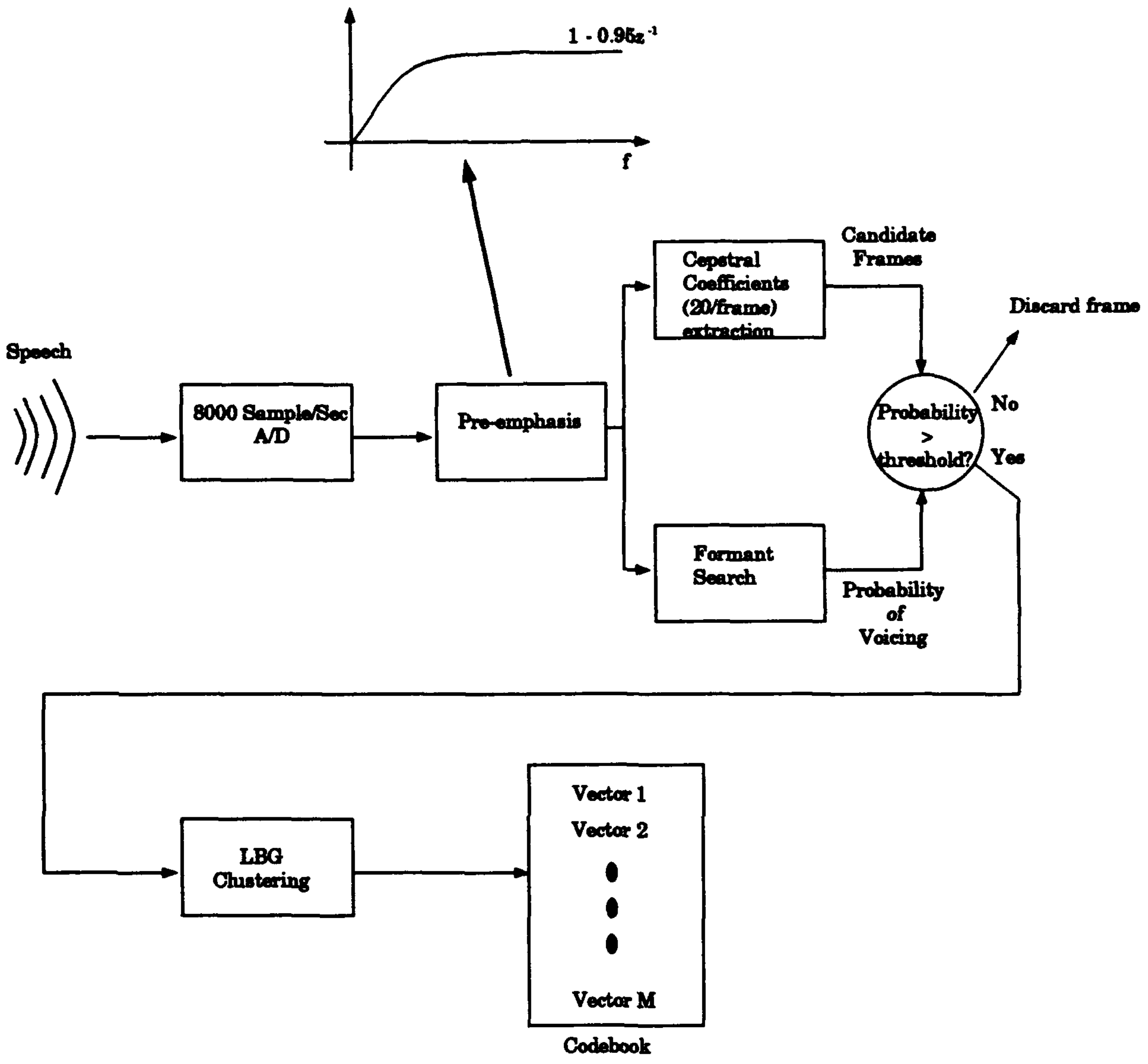

Figure 3.9 Training the speaker identification system.

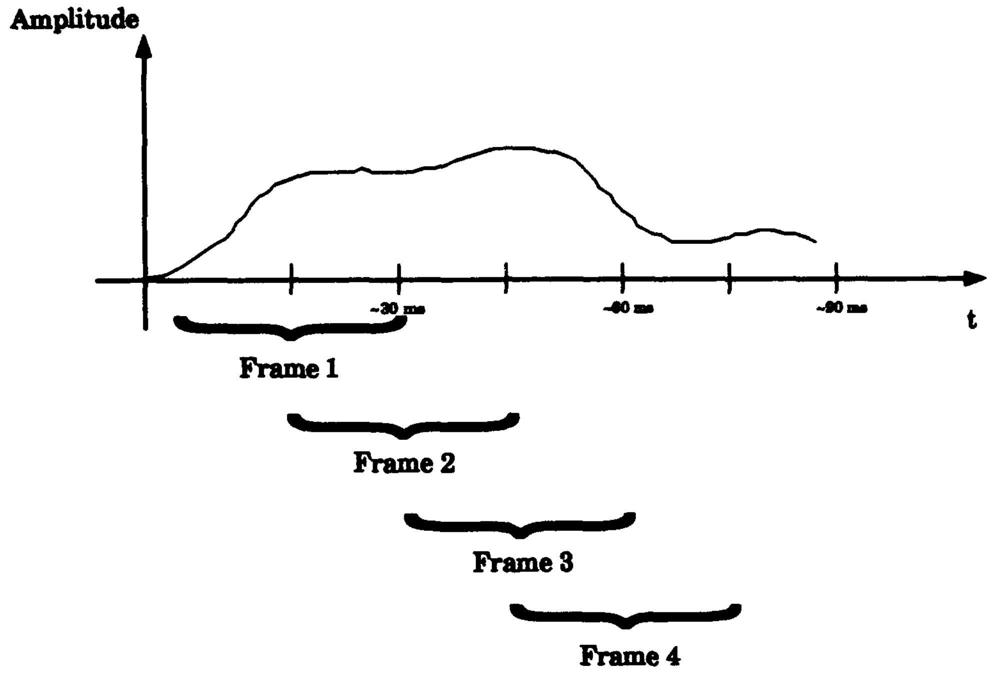

Figure 3.10 Illustration of the overlap of speech frames (each frame ~30 milliseconds long).

Twenty Cepstral coefficients are extracted from each of the resultant frames by the Entropic Signal Processing System (ESPS©) software package, and then ESPS is used to determine a probability of voicing factor for each of these frames. A probability of voicing at or above some threshold indicates the presence of formants, which provide a better correlation with an individual's vocal tract than do fricatives, and using such a threshold will allow us to discard the frames containing only fricatives or silence. All of the frames that are retained are then presented to a Linde-Buzo-Gray clustering algorithm, which is used to transform the initial coefficient vectors into a set of 64 new cluster centers, or codewords, which are placed in a codebook representing the individual (30).

3.2.2.2 Speaker Recognition. The first stages of the recognition process are identical to those of the training process. The individual to be recognized will provide an utterance to the system, and the speech will be put into frames, digitized, Cepstral processed, and the frames with a sufficient probability of voicing will be retained. Each of these new vectors will be presented to each of the codebooks representing the training population, and distortion measures will be made of the distances from the new speech to each of the codebooks. The distortion measure from each codebook is calculated by

$$distortion = \sum_{j=1}^{N} \sum_{i=1}^{M} min_{i \in M} [d(X_j, V_i)]$$

where N is the number of new speech vectors, M is the number of codewords in the codebook, and $min_{i \in M} [d(X_j, V_i)]$ is the minimum of the Euclidean distances between the new vectors X_j and the codebook vectors V_i. The new speaker is then recognized as the individual whose codebook corresponds to the lowest distortion. Xu showed that pseudo post-probabilities could be calculated from the results of the distortion measurement process through the simple metric

$$p_k(i) = \frac{\frac{1}{distortion_k(i)}}{\sum_{i=1}^{M} \frac{1}{distortion_k(i)}}$$

where i is the class under consideration, k is the classifier (in this case, one of the codebooks), and M is the total number of classes (66).

3.3 Conversion From Identification to Verification

Verification is generally an easier problem to solve than identification, but we must recognize that the two processes are distinct, and can be approached differently.

For identification, we generally begin with a database containing the identities of known individuals and information that can be used to classify new instances of these individuals; the identity of a new instance will be selected from the database based on some set of characteristics and some selection rule. The assumption made is that the person to be identified is present in the database already, or put another way, that the database contains the entire population of possible targets. To overcome this limitation, it is possible to set an acceptance threshold so that an identification will be made only if some minimum error criteria is met; it would be perfectly valid to perform a statistical analysis of the performance of the system for each individual in the database to determine such a threshold [6]. This can become quite cumbersome, however, as the number of individuals and the amount of information in the database increases.

In the verification problem, the assumption is again inherent that the database contains information about every possible target, but we now have a different classification criteria: we may either decide that the target is who he claims to be, or is not, and need not concern ourselves with who he actually is. As with identification, we could again statistically determine thresholds to establish confidence in the performance of our verification system, but that would still be a computationally intensive and time consuming task. We would prefer to find a way to actually represent the possible target population as a whole, and to train our system to determine what makes a new instance of a known target different from another member of that target population.

The approach made in this research is to attempt to do just that: model the average population and train our verification system to recognize what makes an individual stand out from that model. We will use the same back-prop neural net as was used for face recognition for this effort, but instead of training it to differentiate one individual from another, we will train it to differentiate each individual from the entire set of other individuals; one could say that we are treating all the non-target individuals as different instances of a single person. The hope is that, given enough individuals in the data base, we will be able to successfully model the 'average world' person, and thus discriminate the target from this person.

3.3.1 Face Verifier. To provide a face verification capability, we will break down our population of training images into two classes: the person to be verified and everyone else. If we begin with a training set consisting of k prototypes each of N individuals, then one training class will consist of k images, and the other of $(N-1)k$ images. As with the basis system, the training population will be used to form an orthogonal space, and KLT coefficients for each image will be extracted. These coefficients will be presented to the

[6]We may envision the analysis as requiring 'many' new instances of each individual to be presented to the system, and recording the number of correct and false recognitions. A threshold could be established to minimize the error for each individual based on this analysis.

neural net, along with a tag identifying each set as either belonging to the target individual or not, and the net will be trained to provide proper classification.

After the net is trained for a specific individual, the weights calculated will be stored in a file specific to him, and will be used to rebuild the net whenever an individual presents himself to the system claiming to be that person. Ruck, et al, has shown that the outputs of a multi-layer perceptron configured as ours will approximate the a posteriori probabilities of being in a specified class (50). For the recognition problem, where equal numbers of protoypes for each class are presented for training, the probability of being in a certain class is simply

$$prob_m = \frac{Z_m}{\sum_{i=1}^{K} Z_i}$$

where Z_m is the output of a specific node in the output layer, and K is the number of nodes in that layer. A source of bias exists in the verification case in that there will be many more instances of 'not-Joe' than there will be of 'Joe' presented to the net for training (in fact there will be $N - 1$ times more instances). This bias must be accounted for at the net output if we wish to classify in terms of post-probabilities, which will be quite useful when attempting to fuse our two basis verification systems. Hush and Horne have shown a simple technique for compensating for such a bias.

> If the training set distribution does not accurately reflect the actual a priori probabilities, the network outputs can be scaled to compensate . . . The proper adjustment can be made by scaling the estimate of $P(\omega_i|x)$ by $P(\omega_i)/P_t(\omega_i)$, where $P(\omega_i)$ is the true a priori probability, and $P_t(\omega_i)$ is the a priori probability implied by the training set distribution (24).

In our case, we will scale the probability of a correct verification calculated at the output of the neural net by

$$\frac{0.5}{(\text{number of target vectors})/(\text{number of non target vectors})} = \frac{0.5 * \text{number of non target vectors}}{\text{number of target vectors}}$$

3.3.2 Speaker Verifier. The original speaker identifier classifies by selecting the codebook resulting in the lowest distortion measure between that codebook and a set of test vectors. Because we wish to project the classification information into a probabilistic space, we will again use a neural net classifier. We will retain the earlier stages of the system, extracting Cepstral coefficients and developing a codebook for each person, but will then present the vectors within the codebook to a neural net, tagging the vectors in the target codebook as belonging to one class, and all other codebooks' vectors as belonging to another. Similarly to what we did with the face verifier, we wish to, in effect, build a composite 'not-Joe' to train against the person to be verified. [7]

The vectors within all the training codebooks will again be used to form an orthogonal space into which to project each of the targets, and the net will be trained on the extracted KLT coefficients exactly as with the face verifier. When new instances of a person are presented, the speech will be converted into a new test codebook, and verifying the speaker's identity will be accomplished in the same manner as verifying faces, including the compensation for training bias produced by training on more 'non-Joe' vectors than 'Joe' vectors.

3.4 Fusion of Face and Speaker Verifiers

Though many options exist for fusing multiple sensor outputs, for this implementation we will only look at simple linear combinations of the post probabilities output from our two neural nets. In other words, we will attempt to minimize the classification error by choosing the appropriate combination of posteriori probabilities.

3.5 Potential Improvements to the Face Verifier

The existing face identification system relies on the KLT process to reduce the dimensionality of the feature set and to project the images into an orthogonal feature space. The number of features are further reduced by selecting the desired number of eigenvectors corresponding to the highest eigenvalues of the training set covariance matrix. With faces, previous work demonstrated that the top five or six eigen-coefficients were adequate for providing good classification performance under fairly controlled conditions, but no attempt was made to determine if we were actually selecting the dimensions with the most

[7]Note that we could train our net using the Cepstral vectors prior to clustering into codebooks, but the clustering process was designed to capture the essence of the speaker; any small loss of information is felt to be more than offset by the elimination of the overhead of processing hundreds of 'raw' vectors (versus 64 for the codebook) for each individual.

classification information. Recall that the KLT process chooses as the optimal dimensions those that have the maximum variance across the entire training population. Though the higher variance does imply a higher amount of information, the information is not necessarily most suitable for differentiation between classes. Instead, this type of information is more suited to reconstruction of some initial pattern using a reduced number of features; reconstructing with a given number of eigen-coefficients (in descending order from the maximum eigenvalue) guarantees that there is no other combination of coefficients that will result in a lower mean-squared reconstruction error (63:275).

For the classification problem, we wish to determine if there is a method of dimension selection that is more appropriate to the multi-class problem. Forming an orthogonal space based on the training population is still appropriate, so that process will be retained, as will the projection of the training patterns into that space. As an alternative to the KLT selection method, we would like to select the dimensions that correspond to the lowest amount of in-class variance, as well as the highest amount of out-of-class variance. This metric should result in retaining the dimensions that have the most information about the difference between different classes. Such a metric is the F ratio, which is defined as $\frac{\text{variance of the means (over all classes)}}{\text{mean of the variances (within classes)}}$ (39:177). Given n instances of m different classes results in the F ratio

$$F = \frac{\frac{1}{m-1}\sum_{j=1}^{m}(\mu_j - \bar{\mu})^2}{\frac{1}{m(n-1)}\sum_{j=1}^{m}\sum_{i=1}^{n}(x_{ij} - \mu_j)^2}$$

where $\bar{\mu}$ is the mean of all measurements across all classes, μ_j is the mean of all measurements for Class j, and x_{ij} is the ith measurement of Class j. We will implement this figure of merit in both of the verification systems we develop and test the performance of the systems using dimensions selected via this metric versus those selected by eigenvalue order.

Recognizing that we are not restricted to selecting the 'important' features by linear means only (both eigenvalue ordering and the F ratio provide linear selection criteria), we can also attempt to non-linearly transform our set of orthogonal dimensions into a new, reduced space. Using a sigmoidal back-prop net similar to the one we are using for verification, we will explore the possibility of increasing classification performance by training the net to maximize the distance between the classes.

In order to use a neural net for such a dimensional transformation, we must derive a rule for updating the weights. Many update methodologies exist, but we will develop one that will transform the initial multi-class data (two-class in the verification case) into a more easily separable, and thus more easily classifiable, space. The rule shall be based on maximizing the separation between elements of the different classes, and the error from which it is derived can be stated as

$$E = \sum_k \frac{1}{2}(Z_k^1 - Z_k^2)^2$$

where Z_k^l is the output from output node k when the net is presented a vector from Class l, and E is a measure of the total distance between the net outputs when presented with vectors first from Class 1, then from Class 2. Maximizing this function should maximize the distance between the members of different classes. We can use gradient ascent to accomplish our goals because we are maximizing. To derive our update rule, we shall begin by finding the partial derivatives of the output 'error' with respect to the various weights. For the weights connecting node j in the hidden layer to node k in the output layer

$$\begin{aligned}
\frac{\partial E}{\partial w_{kj}} &= \frac{\partial}{\partial w_{kj}} \sum_k \frac{1}{2}(Z_k^1 - Z_k^2)^2 \\
&= (Z_k^1 - Z_k^2)\frac{\partial}{\partial w_{kj}}(Z_k^1 - Z_k^2) \\
&= (Z_k^1 - Z_k^2)\left[Z_k^1(1 - Z_k^1)Y_j^1 - Z_k^2(1 - Z_k^2)Y_j^2\right]
\end{aligned}$$

Then the update rule can be stated:

$$w_{kj}^+ = w_{kj}^- + \eta_{out}\left(\delta_k^1 Y_j^1 - \delta_k^2 Y_j^2\right)$$

where $\delta_k^l = (Z_k^1 - Z_k^2)Z_k^l(1 - Z_k^l), l = 1,2$

The update rule for the weights connecting the *i*th input node to the *j*th hidden node can be found in a similar manner:

$$\begin{aligned}
\frac{\partial E}{\partial w_{ji}} &= \sum_k \left[(Z_k^1 - Z_k^2)\frac{\partial}{\partial w_{ji}}(Z_k^1 - Z_k^2)\right] \\
&= \sum_k \left[(Z_k^1 - Z_k^2)\frac{\partial}{\partial w_{ji}}\left(\frac{1}{1+e^{-\sum_j w_{kj}Y_j^1}} - \frac{1}{1+e^{-\sum_j w_{kj}Y_j^2}}\right)\right] \\
&= \sum_k \left[(Z_k^1 - Z_k^2)\left(-(Z_k^1)(1-Z_k^1)\frac{\partial}{\partial w_{ji}}(-w_{kj}Y_j^1) + (Z_k^2)(1-Z_k^2)\frac{\partial}{\partial w_{ji}}(-w_{kj}Y_j^2)\right)\right] \\
&= \sum_k \left[(Z_k^1 - Z_k^2)\left[(Z_k^1)(1-Z_k^1)(w_{kj})(Y_j^1)(1-Y_j^1)X_i^1 - (Z_k^2)(1-Z_k^2)(w_{kj})(Y_j^2)(1-Y_j^2)X_i^2\right]\right] \\
&= \sum_k \left[(Z_k^1 - Z_k^2)(w_{kj})\right]\left[(Z_k^1)(1-Z_k^1)(Y_j^1)(1-Y_j^1)X_i^1 - (Z_k^2)(1-Z_k^2)(Y_j^2)(1-Y_j^2)X_i^2\right]
\end{aligned}$$

This then leads to the weight update rule

$$W_{ji}^{+} = W_{ji}^{-} + \left[\sum_k \delta_j^1 X_i^1 - \sum_k \delta_j^2 X_i^2\right]$$

where $\delta_j^l = (Z_k^1 - Z_k^2)(w_{kj})(Z_k^l)(1 - Z_k^l)(Y_j^l)(1 - Y_j^l)$

Note that the derivation performed above assumes only two classes are being examined; in the case of our verification problems, this would be an appropriate solution. The error term can easily be generalized to m classes:

$$E = \sum_k \sum_m \sum_{\bar{m}} \frac{1}{2}(Z_k^m - Z_k^{\bar{m}})^2$$

where the summations over m and $\bar{m}$ will account for all the distances between the outputs of each member of one class and each member of every other class. The weight update rules can be derived in a manner similar to that shown above for two classes.

This chapter has provided an overview of the methodology to be used in the performance of this research. The details of the actual implementation and the software generated and modified are presented in Appendix A. The next chapter provides the results from the experimentation accomplished in the course of this research. Performance of the individual verifiers under different conditions will be detailed, as will the performance of the overall identity verification system.

IV. Results

4.1 Introduction

This chapter contains the results obtained during this thesis research. Because a major thrust of this work has been to determine if the dimensional reduction achieved through the traditional KLT methodology is well suited to face recognition/verification, we shall first examine the results of applying the Figure of Merit (FoM) described in Chapter 3 to a contrived 'toy' problem. In this way, we will determine the effectiveness of the metric on a problem with a known data distribution, and if it is shown to be effective we will extend its use to actual face and speaker data.

We will then examine the face verification problem, focusing again on the dimensionality reduction task, comparing clustering and accuracy obtained by selecting dimensions based on FoM, eigenvalue, and the dimensions resulting in the lowest training error for a set number of training epochs. We will perform the same comparison using speaker data, and will use the results from the face and speaker methodology comparisons to implement the two components of the identity verification system using the technique best suited to each.

Next, we will examine the performance of the nonlinear dimensional transformation method on toy and actual data. In this examination, we will attempt to determine the viability of such a metric.

We will finally look at the verification accuracy resulting from the fusion of the face and speaker verification systems. We will first determine the optimal linear combination of individual verifier cutputs that will provide the highest training accuracy, and will then use that combination to test the system against new data.

4.2 The Figure of Merit

Recall that the Figure of Merit (FoM) was developed through a straightforward process:

- Calculate the orthogonal eigenspace associated with the entire population of training samples by determineing the eigenvectors of the covariance matrix of the sample population.
- Project each individual class into the new eigenspace using simple matrix multiplication.
- Determine the means and variances of each individual class within the orthogonal space, as well as the variance of the class means and the mean of the class variances.
- Calculate the figure of merit using the following formula:

$$\text{FoM} = \frac{variance\ of\ interclass\ means}{mean\ of\ intraclass\ variances}$$

This FoM represents a measure of how separable the individual classes are within the orthogonal space in relation to how tightly clustered the elements of each class are. A higher value will indicate better separability.

To test this FoM, we developed a toy problem using a two-class, three-dimensional environment. Using MatLab ®, 50 sample points were used to generate a uniform distribution with dimensions on the $x - y - z$ axis of 5 units by 1 unit by 1 unit. A copy of this distribution was then made, and was shifted along the y and z axes so that the two classes were parallel along the x axis (Figure 4.1).

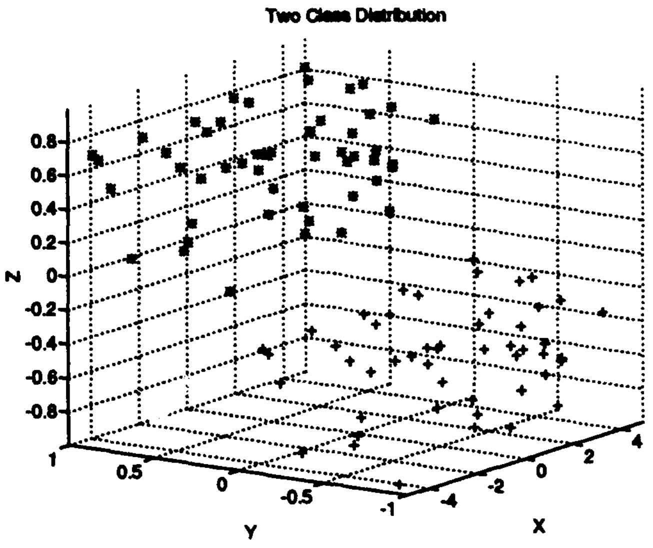

Figure 4.1 Toy problem distribution.

This type of class distribution will provide a basis for a comparison between the traditional KLT method and the FoM approach. Using the KLT, the importance of each eigenvector calculated will be directly proportional to the variance of each dimension among the entire sample population; the dimension with the highest variance, in this case along the x axis, will be the most important. However, we can easily see there is no information in the x-dimension that allows us to discriminate between the two classes, so using the KLT and one eigen-coefficient should result in poor discriminator performance. Using the FoM approach, however, more importance is placed on those dimensions having a higher class mean separation in relation to the class variances. In this case, obviously either the y- or z-dimensions offer more information than the x-dimension for class discriminability.

The training data were processed through the two different algorithms, and a back-prop neural net was trained for each with one eigen-coefficient, using half the data points from each distribution for training. The classification results using a single coefficient from the remaining data points are shown in Table 4.1. Note that, as expected, the KLT method does not allow discrimination between the classes using only one coefficient, while the FoM method does indeed allow us to classify the test data with a high probability (recall that the outputs of the neural net can be directly related to pseudo post-probabilities).

Table 4.1 Classification accuracy of KLT vs FoM.

	KLT	*FoM*
Mean Probability of Correct Classification (one coefficient)	0.22	0.72

4.3 Face Verification Dimensionality Reduction

In this section we shall examine the characteristics and importance of the different dimensions resulting from the orthogonalization process on face data.

4.3.1 Clustering in Two Dimensions. To get a feeling for how our data will cluster using the different dimensionality reductions schemes, we will examine six classes of faces composed of five prototypes each. We will project these faces into an orthogonal eigenspace, but will only retain two of the thirty possible dimensions and plot all the extracted data eigen-coefficients onto a single plot. Retaining the two bases corresponding to the dimensions producing the lowest training error resulted in the cluster plots of Figures 4.2 (clustering of the same data on which the net was trained) and 4.3 (clustering of test data not previously presented to the net) [1]. Similar plots are presented in Figures 4.4 and 4.5 for coefficients selected based on eigenvalue order, and Figures 4.6 and 4.7 for those based on our figure of merit. Figure 4.8 provides an example of an original face in the training set and the reconstruction of that face using two eigen-dimensions selected by each of our three different schemes.

[1]The minimum error dimensions were found by training the net on one dimension at a time until one was found producing the lowest error, repeating this process while retaining the first dimension, and so on.

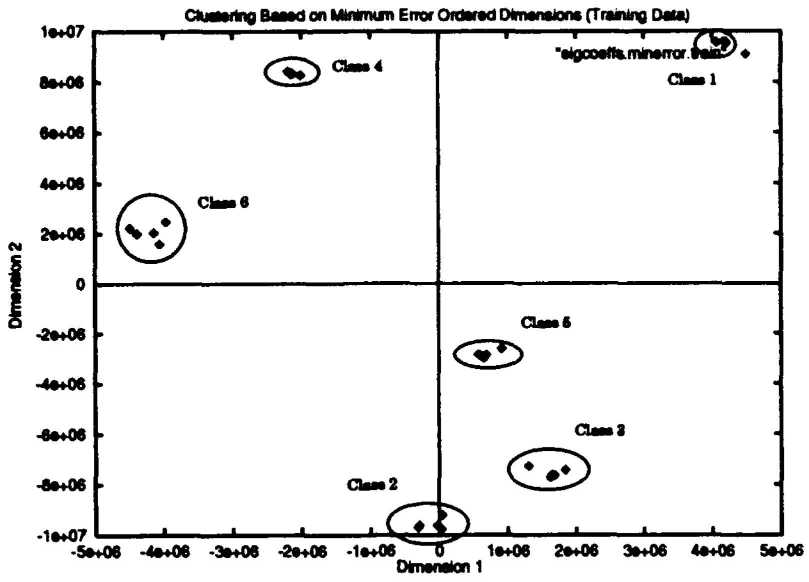

Figure 4.2 Clustering of six training classes (faces) using the eigen-dimensions producing the minimum training error.

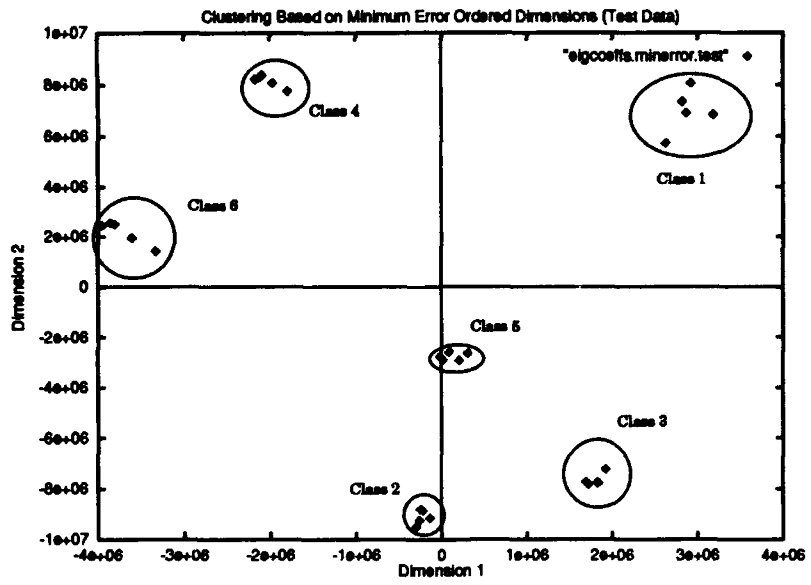

Figure 4.3 Clustering of six test classes (faces) using the eigen-dimensions producing the minimum training error.

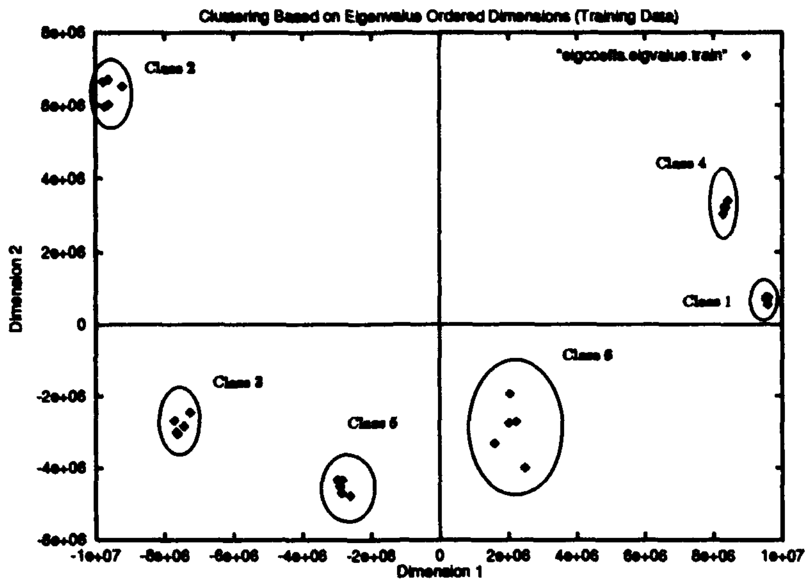

Figure 4.4 Clustering of six training classes (faces) using the eigen-dimensions corresponding to the two highest eigenvalues.

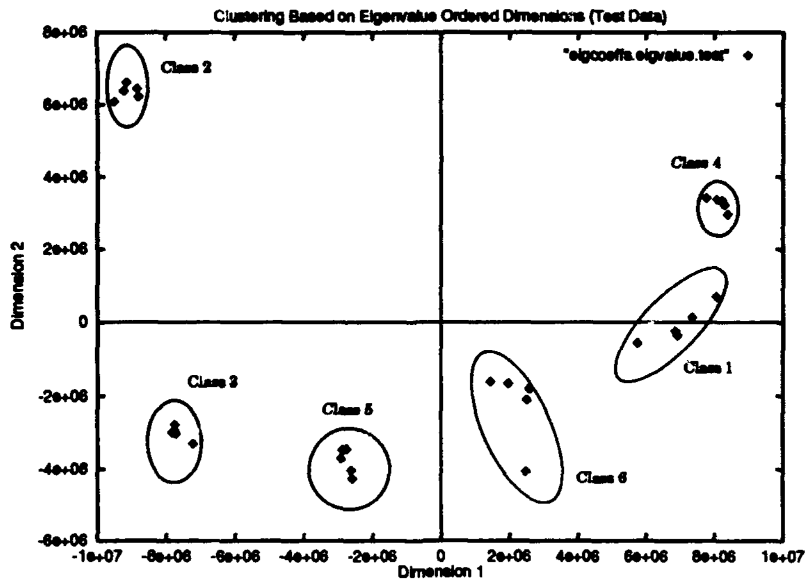

Figure 4.5 Clustering of six test classes (faces) using the eigen-dimensions corresponding to the two highest eigenvalues.

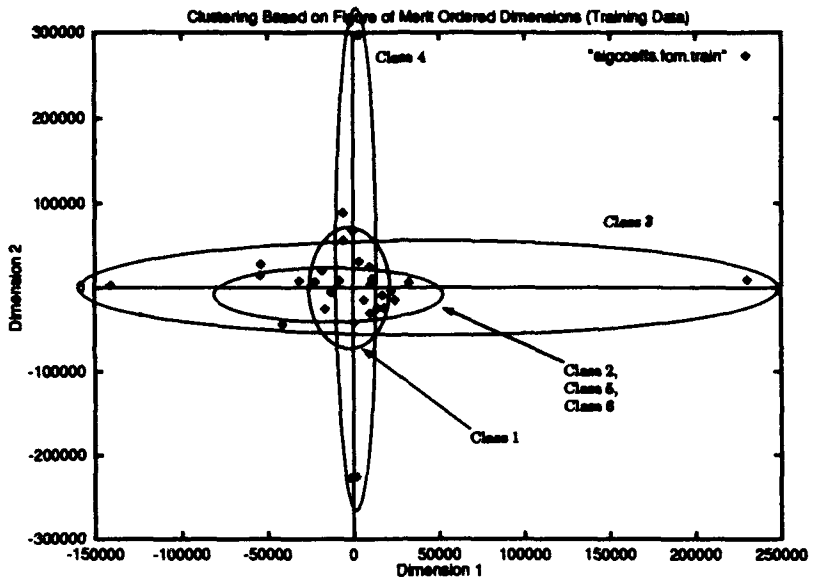

Figure 4.6 Clustering of six training classes (faces) using the eigen-dimensions corresponding to the two highest Figures of Merit.

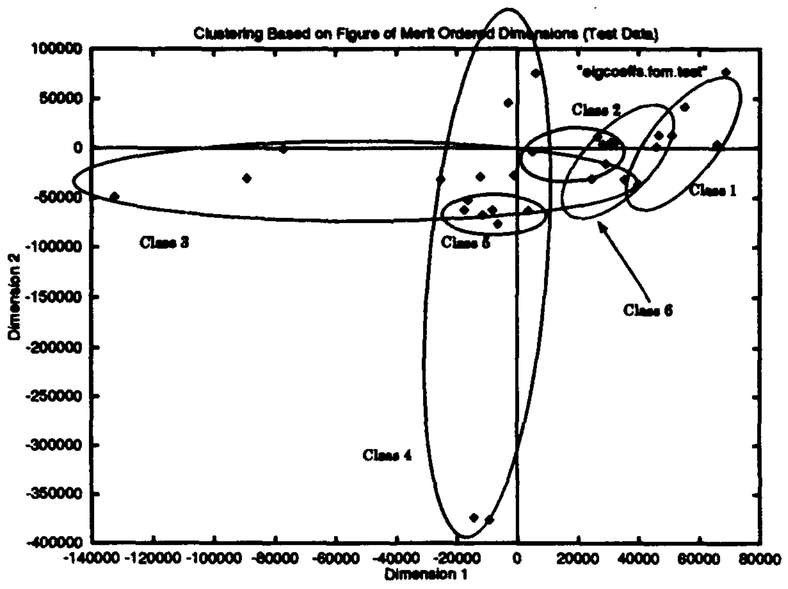

Figure 4.7 Clustering of six test classes (faces) using the eigen-dimensions corresponding to the two highest Figures of Merit.

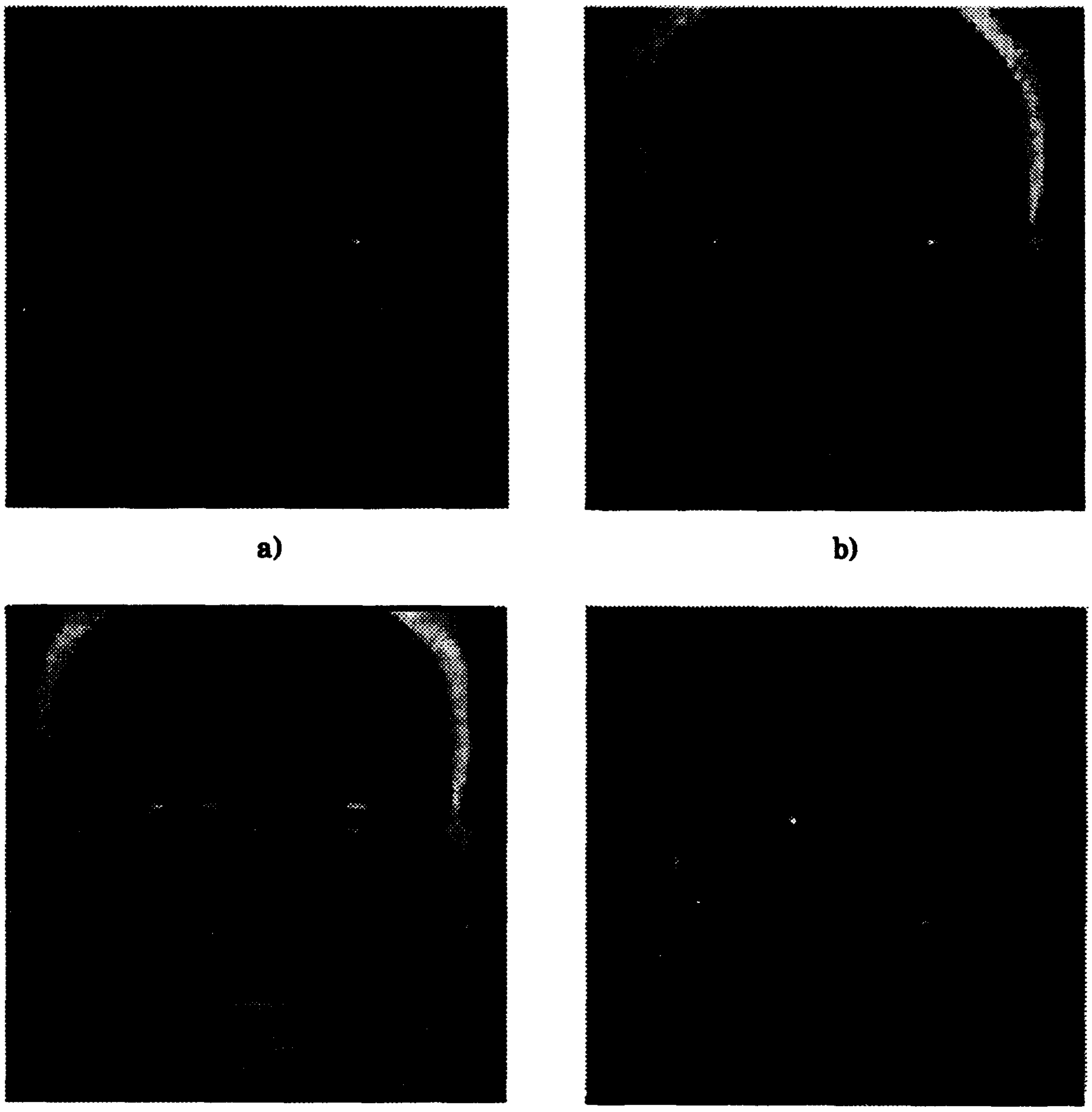

Figure 4.8 Original training image (a) reconstructed using two dimensions selected by minimum error (b), eigenvalue (c), and figure of merit (d).

Table 4.2 Face Acceptance/Rejection Accuracy

Type	*Num Dimensions*	*Min_error*	*Eigenvalue*	*FoM*
True Accept	2	100.0	100.0	80.0
True Reject	2	76.0	77.3	43.3
Overall	2	80.0	81.1	49.4
True Accept	4	100.0	100.0	83.3
True Reject	4	91.3	85.3	49.3
Overall	4	92.8	87.8	55.0
True Accept	6	100.0	100.0	76.7
True Reject	6	94.0	96.0	61.3
Overall	6	95.0	96.7	63.9
True Accept	8	100.0	100.0	93.3
True Reject	8	96.7	96.0	60.7
Overall	8	97.3	96.7	66.1

Note that for the first two selection methodologies, the six classes form quite distinct clusters for both training and testing data, implying that classification should be fairly easy. The FoM plots, however, show no such obvious separability. The distribution of the data apparently does not lend itself to separability using such a process. The implication is that selecting dimensions based on the FoM may not provide the performance desired, but because we are only looking at two dimensions out of the possible 30, the real test will be to compare classification accuracies for larger numbers of dimensions.

Remembering that one of our basic goals is to significantly reduce the number of dimensions needed to accurately represent members of each class, we will examine the results of training and testing the net with two, four, six, and eight eigen-dimensions, selecting the dimensions based on minimum training error (min_error), maximum eigenvalue, and FoM. The training and test sets will again consist of six classes, with five instances of each class in each set. Table 4.2 gives the results of these tests; the source data for these tables can be found in Appendix A. Two tables are provided here for each scheme, one giving the accuracies in accepting/rejecting actual instances of the claimed identity (True Accept) and rejecting/accepting false instances of the claimed identity (True Reject of imposters) and one giving overall verification accuracy. Figure 4.9 provides a visual summary of the overall accuracy of the face verifier using the different dimension reduction schemes. Note

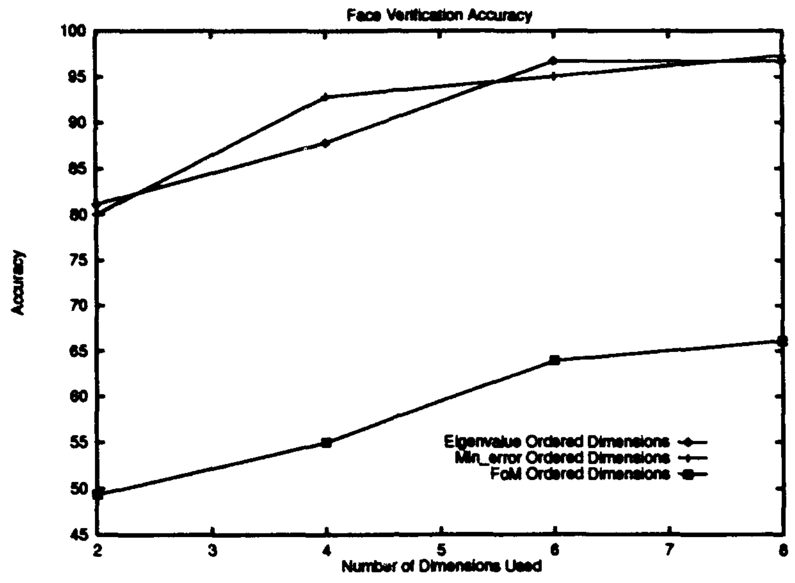

Figure 4.9 Face verification accuracy using two, four, six, and eight dimensions selected by minimum error order, eigenvalue order, and figure of merit order.

that in each of the tests the FoM dimension selection scheme provided a substantially lower verification accuracy than the minimum error or eigenvalue methods. This appears to support the impression of poor separability provided by the clustering plots of Figures 4.6 and 4.7. Figure 4.9 indicates that when using six or eight dimensions, we will attain virtually the same verification accuracy using dimensions selected by eigenvalue as we would if we selected them based on minimum error. Because the KLT process naturally produces dimensions ordered by eigenvalue, we will retain this method for selecting our dimensions for face verification. Because six dimensions provide an adequate level of accuracy (96.0 %), we will use that number for testing on the face verifier portion of the identity verification system.

4.4 Speaker Verification Dimensionality Reduction

In this section we shall perform a similar examination of speaker data dimensionality as we did with face data, including analysing the clustering performance of the data with dimensions selected through our three different criteria. We shall also examine the verification performance of the speaker verifier using these different dimension selection schemes.

4.4.1 Clustering with Two Dimensions. Because our speaker verifier is based on training and testing on 64 code-vector long codebooks, the clustering plots will necessarily contain many more data points than the face data plots did; with the six classes we will use, there will be a total of 384 points on the plot. We would still hope, however, to see some distinct clustering of these points by class, and Figures 4.10 through 4.15 show that clustering behavior of some of the classes is evident, but there is substantial overlap between the classes.

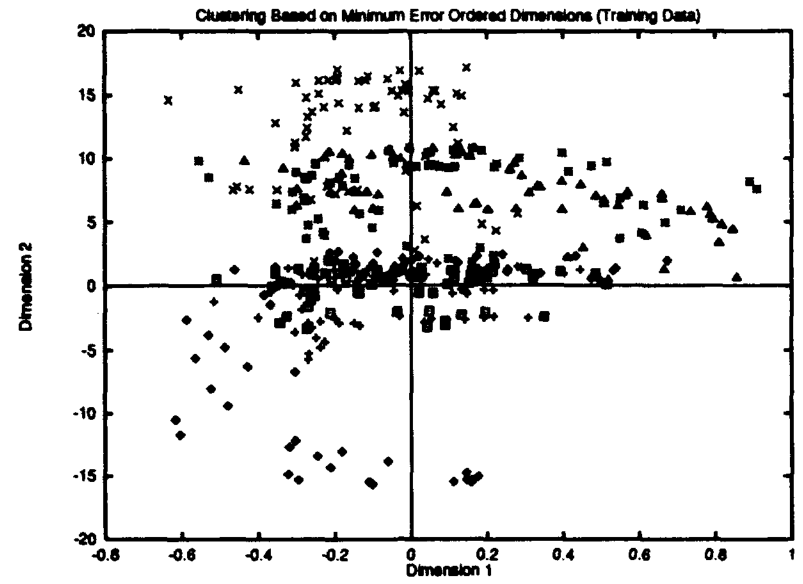

Figure 4.10 Clustering of six training classes (speakers) using the eigen-dimensions producing the minimum training error.

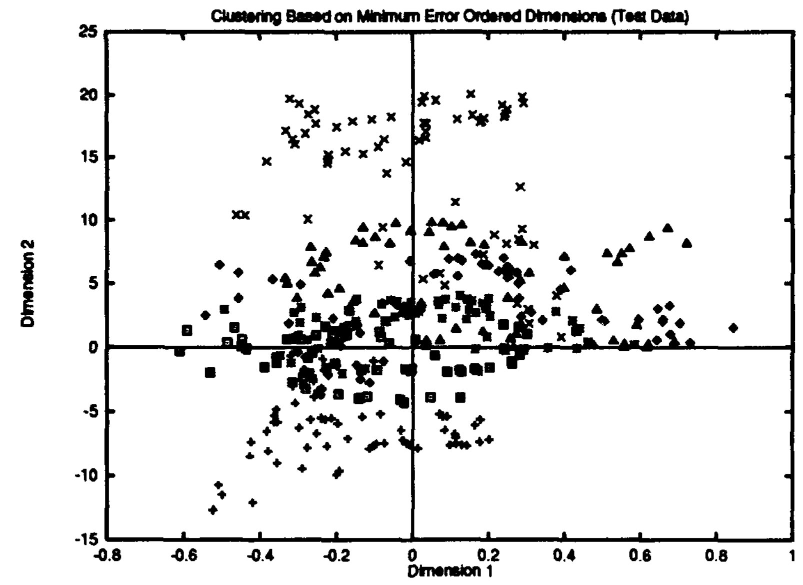

Figure 4.11 Clustering of six test classes (speakers) using the eigen-dimensions producing the minimum training error.

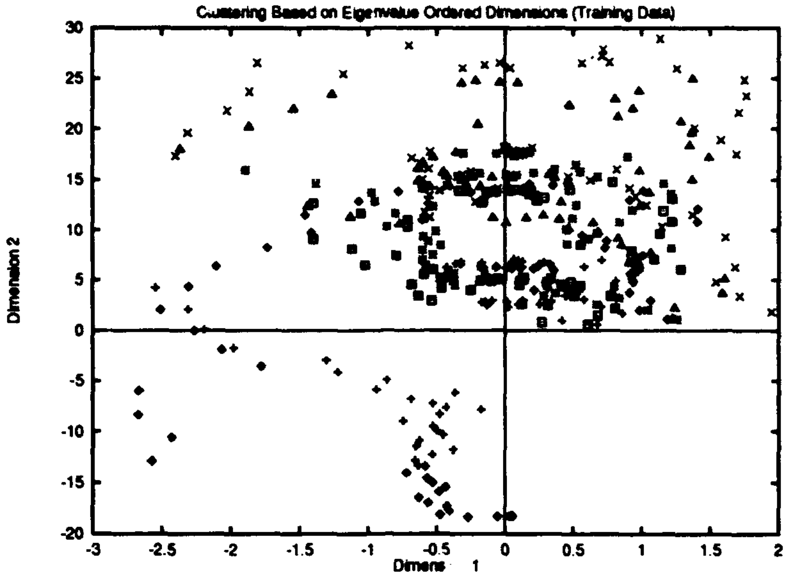

Figure 4.12 Clustering of six training classes (speakers) using the eigen-dimensions corresponding to the two highest eigenvalues.

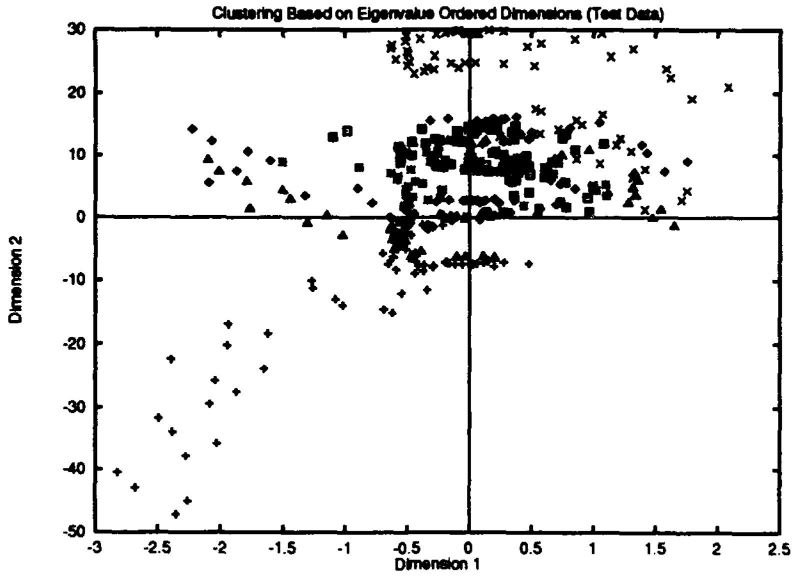

Figure 4.13 Clustering of six test classes (speakers) using the eigen-dimensions corresponding to the two highest eigenvalues.

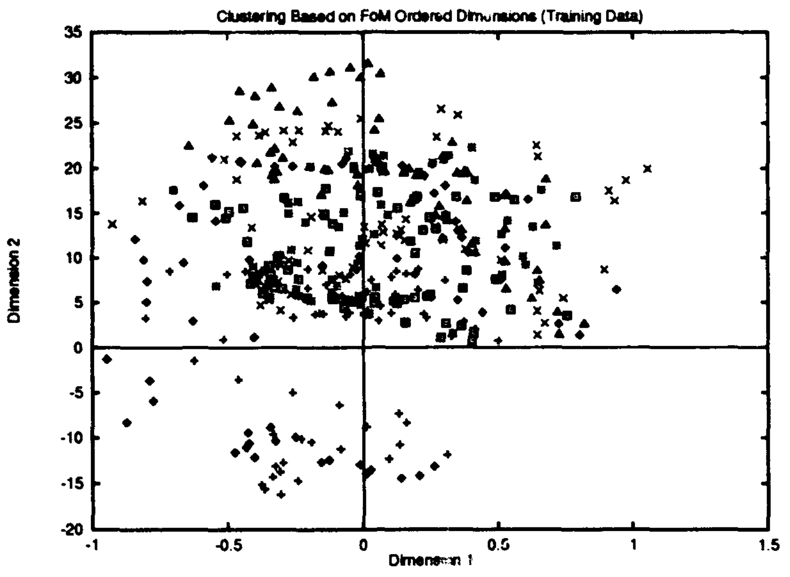

Figure 4.14 Clustering of six training classes (speakers) using the eigen-dimensions corresponding to the two highest Figures of Merit.

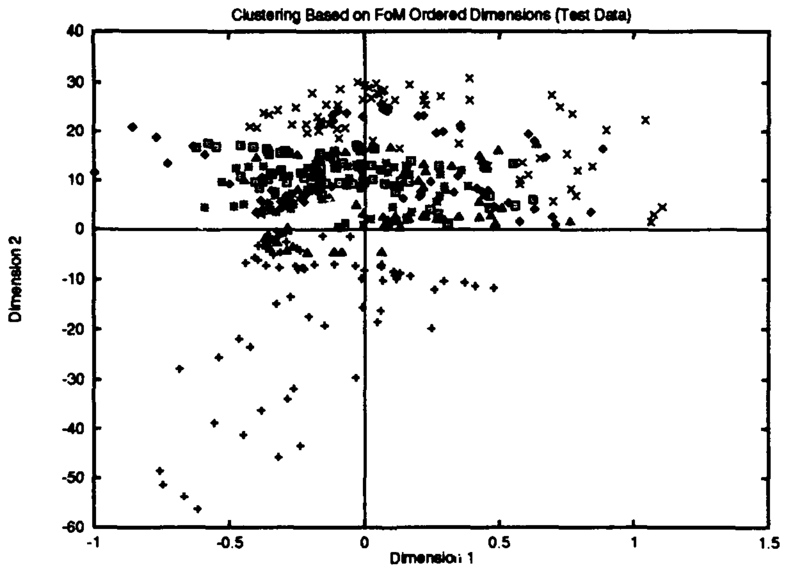

Figure 4.15 Clustering of six test classes (speakers) using the eigen-dimensions corresponding to the two highest Figures of Merit.

There does not appear to be a significant difference between any of the dimension selection methods, so we next performed an analysis of verification performance using

Table 4.3 Speaker Acceptance/Rejection Accuracy

Type	*Num Dimensions*	*Min_error*	*Eigenvalue*	*FoM*
True Accept	4	38.9	52.8	58.3
True Reject	4	58.0	58.3	60.1
Overall	2	55.9	57.7	59.9
True Accept	6	77.8	38.9	61.1
True Reject	6	61.5	54.9	58.0
Overall	4	63.3	53.1	58.3
True Accept	8	77.8	58.3	66.7
True Reject	8	59.0	62.9	52.1
Overall	6	61.1	62.4	53.7
True Accept	10	88.9	47.2	72.2
True Reject	10	69.8	62.2	53.1
Overall	8	71.9	60.5	55.2

four, six, eight, and ten dimensions selected by the three different schemes, minimum error, eigenvalue, and FoM based. Table 4.3 provides the results of these tests, in a format identical to that used when presenting the face verification results. As with the face verifier testing, source data for these tables are provided in Appendix A. With speaker data, as with the face data, using eigenvalue based dimension selection provides superior performance over using the figure of merit, so for the speaker verifier portion of our identity verifier this method shall be used. Because we wish to reduce the dimensionality of the problem, we will select the top ten dimensions. Figure 4.16 provides a visual representation of the verification accuracy for the different number of dimensions.

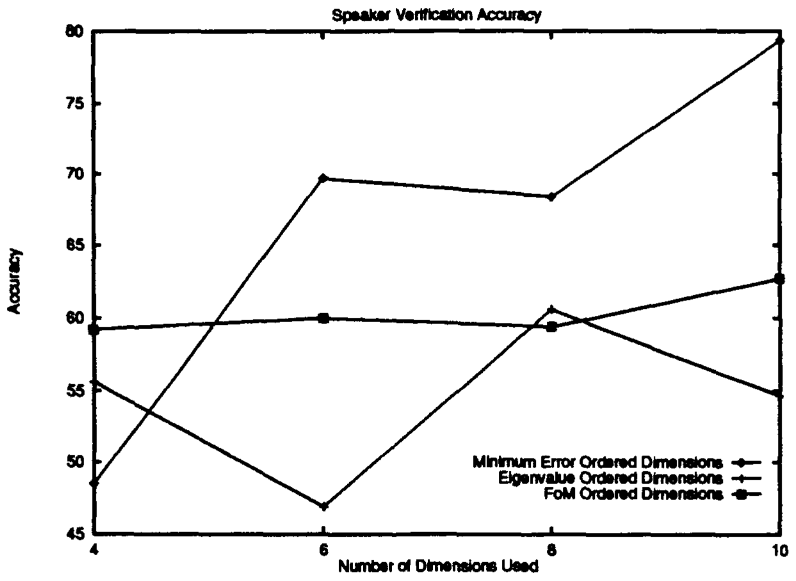

Figure 4.16 Speaker verification accuracy using four, six, eight, and ten dimensions selected by minimum error order, eigenvalue order, and figure of merit order.

4.5 Nonlinear Feature Transformation

For this test, we used the backprop neural net and weight update rule as described in Chapter 3. The system was first tested against a set of linearly separable, two-class toy data, as shown in Figure 4.17. Rather than attempting to reduce the dimensionality of the test space (from two to one in this case), we first examined the case of simply transforming the original input dimensions into the same number of output dimensions. This could provide some idea of if and how the data would cluster in a transformed space; similar cluster plots were shown earlier in the chapter for face and speaker data. Figure 4.18 shows the result of transforming the original data through the net after 1000 iterations (50 epochs).

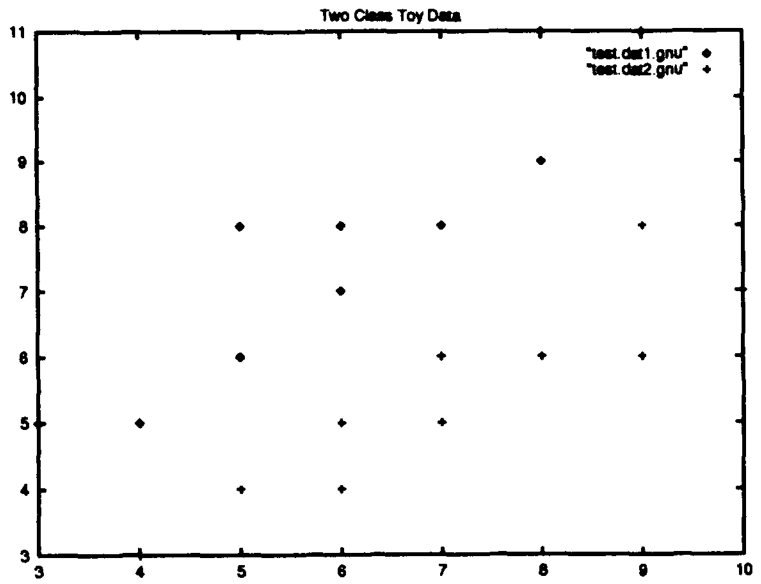

Figure 4.17 Toy data set used for testing against the nonlinear transformation net.

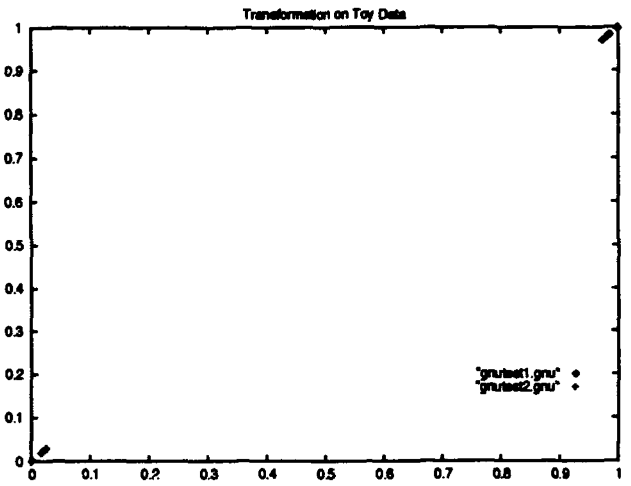

Figure 4.18 Transformed toy data.

Note that the data clustered quite well, leading us to believe that this methodology holds promise; the net was able to transform the data points into a more easily classifiable space. Actual face data was next projected into an eigenspace, 30 coefficients were extracted from each of two classes of faces consisting of five protoypes apiece. Figures 4.19 through 4.23 show the movement of the data points as the number of training iterations increases.

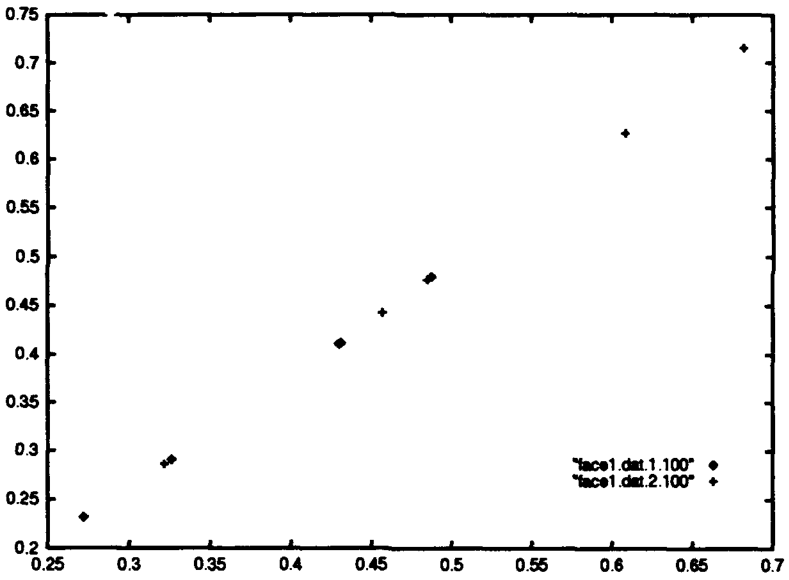

Figure 4.19 Transformed face data (100 iterations).

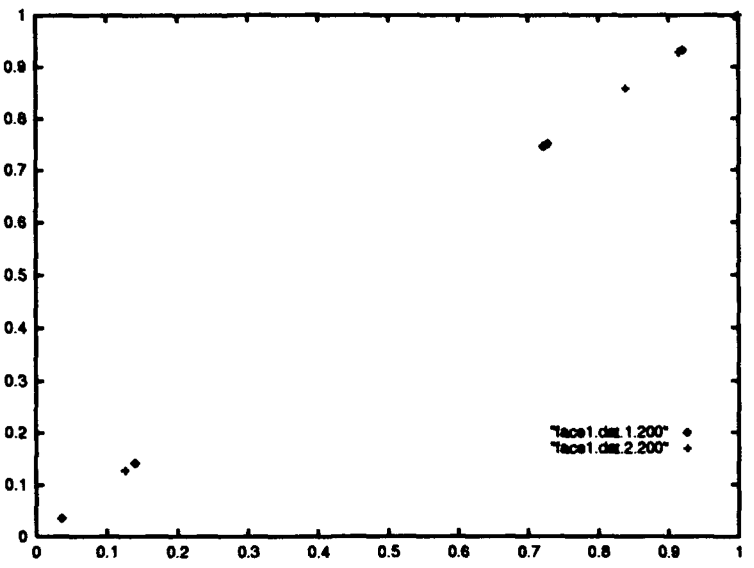

Figure 4.20 Transformed face data (200 iterations).

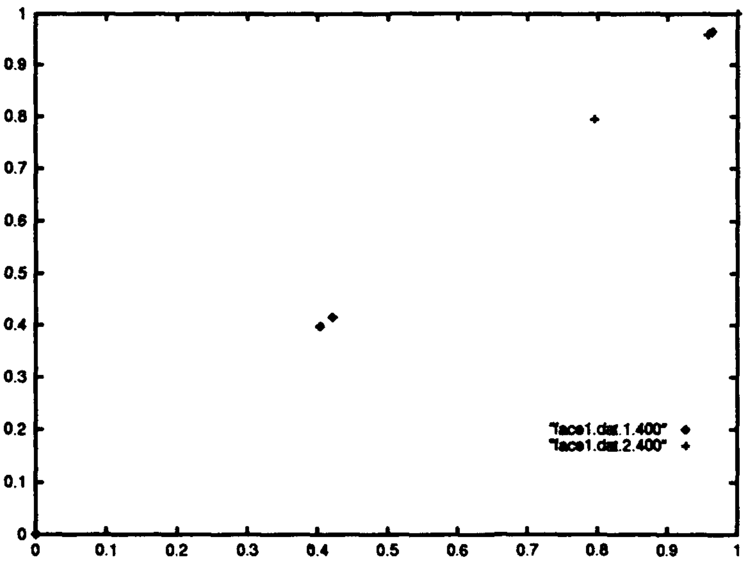

Figure 4.21 Transformed face data (400 iterations).

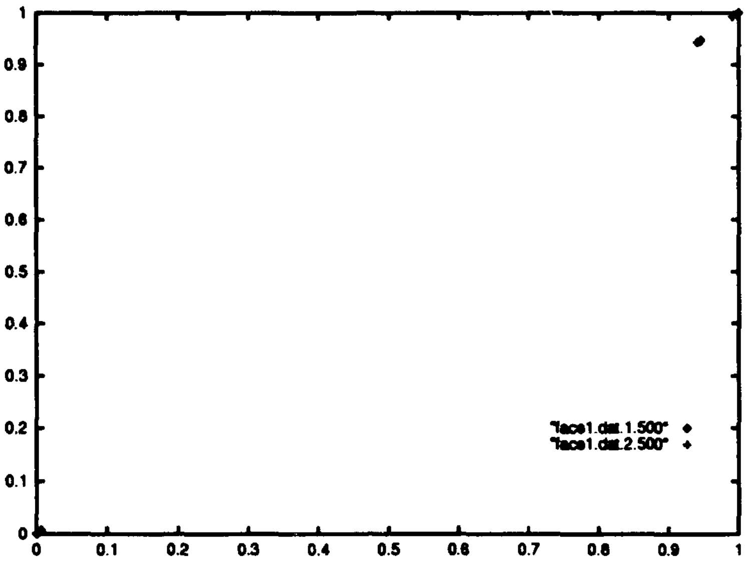

Figure 4.22 Transformed face data (500 iterations).

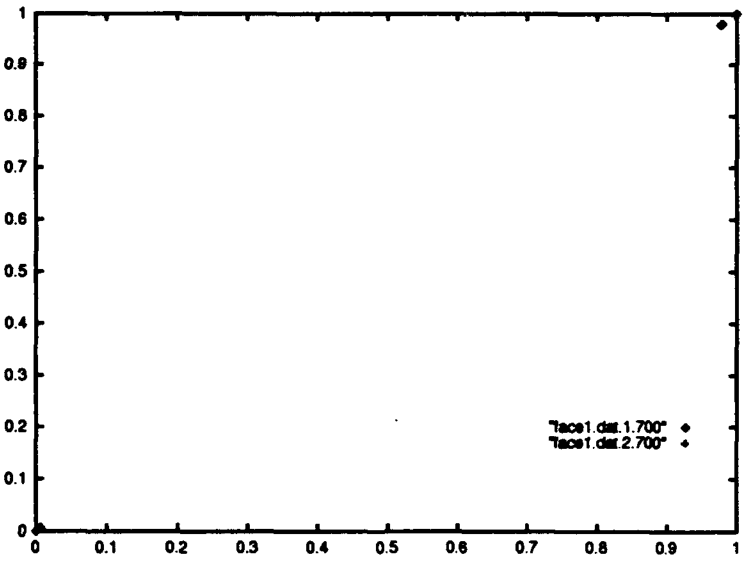

Figure 4.23 Transformed face data (700 iterations).

There is a definite movement of cluster centers toward the far corners of the reduced, two-dimensional space, showing that, as with the toy data, distance maximization is taking place. Note, however, that each of the clusters has four members of one class and one member of the other class in it, or an 80% clustering accuracy. The net appears to be finding a solution to the transformation problem that may not be optimal; it may be getting caught in a local minima. Recall that the neural net is not guaranteed to find the best solution (for that matter, it is not guaranteed to find any solution), but may find some local solution that is 'almost' correct, as was the case here. Nevertheless, the technique still appears to have done a good job of reducing 30-dimensional data to only two dimensions while still providing fair separability. Using the same data set, we also trained the net for 1,000,000 iterations (100,000 epochs), and found that one of the abberrant points had been pushed back across to its own class cluster, leaving one cluster with members of one class, and the other with all the members of one class and one member from the other, a clustering accuracy of 90%. This seems to show that given enough time and/or different initial parameterization the net may indeed be able to better separate the data in reduced dimensions.

We next looked at a five class problem, using five different classes of face eigen-coefficients, again with five prototypes of each class, and transformed the data points into a two-dimensional space. Figure 4.24 shows the result of 5,000 iterations (200 epochs) of the data through the net. The hope of distinct clustering does not materialize for this data set, though some clustering has occurred. There is a great deal of class overlap, though, so classification would likely not be an easy task.

Further training on this set collapsed the data to only two cluster points a very small distance apart, making the transformed space virtually unclassifiable. With further 'tweaking', the performance of the transformation net on multi-class (greater than two) problems could very likely be improved, but for the purposes of this research such tweaking was not attempted. In the next chapter, we make recommendations for areas of future research on and testing of this nonlinear dimensional reduction scheme.

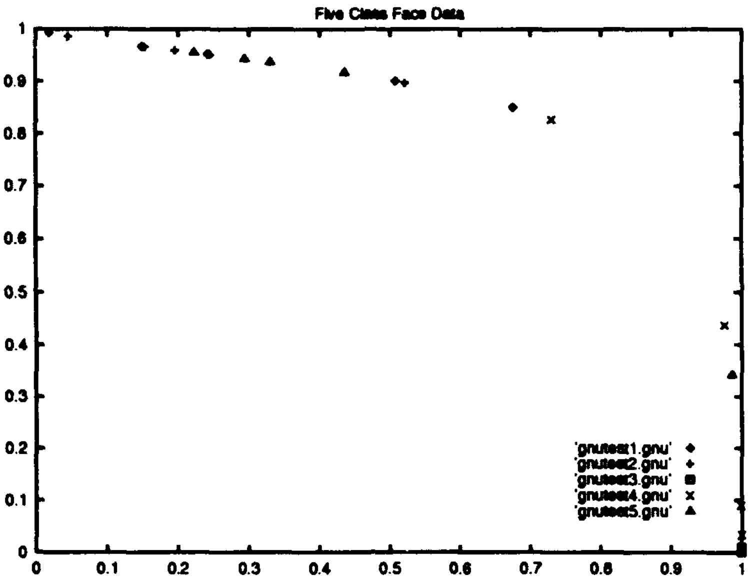

Figure 4.24 Transformed Five Class Problem.

4.6 Fusion of the Face and Speaker Verifiers

One traditional scheme for sensor fusion has been to fuse the post-probabilities from the outputs of the individual sensors, and we first examined this method for our fused face/speaker verification system. Using the four common classes between the face and speaker data described earlier in this chapter, testing was performed to find the linear combination of net output face/speaker probabilities that provided the highest verification probability. As can be easily seen by the individual class accuracies provided in the tables in Appendix B, the accuracy of the face verifier appears to overpower that of the speaker verifier. Recall, though, that the verification scheme has been based on a simple 0.5 threshold; that is, if the probability calculated from the output of the net is greater than 0.5, the verifier is said to have accepted the individual as who he claims to be. Conversely, a post-probability of less than 0.5 will indicate non-acceptance of the individual. We found that the levels of probability out of the face verifier were so high (for acceptance) and so low (for rejection) that the face probability still greatly overpowered the speaker probability, the levels of which did not stray to the extremes nearly as often or as significantly. Based strictly on the probability levels, we would select 100% face verification and 0% speaker verification for the 'fused' verifier [2].

What is more important than simply the levels of the probability output, however, is the accuracy of verification. For this system, a verification probability of 0.9 provides the same correct (or incorrect) answer as a probability of 0.6. Therefore, the decision was made to track the verification accuracy of the system as a function of the linear combinations of the outputs of the two verifiers. Table 4.4 gives the result of this test, showing that the highest fused verification accuracy was achieved when using 40% face and 60% speaker probabilities (recall that True Accept shows the accuracy in accepting actual instances of the individual, and True Reject shows accuracy in rejecting imposters). This may be somewhat counter-intuitive, but is due to both the smaller variation in the levels of speaker verifier output probability and the different type of information examined by the individual verifiers; a poor performance by one verifier can be compensated for by good performance by the other.

Table 4.4 Accuracy of fused test classes.

Acceptance/rejection accuracy				
% Face	**% Speaker**	**True Accept**	**True Reject**	**Overall**
10	90	69	72.9	72
20	80	94	77.1	81
30	70	100	87.5	91
40	60	100	89.6	92
50	50	100	81.3	86
60	40	100	83.3	88
70	30	100	79.2	84
80	20	100	79.2	84
90	10	100	81.3	86

[2]This may not be strictly true. We would actually perform a statistical analysis of the data, and determine if the speaker probabilities could narrow some confidence interval within which we could expect some restrictive percentage of correct verification answers to lie.

To test the fused identity verification system, we selected nine individuals from whom to collect both face and speaker data. Recall that for the fused system it was determined that six-dimensional face data and ten-dimensional speaker data (both sets of dimensions selected by descending eigenvalue order), and a fusion ratio of 40 % face and 60 % speaker would be used. Table 4.5 shows the accuracy of each separate verifier taken alone and the overall fused accuracy. The most important thing to notice about this table is that the performance of the fused identity verifier is indeed improved over that of either of the two components.

Table 4.5 Fused Identity Verification Accuracy.

Face Verifier	**Speaker Verifier**	**Identity Verifier**
55 %	61 %	66 %

It should also be mentioned that the levels of face verification accuracy found during this test appears to be substantially lower than was found during the tests earlier in this chapter. This is due to the increased number of classes used during this portion of the testing; we used a total of 50 % more prototypes for this test (nine, vice six for the earlier tests). Additionally, an effort was made to purposely select individuals without radically different appearances, creating a somewhat more difficult classification problem. The goal was to see if sub-optimal performance of the face verifier could be overcome by input from the speaker verifier, and the results appear to indicate this is the case.

To provide a 'sanity check' that the ratio of face to speaker probabilities selected was appropriate, we also ran a test on the nine classes of test data, with the results shown in Table 4.6. Notice that if the decision had been based on this data, we would have selected the 50 %/50 % mix, but the 40 %/60 % chosen also performed quite well.

Table 4.6 Accuracy of fused test classes.

Acceptance/rejection accuracy				
% Face	**% Speaker**	**True Accept**	**True Reject**	**Overall**
10	90	53	63	62
20	80	67	62	63
30	70	72	63	64
40	60	83	64	66
50	50	97	64	68
60	40	100	61	65
70	30	100	59	64
80	20	100	54	59
90	10	100	52	57

In the next chapter we will summarize the conclusions resulting from the testing described in this chapter. We shall also present a summary of areas recommended for future research.

V. Conclusion

5.1 Introduction

This chapter will provide a short summary of the purpose and results of this research effort, and will suggest areas suitable for future research. Recall that our overall goals were threefold:

- To convert existing face recognition and speaker identification systems to face and speaker verification systems.
- To determine whether the method of dimensionality reduction currently used for the face recognition system was suitable for the task.
- To fuse the face and speaker verifiers in the hope of providing enhanced performance over either of the individual verifiers alone.

These items will be addressed in the following sections.

5.2 Conversion From Recognition to Verification

5.2.1 Face Verifier. The existing neural-net based face recognition system was converted into a verifier by altering the training methodology. Instead of training the net to recognize all the individuals in the training data base, a separate net was trained for each person. The feature set used consisted of the KLT coefficients extracted from the original 64×64 pixel images. This net was trained to recognize two classes: the person for whom the net was trained and everyone else (Joe and not-Joe). Training was accomplished by presenting prototypes of the individual as instances of one class, and all other prototypes in the training database as instances of the other class. With such a scheme we hoped to train the net to 'learn' what makes the individual different from an average person, where the average person is represented by all the other people in the training set.

This system worked well, providing an overall verification accuracy of greater than 96% for a six class problem using six features for each prototype. The system provided 100% true acceptance accuracy (correctly identifying new instances of the individual) and

96% true rejection accuracy (correctly rejecting instances of people other than the individual).

5.2.2 Speaker Verifier. The original speaker identifier was based on Cepstral extraction, and classified using a distortion metric in a 20-dimensional Cepstral space. To increase the commonality between the face and speaker verifiers, we chose to convert this system to a neural net based verifier, using the extracted Cepstral coefficients as the base features for each speaker. As with the face verifier, these features were projected into KLT space and KLT coefficients were extracted for presentation to the net. The training scheme was identical to that used for the face verifier, training a net for each individual in the training database.

The system showed substantially lower accuracy than the face verifier when tested on a nine class, ten-dimensional problem. Overall accuracy was 60.5%, derived from 47% true acceptance accuracy and 62% true reject accuracy. This decrease in accuracy is not unexpected, and is due, we feel, primarily to one specific cause. All of the data (face and speaker) was obtained on a single day. With face data, it was a relatively straightforward process to ensure consistent positioning and scale of the face within the image using manual segmentation. Capturing the speaker data, however, was a much less consistent process, and it has been shown that more samples of speech would be necessary to form an 'accurate' representation of an individual(9). Nonetheless, the goal was not necessarily to provide some specific level of speaker verification accuracy, but instead to develop a baseline with which to compare the performance of the fused identity verification system.

5.3 Dimensionality Reduction

The method used to select the number of features to present to the neural net was based on KLT analysis for the base verification systems. We questioned whether selecting the dimensions based on the eigenvalues of the covariance matrix of the original data was a suitable metric. As a baseline, we trained nets for each of the individuals and determined the eigen-dimensions to select which would result in the minimum training error. We then trained each net using specific numbers of eigenvalue ordered dimensions (the KLT method), and also the same number of dimensions ordered by a new figure of merit based on the ratio of inter-class means to intra-class variance. It was found that the eigenvalue ordered dimension selection performed nearly as well for faces as if the dimensions had been selected based on minimum training error, and substantially better than the figure of merit method. With speakers, the eigenvalue method worked somewhat better than the figure of merit method, but provided poorer performance than the baseline, minimum training error method. Therefore it was decided that the fused verification system would use the KLT dimensional selection method for both the face and speaker verifiers.

We also explored a method of dimension reduction using a nonlinear, transformation neural net, hoping to determine a new, reduced set of dimensions that would provide better separability than does the KLT or figure of merit methodology. Using a standard, backprop net, we first trained on two classes of data, using for weight update a rule based on 'pushing' elements of differing classes apart in the transformed space. When transforming two-class, linearly separable, two-dimensional toy data into a new two-dimensional space, the net clustered the data points to two points at opposite corners of the new space, the desired effect. When two classes (five prototypes each) of actual 30-dimensional face data were presented to the net, we again saw the same, distinct two-point clustering take place, but each cluster contained four data points from one class and one from the other. We also presented five classes of face data to the net, and found that though some clustering did take place, there was a great deal of class overlap in the transformed space. Preliminary classification testing using these transformed dimensions showed significantly poorer performance than when eigen-dimensions based on either KLT or our figure of merit were selected. We did not pursue this methodology any further.

5.4 Face/Speaker Fusion

As described above the identity verification system was based on extraction of six KLT face coefficients and ten KLT speaker coefficients presented to nets specifically trained for each individual in the training set. Using training data, the fused system was tested to determine the optimal ratio of face net output to speaker net output to provide maximum verification accuracy, and it was found that a ratio of 40% face to 60% voice provided the best performance. With individual verification systems attaining only 55% (face) and 61% (speaker) accuracy, the fused system was able to attain 66% overall accuracy using this ratio. This shows that the fusion did indeed increase the performance of the identity verification system over either of the base verifiers.

5.5 Future Research

We feel there are are three primary areas that warrant further research. The first involves segmentation of faces from images and the second exploring the effect of using multiple days training and test data with the identity verifier. Finally, the nonlinear transformation net has shown substantial promise, and we feel with appropriate 'tweaking' it could serve a useful role in the dimension selection arena.

5.5.1 Face Segmentation. We have made the assumption in this research that methods can be found to segment the face from an image and position, scale, and rotate it into a desired, standard position. One promising method of finding the face has been explored in a collateral thesis by McCrae and involves color segmentation (36). She has trained a backprop net to recognize face color and has been able to successfully extract the face from an image. Additionally, she has been able to use the same technique to locate the eyes within the face. Once the positions of the eyes are known, the imag ,an be scaled, shifted, and rotated until the eye points overlay some specific locations. -istent for every individual.

Another method showing promise for segmentation uses three dimensional, temporal wavelets, introduced by Burns in his Doctoral dissertation (6). These wavelets can be used to detect the movement of faces within images over time, and have the additional advantage of removing stationary backgrounds from the images. This could make for more robust classification/verification over systems requiring constant image backgrounds (over the entire training and test sets) for good performance.

5.5.2 Multiple Day Problem. The multiple day face classification problem has not been extensively addressed in the literature, though Colombi, Krepp, et al have studied it at some length here at AFIT (9, 27). The basic problem is that though a face recognizer may function quite well when using a single day's data, when presented with data from other days the performance may be severely degraded. The fusion of face and speaker classification technologies may help to solve this problem by providing additional information to the decision-making process. We have shown that the fusion of face and speaker verification can provide better performance for a single day's data than either verifier alone, and believe that this performance increase will also be seen when the data set is extended to include data from some larger, but still small, number of days.

To extend this concept, fusing other biometric classification systems with the identity verifier developed for this effort may provide even better performance. Work is being done in concurrent theses on fingerprint identification and written word recognition, and we believe these techniques can be adapted to synergistically combine with our verifier (32, 58).

5.5.3 Nonlinear Dimensionality Transformation. Though the initial testing with classification using the nonlinear dimensional transformation net did not show any increased, multi-class classification capability, there were some interesting results which lead us to believe the method holds promise. The clustering tendencies exhibited show that it may be possible to push class data points to different points in the transformed space, given the proper weight update rule. We used a very simple rule based on pushing apart members of opposite classes; this rule could be altered in several different ways. It could be implemented in such a way that members of like classes could be attracted to each other, while members of different classes would still be repelled. The net could also be implemented with either of the above rules, but in a batch style, with all training vectors presented to the net before updating the weights. An alternate update rule could be derived based on a similar scheme to the figure of merit we developed for this thesis, maximizing the variance of means between different classes and minimizing the intra-class means. Other transformation methodologies within the net could also be explored, such as adding additional hidden layers or using linear activations on the output nodes of the net.

5.6 Conclusion

In the course of this research, we have found the answer to providing an identity verification capability to be the fusion of face and speaker verification systems. Each of these systems are based on KLT dimensionality orthogonalization and reduction, followed by back propagation neural net classification. We determined that using the KLT provided superior performance to using either a nonlinear transformation net or a figure of merit based on the F-ratio.

We also found that for verification purposes, a collection of training prototypes of different individuals can be used to represent a single class, an 'average' person. Using this concept, a neural net can be trained to recognize the difference between an individual and this average person. A net must be trained for each individual based on this two-class idea, and when anyone presents themself claiming to be a specific person, the net trained for that individual must be used to test the claim.

Appendix A. Software Development

A.1 Introduction

In this Appendix we shall examine the software modified or developed in support of this research. All code was written in the ANSI Standard C programming language; though implemented on a Sun SPARCstation 2, the great majority of it should be portable between platforms. All the code developed for this effort is included at the end of this appendix for reference. Much of the code is based on software developed during previous AFIT thesis efforts, and such information will be provided in the headers of the code listings.

We will look first at the code written for the face verification portion of the system, and then at the code written for the speaker verifier. We will next examine some of the techniques developed to implement modifications to the individual verification systems, and will finally look at the code used to tie together the two separate verifiers into a single identity verification system.

A.2 Implementing the Face Verifier

The face verification system was designed to operate in three distinct stages:

1. Acquisition of images for training and/or testing.
2. Training of the multi-layer perceptron for each user.
3. Testing of new instances of a user claiming to be a member of the training database.

For the first stage, the program **acquire** was developed to capture images from a video camera connected to a VideoPix frame grabber installed in the workstation. The training of the neural net is accomplished by **train_net**, and **verify_face_net** permits testing of the face verifier. Each of these programs are described in greater detail in the following sections.

A.2.1 acquire. This program was designed to allow easy acquisition of face images for either training or testing the verification system. If beginning a new acquisition session, the program will first build training and/or testing sub-directories in which to place the captured images; when adding images to an existing database, the new captures will be placed into the previously created sub-directories.

The software controls the frame-grabbing capabilities of the VideoPix video capture board, which is connected to the output of a video camera, and the user is allowed to capture images at an arbitrary rate and time (limited by the hardware to four frames per second). The captured image is presented to the user for examination, and if satisfactory is saved to the appropriate sub-directory. This process continues until the desired number of images have been captured.

A.2.2 train_net. This code allows training on each of the target images in the image database and produces weight files for each target. It first uses the entire population of training images to calculate the orthogonal eigenspace into which to project the faces, and then saves the appropriate number of bases (where that number is user-determined) as eigenface files. The eigenfaces are then used to extract KLT coefficients from each training file, and these coefficients are written to a single data file for presentation to the neural net.

Recall that because this is a verification net, there are only two outputs, verified or not verified. For each target on which the net is to be trained we augment all the target's coefficients with the desired node outputs of 0.9 and 0.1, and all other coefficients with 0.1 and 0.9. The net is then trained on this data to the desired levels of accuracy and error, and the weights calculated for the target are saved to a separate file. The process is continued until a net has been trained for all targets represented in the training database and weight files have been produced for each.

A.2.3 verify_face_net. Performing verification with the trained net is accomplished using this program, where a test image may already exist in a file or may be captured live for presentation to the net. One of the command line arguments is the 'claimed' identity of the image to be verified, and once the program is invoked it will build a net based on the weights calculated by **train_net.c** for that particular individual. After the test image is presented, the outputs of the net are adjusted for bias as explained in Chapter 3, and the pseudo post-probability that the test image is an instance of the claimed individual is calculated.

A.2.4 mlp_fuse. This code is the actual back-prop neural net used for classification by both the training and verifying programs. A complete description and source code listing can be found in the 1991 thesis by Krepp (27).

A.3 Implementing the Speaker Verifier

The speaker verifier is implemented quite similarly to the face verifier, a fact which allowed substantial code re-utilization. Again, a three stage process was used:

1. Acquisition of speaker data for training and/or testing.
2. Training of the multi-layer perceptron for each user.
3. Testing of new instances of a user claiming to be a member of the training database.

The major differences in the two verifiers lie in the data acquisition phase. For capturing and pre-processing speaker data, the ESPS© software package was used extensively, including the routines **mu2eps**, **filter**, **formant**, **refcof**, **select**, and **vqdes**. A windowed menu environment, SIDtool (Speaker Identification Tool) was developed to implement a series of shell script files, which in turn called the routines listed above. After all the speech was captured, **train_net** was again used to train a net for each target invididual, and finally **verify_voice_net** was used to test the verification performance of the system.

A.3.1 mu2eps. This routine converts a standard Sun audio file captured captured with the built-in SPARCstation audio equipment into a sampled format usable by the other ESPS routines.

A.3.2 filter. An FIR (finite impulse response) pre-emphasis filter is applied to the audio file by **filter**.

A.3.3 formant. **formant** applies a probability-of-voicing label to each frame of speech.

A.3.4 refcof, spectrans. These programs will generate 20 Cepstral coefficients for each frame of speech.

A.3.5 select. Using this routine, the frames containing voiced speech (determined by the probability-of-voicing label) are segmented from the utterance.

A.3.6 vqdes. **vqdes** takes the raw Cepstral data and builds a codebook for each individual consisting of the 64 codewords resulting from the LBG clustering process. These codebooks are used to train and test the nets.

A.3.7 train_net. This program functions as it did for training on images, projecting the speaker data into KLT space and training on the coefficients for each target. Weight files are also produced, corresponding to the nets built and trained for each individual.

A.3.8 verify_voice_net. As with **verify_face_net**, speaker data is presented to the appropriate net for verification and the pseudo post-probability of verification is computed.

A.4 Implementing Verification Modifications

To compare the performance of the verification nets using eigen-bases selected by different criteria, capability was built into **train_net** to allow selection of the eigenvectors based on eigenvalue, the figure of merit discussed in Chapter 3, or simple user selection. Additionally, a method of non-linear transformation of the original eigen-bases to a new reduced basis set was explored via the program **trainxnet**.

A.5 Implementing the Identity Verifier

The program verify_identity was written to fuse the verification probabilities produced by the programs **verify_face_net** and **verify_voice_net**. It functions by invoking the two verifiers in turn, and then simply calculating the linear combination of probabilities that resulted in the best overall system performance during training.

A.6 Code

The source code for most of the face and speaker verification programs described above are included here. As mentioned, the scripts for controlling the speaker capture and Cepstral processing can be found in a collateral thesis by Prescott (41).

A.6.1 acquire.c.

```
/*********************************************************************
Program:     acquire.c

Description:     This program was written to allow the capture of gray-scale
images from a video camera and the VideoPix image capturing hardware
in a Sun SPARCstation.  It will allow the creation of a new database
of training images, or the addition of new images to an existing data-
base.  It will similarly allow the creation of a new database of test
images or the addition of new images to an existing set.  Provisions
are included to either capture the images using manual segmentation,
or to capture images using Gay's automatic motion segmentation routines.

Author:          John G. Keller

Date:          1 Sep 93

*********************************************************************/

#include <stdio.h>
#include <string.h>
#include "vfc_lib.h"
#include "globals.h"
#include "jkmacros.h"

int     i,
    finished,
    done,
    quit,
    num_protos,
    num_train = 0,
    num_class = 1;

FILE     *fnograb;

char     command[80],
    u_name[8],
    nu_name[8],
    filename[20],
    waste[2],
    another[4],
    answer[4];
```

```
main(int argc, char *argv[])
{
int     testnum = 0;
char     testdir[30] = "", show[12];


system("clear");

/********* Make a Training Folder to Hold the Prototypes *******/

if (fopen("./train_images/*.gra") == NULL) system("mkdir train_images");
if ((argc == 2 || argc == 3) && strcmp("add", argv[1]) == 0)
    {
    open_read(fnograb, "nograb_param.dat");
    fscanf(fnograb, "%d\n", &num_protos);
    fscanf(fnograb, "%d\n", &num_class);
    fscanf(fnograb, "%d", &num_train);
    fclose(fnograb);
    }
else if (argc == 2 && strcmp("test", argv[1]) ≠ 0)
    {
    printf("\n\nSYNTAX:  acquire <add><test>.\n\n");
    exit(0);
    }

/********* Prompt User for Number of Prototypes if necessary*******/

if ((argc == 2 || argc == 3) && strcmp("add", argv[1]) == 0)
    {
    system("clear");
      printf("\nYou have chosen to add one or more individuals to the existing database.\nFor
this database you will need %d prototypes for each subject.\n", num_protos);
    num_class++;
    }
else
    {
    done = 0;
    while (!done)
        {
          printf("\nEnter the number of prototypes to be used for each user <1-64>: ");
          scanf("%d",&num_protos);
          /*gets(waste);*/

         printf("\n");
          if ((num_protos ≤ 64) && (num_protos ≥ 1))
            {
            done = 1;
              }
          else
            printf("\nYou need to do at least 1 and at most 64.\n");
        }
    }

/****************** Enter Users Until You're Done **************/

while (!finished)
    {
    done = 0;
    quit = 0;

/********* Prompt User for User Name *******/

    printf("\nEnter the person's username <8 letters or less>: ");
    scanf("%s", u_name);
    gets(waste);
    printf("\n");
    i = 0;
    while (u_name[i] ≠ '\0')
        {
        if (strlen(u_name) > 8)
            {
            printf("\nSorry, you're limited to 8 letters.\n");
            break;
            }

        if (isalpha(u_name[i]))
            i++;
        else
            {
            printf("\nSorry, you can't enter any numerics into the user name.\n");
            break;
            }
        }

     while(!done)
        {
        printf("\nThe name you entered was : %s\n",u_name);
        printf("Please re-enter the name if necessary or press return to continue.  ");
        gets(nu_name);
        if (nu_name[0] == '\0') done = 1;
        else
            {
             strcpy(u_name,nu_name);
               nu_name[0] = '\0';
            }
         }

/*********** grab training images of the user ***************/

     Loop1i(num_protos)
        {
          sprintf(filename,"%s%d",u_name,i);
          autograb(filename);
        sprintf(show, "display%d", SM_WIDTH);
        sprintf(command, "%s %s.gra", show, filename);
        system(command);
        printf("\nIs the picture satisfactory (y/n)? ");
           gets(answer);
        if (answer[0] == 'y')
            {
            if ( i < num_protos) printf("\nOkay, now for prototype number %d.\n", i+1);
            sprintf(filename,"%s%d%s",u_name,i,".gra");

            sprintf(command, "mv %s ./train_images", filename);
            system(command);
            num_train++;
            }
            else i -= 1;
        }

/********Grab test images if the 'test' argument was used**************/


if ((argc == 3 && strcmp(argv[2], "test") == 0) || (argc == 2 && strcmp(argv[1], "test") == 0))
    {
    if (testnum == 0)
        {
        printf("\nHow many test images will you want for each user? ");
        scanf("%d", &testnum);
        gets(waste);
        if (fopen("./test_images/*") == NULL)
            {
            sprintf(command, "mkdir ./test_images", testdir);
            system(command);
            }
        }
    Loop1i(testnum)
        {
          sprintf(filename,"%s%dt",u_name,i);
          autograb(filename);
        sprintf(command, "%s %s.gra", show, filename);
        system(command);
        printf("\nIs the picture satisfactory (y/n)? ");
           gets(answer);
        if (answer[0] == 'y')
            {
            if ( i < testnum) printf("\nOkay, now for test image number %d.\n", i+1);
            sprintf(filename,"%s%dt%s",u_name,i,".gra");

            sprintf(command, "mv %s ./test_images", filename, testdir);
            system(command);
            }
            else i -= 1;
        }
    }
    printf("\nProcess another individual? <y or n>: ");
    gets(another);
    while (!quit)
        {
        if ((another[0] == 'n') || (another[0] == 'N'))
            {
            finished = 1;

    /*********Save the nograb_param.dat information*************/

            open_write(fnograb, "nograb_param.dat");
            fprintf(fnograb, "%d\n", num_protos);
            fprintf(fnograb, "%d\n", num_class);
            fprintf(fnograb, "%d\n", num_train);
            fclose(fnograb);
            quit = 1;
             }
        else if ((another[0] == 'y') || (another[0] == 'Y'))
            {
            finished = 0;
            num_class++;
            quit = 1;
            }
        else
            {
           printf("\n\nHit y if you want to enter another user.\nHit n if you're done entering
users.  : ");
            gets(another);
            printf("\n");
             }
        }
    }
system("rm *.sm");
system("rm *.gra");

}
```

A.6.2 train_net.c.

```
/*******************************************************************
Program:    train_net.c

Description:  This program is used to train a system based on KLT feature
extraction and a neural net classifier.  The grab routine is first
called to collect the training images.  After all images of a
particular user have been collected, each of the images are
preprocessed (centered and gaussian windowed).  The preprocessed images
are then used by kl_transform to create an average face and a user determined
```

```
number of eigen faces. The coefficients module is then called to extract the kl coefficients from the training
images. These coefficients are stored in a data
file called klt.dat to be used by the neural network training algorithm.
The neural network algorithm creates a weight file which will be used
in the recognition phase. The outputs of this code are 1) the klt.dat
file, 2) the setup file for the network, and 3) the weight file created
by the network. All training images are stored in a folder called training_images for possible use in
retraining the system at a later date.

Author:     Ken Runyon

Date:     25 Sep 92

Modified by:   John Keller

Date:     15 Jul 93

Modification Description:
  - Modified to allow multiple command line options.
  - Modified to function with the face/speaker fused verification system. Images will be grabbed by the
program 'acquire' beforethis program is invoked. It will read prototype and image information from the
file nograb_param.dat, then will pre-process theimages if required. The basis set will be calculated, and
thenindividual weight files will be produced by training the net for
each user on the list user_list, with a particular user's files being
assigned to class 1 while all the other files are assigned to class
2 and so on.
  - Modified to perform same function with speaker files as face files. Will allow processing of individual
vectors from speaker codebook, or concatenation of raw speaker Cepstral vectors to allow retention of
temporal relationships.
  - Modified to permit selection between three different dimension ordering schemes: eigenvalue (traditional
KLT), figure of merit, and nonlinear transformation.
  - Modified to require external source_setup.mlp file be kept in same directory from which train_net is
run. This allows reparameterization without recompiling.
*************************************************************************/

#include <stdio.h>
#include <string.h>
#include "vfc_lib.h"
#include "globals.h"
#include "jkmacros.h"

/**********Define neural net parameters****************/

#define   NUM_LAYRS   2
#define   WT_SED      1918940490
#define PART_SED   1191645590
#define   RNDM_SED   123456789
#define MAX_ITS      1600
#define   OUT_INT      100
#define   ETA_IN     0.15
#define   ETA_OUT      0.3
#define ETA_1_2     0.0
#define   ALPHA      0.5
```

```
#define  BAT_SZ       1
#define  TRAIN_PCT    1.0
#define  NORM         1


void   voice_net_coefficients(), face_net_coefficients();

float   flnumber, fl_waste;

int    i, j, k,
    temp,
    done,
    num_protos,
    user_coefs,
    num_coefs,
    num_train = 0,
    num_class,
    number1,
    number2,
    type,
    FACE,
    SPEAKER,
    NOPROCESS,
    FOM,
    int_waste,
    num_vectors,
    vectors_per_class,
    num_features,
    num_chunks,
    chunk_size,
    num_in_chunk,
    total_num_coefs,
    min_num_vectors,
    user_counter = 1,
    identity,
    count,
    leftover;

FILE    *fparam,
    *flist,
    *fweights,
    *fset,
    *ftable,
    *fnograb,
    *fdat,
    *fdat2,
    *fuser,
    *fsample,
    *handle,
    *nonlindata;

char    wt_file[10],
   msg[40],
   dat_file[10],
   u_name[8],
   nu_name[8],
   filename[20],
   waste[2],
   username[16],
   username2[16],
   user1[20],
   hid_nodes,
   chwaste[40],
   hid_nodes2,
   command[30];

main(int argc, char *argv[])
{
if (argc == 1)
   {
  printf("\n\nSyntax:  train_net <face, speaker> <fom, eigvalue,nonlin>[noprocess (only valid
for face)]\n\n");
   exit(0);
   }

if (strcmp("face", argv[1]) == 0)
   {
   NOPROCESS = 0;
   if (strcmp(argv[2], "fom") == 0) FOM = 1;
   else if (strcmp(argv[2], "eigvalue") == 0) FOM = 0;
   else if (strcmp(argv[2], "nonlin") == 0) FOM = 2;
   else
      {
        printf("\n\nSyntax:  train_net <face, speaker> <fom, eigvalue,nonlin>[noprocess (only
valid for face)]\n\n");
      exit(0);
      }
   if (argc == 4 && (strcmp(argv[3], "noprocess") == 0)) NOPROCESS = 1;
   else if (argc == 4 && (strcmp(argv[3], "noprocess") ≠ 0))
      {
       printf("\n\nSyntax:  train_net <face, speaker> <fom, eigvalue, nonlin>[noprocess (only
valid for face)]\n\n");
      exit(0);
      }
   type = 1;
   FACE = 1;
   SPEAKER = 0;
   }
else if (strcmp("speaker", argv[1]) == 0)
   {
   if (strcmp(argv[2], "fom") == 0) FOM = 1;
   else if (strcmp(argv[2], "eigvalue") == 0) FOM = 0;
   else if (strcmp(argv[2], "nonlin") == 0) FOM = 2;
   else
```

```
      {
        printf("\n\nSyntax:  train_net <face, speaker> <fom, eigvalue,nonlin>[noprocess (only
valid for face)]\n\n");
      exit(0);
      }

   type = 0;
   FACE = 0;
   SPEAKER = 1;
   open_read(fuser, "speaker_list");
   }
else
   {
  printf("\n\nSyntax:  train_net <face, speaker> <fom, eigvalue, nonlin>[noprocess (only valid
for face)]\n\n");
   exit(0);
   }

/**************If Face processing, start here *******************/

NOPROCESS = 1;
if (FACE == 1 && fopen("*.gra", "r") ≠ NULL) system("rm *.gra");
if (FACE == 1)
   {

/***********Get the face info from the nograb_param.dat file*************/

   open_read(fnograb, "nograb_param.dat");
   fscanf(fnograb, "%d\n", &num_protos);
   fscanf(fnograb, "%d\n", &num_class);
   fscanf(fnograb, "%d", &num_train);
   fclose(fnograb);


/************Put together the list of training faces*********************/

   system("cp train_images/*.gra .");
   system("ls *.gra > face_list");
   open_read(flist, "face_list");
   if (NOPROCESS == 0)      /*If pre processing will be required...*/
      {

/***************Check that the correlation files exist ************/

      open_read(handle, "correlate.ref");
      open_read(handle, "wind.ref");

/*******************Pre-process the images********************/

      Loopli(num_train)
         {
         printf("\nNow pre-processing image # %...\r" [illegible]);
```

```
            fscanf(flist,"%s\n",filename);
          center(SM_WIDTH,"correlate.ref",filename);
          gwind(SM_WIDTH,filename);
          center(SM_WIDTH,"wind.ref",filename);
           }
       fclose(flist);
       }

/************ Decide how many eigenvectors you need ***********/

   done = 0;
   while (!done)
       {
       printf("\nEnter the number of eigenfaces on which you want to train <%d>: ",num_train/3);
         scanf("%d",&user_coefs);
         gets(waste);
        printf("\n");
         if ((user_coefs > 0) && (user_coefs <= num_train))
          {
          num_coefs = user_coefs;
          done = 1;
            }
         else if (user_coefs == '\0')
          {
          num_coefs = num_train/3;
          done = 1;
          }
         else
          printf("\nYou need to train on at least 1 and at most %d eigenfaces\n",num_train);
       }

   }

/*****************If speaker processing, start here**********************/

else if (SPEAKER == 1 && argc <= 3)
   {
   system("rm eigenspeaker*");
   printf("\nEnter the number of eigenspeakers on which you wish to train:  ");
   scanf("%d", &num_coefs);
   gets(waste);
   printf("\nEnter the number of individual eigenframes to concatenate:  ");
   scanf("%d", &num_in_chunk);
   gets(waste);
   }
else if (SPEAKER == 1 && argc > 3)
   {
   num_coefs = atoi(argv[3]);
   num_in_chunk = atoi(argv[4]);
   }
```

```
/************ Calculate the orthogonal basis set *****************/

if(FACE == 1)
   {
   if (fopen("eigenface*", "r") ≠ NULL) system("rm eigenface*");
  if (FOM == 1) fom_routine("face_list", type, num_coefs, SM_WIDTH, num_train, num_class, num_protos);
   else if (FOM == 0) klt_routine("face_list", type, num_coefs, SM_WIDTH, num_train);
   else if (FOM == 2) /*** Using non linear xformation***/
      {
      klt_routine("face_list", type, num_features, SM_WIDTH, num_train);
      }
   }

else if (SPEAKER == 1)
   {
   if (fopen("eigenspeaker*", "r") ≠ NULL)
      system("rm eigenspeaker*");
   if (FOM == 1)
      fom_routine("speaker_list", type, num_coefs);
   else if (FOM == 0)
      klt_routine("speaker_list", type, num_coefs);
   else if (FOM == 2)
      {
      int_waste = 20;
      klt_routine("speaker_list", type, int_waste, num_coefs);
      }
   }

num_class = 2; /****Classes are either Joe or Not-Joe************/

/****** Create the lookup table for the neural network ******/

open_write(ftable, "lookup");
fprintf(ftable, "%8s\n", "invalid");
fprintf(ftable, "%8s\n", "valid");
fclose(ftable);


if (FACE == 1)
   {
   open_read(flist, "face_list");
   open_write(fuser, "user_face_list");
   }
else if (SPEAKER == 1)
   {
   open_read(flist, "speaker_list");
   fscanf(flist, "%s", chwaste);
   fscanf(flist, "%d", &num_train);
   fscanf(flist, "%d", &vectors_per_class);
   fscanf(flist, "%d", &num_features);
   open_write(fuser, "user_speaker_list");
   }
```

```
strcpy(username2, "");
Loop1i(num_train)
   {
     fscanf(flist,"%s\n",filename);
   strcpy(username, filename);
   j = 0;
   while(username[j] ≠ '\0')
      {
      if (isalpha(username[j]))
         j++;
      else
         username[j] = 0;
      }
   if (strcmp(username2, username) ≠ 0)
      {
      strcpy(username2, username);
      fprintf(fuser, "%s\n", username2);
      }
   }
fclose(fuser);
fclose(flist);

/************ Create the data file for the neural network ************/

if (FACE == 1)
   {
   type = 1;
   open_read(flist, "face_list");
   open_write(fweights, "klt_f.dat");
   fprintf(fweights,"%d\n%d\n",num_coefs,num_class);
   system("rm eigcoeffs");
   Loop1i(num_train)
      {
        fscanf(flist,"%s\n",filename);
      if (FOM ≠ 2)
         face_net_coefficients(SM_WIDTH, num_coefs, filename, fweights, ftable, num_class, FOM, type);

/*************************************************************************
For the nonlinear transformation, first use 'net_coefficients to extract
all the eigencoefficients from the training set, then use 'xnetpush' to
transform these coefficients into num_coefs dimensions for each prototype.
*************************************************************************/

      else if (FOM == 2)
         face_net_coefficients(SM_WIDTH, num_train, filename, fweights, ftable, num_class, FOM, type);
      }
   fclose(fweights);
   if (FOM == 2)
      {
      open_read(fweights, "klt_f.dat");
      open_write(nonlindata, "facetest.dat");
      fprintf(nonlindata, "%d\n%d\n%d\n", num_train, num_train/num_protos, num_protos);
```

```
    Loopli(2)
       fscanf(fweights, "%d", &int_waste);
    Loopli(num_train)
       {
       fscanf(fweights, "%d", &int_waste);
       Looplj(num_train) /***Num of features***/
          {
          fscanf(fweights, "%f ", &fl_waste);
          fprintf(nonlindata, "%f ", fl_waste);
          }
       Looplj(2)
          fscanf(fweights, "%f", &fl_waste);
       fprintf(nonlindata, "\n");
       }
    fclose(fweights);
    fclose(nonlindata);
    sprintf(command, "make_bpnet %d", num_coefs);
    system(command);
    system("xnetpush facetest.dat");
    open_read(nonlindata, "newtest.dat");
    open_write(fweights, "klt_f.dat");
    fprintf(fweights, "%d\n%d\n", num_coefs, num_class);
    Loopli(num_train)
       {
       fprintf(fweights, "%d ", i - 1); /***Exemplar***/
       Looplj(num_coefs)
          {
          fscanf(nonlindata, "%f", &fl_waste);
          fprintf(fweights, "%f ", fl_waste);
          }
       fprintf(fweights, "0.90000 0.90000\n");
       }
    fclose(fweights);
    fclose(nonlindata);
    }

/***************************************************************************
This is the end of the nonlinear xformation part.  The klt_f.dat file
has been rebuilt with 'num_coefs' number of new features for each
prototype.
***************************************************************************/

  }
else if (SPEAKER == 1)
  {
  open_read(fuser, "user_speaker_list");
  min_num_vectors = 1000;
  Loopli(num_train)
     {
     fscanf(fuser, "%s", username);
     sprintf(msg, "%s.trainspeech", username);
     open_read(flist, msg);
```

```
        fscanf(flist, "%d", &temp);
        if (temp < min_num_vectors)
            min_num_vectors = temp;
        fclose(flist);
        }
    rewind(fuser);
    Loop1i(num_train)
        {
        fscanf(fuser, "%s", username);
        sprintf(msg, "%s.trainspeech", username);
        open_read(flist, msg);
        sprintf(msg, "%s.klt_s.dat", username);
        open_write(fweights, msg);
        if (FOM ≠ 2)
            voice_net_coefficients(num_features, num_coefs, fweights, flist, num_class, min_num_vectors);
        else
            voice_net_coefficients(num_features, num_features, fweights, flist, num_class, min_num_vectors);
        fclose(fweights);
        }
    rewind(fuser);
    open_write(fweights, "klt_s.dat");
    fprintf(fweights,"%d\n%d\n",num_coefs,num_class);
    Loop1i(num_train)
        {
        fscanf(fuser, "%s", username);
        sprintf(msg, "%s.klt_s.dat", username);
        open_read(handle, msg);
        Loop1j(min_num_vectors)
            {
            Loop1k(num_coefs)
                {
                fscanf(handle, "%f", &flnumber);
                fprintf(fweights, "%f ", flnumber);
                }
            fprintf(fweights,"\n");
            }
        fclose(handle);
        }
    fclose(fweights);
    fclose(fuser);


/***********************************************************************
We now have the klt_s.dat file containing an equal number of sets of
klt coefficients for each user.  We next will rewrite the klt_s.dat
file to represent concatenated "chunks" of vectors.  If using the
actual codebook vectors rather than the source speaker data, we don't
need to do this; we'll want each "chunk" equal to 1.
***********************************************************************/

    num_chunks = (int) (min_num_vectors/num_in_chunk);
    total_num_coefs = num_in_chunk * num_coefs;
```

```
   open_read(fweights, "klt_s.dat");

   open_write(handle, "temp.dat");
   Loop1i(2)
      {
      fscanf(tweights, "%d", &temp);
      fprintf(handle, "%d\n", temp);
      }
   count = 0;
   Loop1k(num_train)
      {
      Loop1i(num_chunks)
         {
         fprintf(handle, "%d ", count);
         Loop1j(total_num_coefs)
            {
            fscanf(fweights, "%f", &flnumber);
            fprintf(handle, "%f ", flnumber);
            }
         fprintf(handle, "0.10000 0.10000\n");
count++;
         }
      leftover = (min_num_vectors * num_coefs) - (num_chunks * total_num_coefs);
      Loop1j(leftover) fscanf(fweights, "%f", &waste);

      }
   fclose(handle);
   fclose(fweights);
   system("mv temp.dat klt_s.dat");
/**************************************************************************
This is the beginning of the non-linear transformation portions for
speakers. It functions the same as described above for faces.
**************************************************************************/

   if (FOM == 2)
      {
      open_read(fweights, "klt_s.dat");
      open_write(nonlindata, "spkrtest.dat");
      num_protos = 64;
      fprintf(nonlindata, "%d\n%d\n%d\n", num_features, num_train, num_protos);
      Loop1i(2)
         fscanf(fweights, "%d", &int_waste);
      Loop1i(num_train)
         {
         fscanf(fweights, "%d", &int_waste);
         Loop1j(num_train)
            {
            fscanf(fweights, "%f ", &fl_waste);
            fprintf(nonlindata, "%f ", fl_waste);
            }
         Loop1j(2)
            fscanf(fweights, "%f %f", &fl_waste);
```

```
        fprintf(nonlindata, "\n");
        }
    fclose(fweights);
    fclose(nonlindata);
    sprintf(command, "make_bpnet %d", num_coefs);
    system(command);
    system("xnetpush spkrtest.dat");
    open_read(nonlindata, "newtest.dat");
    open_write(fweights, "klt_f.dat");
    fprintf(fweights, "%d\n%d\n", num_coefs, num_class);
    Loop1i(num_train)
        {
        fprintf(fweights, "%d ", i - 1);
        Loop1j(num_coefs)
            {
            fscanf(nonlindata, "%f", &fl_waste);
            fprintf(fweights, "%f ", fl_waste);
            }
        fprintf(fweights, "0.90000 0.90000\n");
        }
    fclose(fweights);
    fclose(nonlindata);
    }


/*************************************************************************
This is the end of the nonlinear xformation part.  The klt_s.dat file
has been rebuilt with 'eigcoeffs' number of new features for each
prototype.
*************************************************************************/


  } /***End 'if SPEAKER == 1' clause***/


/*************************************************************************
So now the file klt_s.dat contains m rows of eigen-coefficients
corresponding to m prototypes for each user.
*************************************************************************/

/*****Create the train_params file for the face verification phase *****/

if (FACE == 1)
  {
  open_write(fparam, "train_f_params");
  fprintf(fparam,"%d\n%d\n%d\n%d\n%d\n",SM_WIDTH,num_coefs,num_train, num_protos, num_class);
  }
else if (SPEAKER == 1)
  {
  open_write(fparam, "train_s_params");
  fprintf(fparam,"%d\n%d\n%d\n%d\n",num_coefs, num_class, num_in_chunk, num_train - 1);
  }
```

```
fclose(fparam);

/****************Now train the net for each user******************/

if (FACE == 1)
   strcpy(filename, "user_face_list");
else if (SPEAKER == 1)
   strcpy(filename, "user_speaker_list");

open_read(fuser, filename);
if (fopen("temp.dat", "r") ≠ NULL)
   system("rm temp.dat");


while (fscanf(fuser, "%s", username) ≠ EOF)
   {
   printf("\n\n\n\n\n                                    Training the net for user %s.\n", username);


/*********************************************************************
Now copy the klt.dat file created during the (voice_)net_coefficients program
to a new file, replacing the last two numbers in each vector with 0.1 and 0.9,
depending on which class the particular image (speaker) belongs to.  Remember,
there are only two classes: either the vector is a particular user,
or it's one of the other users.
*******************************************************************/

   if (FACE == 1)
      {
      open_read(flist, "face_list");
      open_read(fdat, "klt_f.dat");
      open_write(fdat2, "temp.dat");
      fscanf(fdat, "%d\n%d\n", &number1, &number2);
      fprintf(fdat2, "%d\n%d\n", number1, number2);
      Loop1i(num_train)
         {
         fscanf(fdat, "%d", &number1);
         fprintf(fdat2, "%d ", number1);
         Loop1j(num_coefs)
            {
            fscanf(fdat, "%f", &flnumber);
            fprintf(fdat2, "%f ", flnumber);
            }
         fscanf(fdat, "%f", &flnumber);
         fscanf(fdat, "%f", &flnumber);
           fscanf(flist,"%s\n", &filename);
         strcpy(user1,filename);


/*********************************************************************
Parse the current images filename to determine whether it is the
same as the user currently being trained on.
***************************************************************************/
         j = 0;
         while(user1[j] ≠ '\0')
            {
            if (isalpha(user1[j]))
               j++;
            else
               user1[j] = 0;
            }

         if (strcmp(user1, username) == 0)
            {
            fprintf(fdat2, "0.900000 0.100000\n");
            }
         else
            fprintf(fdat2, "0.100000 0.900000\n");
         }
      }  /***End of 'if (FACE == 1)' clause ***/

   else if (SPEAKER == 1)
      {
      open_read(fdat, "klt_s.dat");
      open_write(fdat2, "temp.dat");
      fscanf(fdat, "%d\n%d\n", &number1, &number2);
      fprintf(fdat2, "%d\n%d\n", total_num_coefs, number2);
      Loop1i(num_train)
         {
         if (i == user_counter) identity = 1;
         else identity = 0;
         Loop1j(num_chunks)
            {
            fscanf(fdat, "%d", &number1);
            fprintf(fdat2, "%d ", number1);
            Loop1k(total_num_coefs)
               {
               fscanf(fdat, "%f", &flnumber);
               fprintf(fdat2, "%f ", flnumber);
               }
            if (identity == 1)
               fprintf(fdat2, "0.900000 0.100000\n");
            else
               fprintf(fdat2, "0.100000 0.900000\n");
            Loop1k(2)
               fscanf(fdat, "%f", &flnumber);
            }
         }
      }

   user_counter++;
   fclose(fdat);
   fclose(fdat2);
   fclose(flist);
```

```
    if (SPEAKER == 1) system("mv temp.dat klt_s.dat");
    else if (FACE == 1) system("mv temp.dat klt_f.dat");

/****************Assign user name to weights file*******************/

    if (FACE == 1) sprintf(wt_file, "%s_f_klt.wts", username);
    else if (SPEAKER == 1) sprintf(wt_file, "%s_s_klt.wts", username);


/************ Create the setup file for the neural network **********/

    if (SPEAKER == 1)
       {
       num_coefs = total_num_coefs;
       strcpy(dat_file,"klt_s.dat");
       }
    else
       strcpy(dat_file, "klt_f.dat");
    hid_nodes = 2 * num_coefs;
    hid_nodes2 = 0;

    open_read(fsample, "source_setup.mlp");
    open_write(fset, "setup.mlp");
    Loopli(4)
       {
       fscanf(fsample, "%d", &int_waste);
       fprintf(fset, "%d\n", int_waste);
       }
    fprintf(fset, "%s           -store weights\n", wt_file);
    fscanf(fsample, "%d", &int_waste);  /***Max iterations***/
    fprintf(fset, "%d\n", int_waste);
    fprintf(fset, "%d %d %d %d\n", num_coefs, hid_nodes, hid_nodes2, num_class);
    fprintf(fset, "%s                     -data\n", dat_file);
    fscanf(fsample, "%d", &int_waste);
    fprintf(fset, "%d\n", int_waste);
    Loopli(4)
       {
       fscanf(fsample, "%f", &fl_waste);
       fprintf(fset, "%f\n", fl_waste);
       }
    fprintf(fset, "%d\n%f\n%d\n", BAT_SZ,TRAIN_PCT,NORM);
    fclose(fset);
    fclose(fsample);

    system("mlp_fuse_trn");
    } /*******Closing while loop*****/

fclose(fuser);
if (FACE == 1) system("rm *.gra");

printf("\nTRAINING IS COMPLETE\n");
}
```

A.6.3 face_net_coefficients.c.

```
/*****************************************************
Program:   face_net_coefficients.c


Description:   This program maps a test face onto the set of eigenfaces and stores the KL coefficients in
train_coefs in a format the neural network can read.

Author:       Pedro Suarez    (Originally recon.c)

Date:      24 July 91

Modified by:   Ken Runyon    (Chopped off reconstruction)

Date:      22 Jun 92

Modification Description:   I decided we didn't need to actually reconstruct and store a face.  I also made
the stand alone program into a module which is called by thesis.
*****************************************************/
#include <stdio.h>
#include <math.h>
#include "jkmacros.h"

void face_net_coefficients(dimension, num_coefs, infilename, outfile, classfile, num_class,fom, type)
   int   dimension,
      num_coefs,
      num_class,
      fom,
      type;

   char   infilename[];

   FILE   *outfile,
      *classfile;
{
FILE    *facel, *eigenin, *train, *face_avg, *testfile;
int    i, j, l, N, M, atoi();
static int count = 0, exemplar = 0;
float     *vector(), **matrix(), *average_face, **u, *pedro, *reconface;
float     *w, *I;
char   filename[81], *strcpy(), user[15], ext[10];
static char userl[15],user2[15];


#ifndef RESULTS
printf("\nPulling Coefficients for %s\n",infilename);
#endif


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

```
/*** Open Test Face ***/

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

 /*** Open Avg Face ***/

if ((face_avg=fopen("avg_face.dat","r")) == NULL){
  printf("I can't open avg_face.dat.");
  exit(-1);
  }

/****** set up matrices ******/

N = dimension * dimension;
M = num_coefs;

u = matrix(1,N, 1,M);
pedro = vector(1, N);
average_face = vector(1, N);
reconface = vector(1, N);    /*** DO I NEED THIS? ***/
w = vector(1, M);
I =  vector(1, N);


/****** Initalize Matrices  ******/

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;

/********* Load the Test Face into the Pedro Vector  *********/

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

 /********* Load the Average Face into the Average_Face Vector ****/

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

/****** Load the Eigenfaces into Matrix U ******/

open_read(train, "face_train.out");

for(j=1; j≤M; j++){
    fscanf(train, "%s\n", filename);
    eigenin = fopen(filename, "r");
    for(i=1;i≤N;i++){
        fscanf(eigenin,"%f\n",&u[i][j]);
    }
    fclose(eigenin);
}
fclose(train);

/****** Subtract the Average Face from the Test Face ******/

    for(i=1;i≤N;i++)
        I[i]= pedro[i] – average_face[i];

/****** Calculate the KL Coefficients ******/

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

/****** Write an exemplar number to the file ******/

fprintf(outfile, "%d    ", exemplar);
exemplar++;



/****** Write the Coefficients to the *_coefs File ******/

/*******Write test file containing just the eigcoeffs******/

open_append(testfile, "eigcoeffs");


for(i=1; i≤M; i++)
    {
    fprintf(outfile, "%f  ", w[i]);
    fprintf(testfile, "%f  ", w[i]);
    }
fprintf(testfile, "\n");
fclose(testfile);

/****** Write the desired outputs to the *_coefs File ******/


for(l=num_class;l≥1;l--)
    fprintf(outfile,"%f    ", 0.90000);

fprintf(outfile, "\n");
free_matrix(u,1,N,1,M);

} /* end coefficients.c */
```

A.6.4 voice_net_coefficients.c.

```
/*****************************************************
Program:    voice_net_coefficients.c

Description:    This program will take a set of voice codebooks, extract the KLT coefficients of each, and
write them to the file klt.dat for use by the net.

Author:         John G. Keller (based on the program "recon.c" originally written by Pedro Suarez and
modified by Ken Runyon).

Date:       22 Aug 93

*****************************************************/
#include <stdio.h>
#include <math.h>
#include "jkmacros.h"

void voice_net_coefficients(length, num_coefs, datfile, userfile, num_class, num_train)
   int    length,
      num_coefs,
      num_class,
      num_train;


   FILE    *datfile,
      *userfile;
{
FILE     *code, *eigenin, *train, *avg_voice;
int     i, j, k;
int      exemplar = 0,
   waste;
float       *vector(),
   **matrix(),
   *average_voice,
   **u,
   *voice,
   *w,
   *I;

char    filename[81];


/****** set up matrices ******/

u = matrix(1,length, 1,num_coefs);
voice = vector(1, length);
average_voice = vector(1, length);
w = vector(1, num_coefs);
I =  vector(1, length);
```

```
/****** Initalize Matrices ******/

Looplij(num_coefs, length)
   w[i]=u[j][i]=I[j]=voice[j]=average_voice[j]=0.0;

/********* Load the average voice into the average_voice vector ****/

open_read(avg_voice, "avg_speaker.dat");
Loopli(length)
        fscanf(avg_voice, "%f\n", &average_voice[i]);
fclose(avg_voice);

/****** Load the Eigenspeakers into matrix U ******/

open_read(train, "speaker_train.out");
Loopli(num_coefs)
   {
   fscanf(train, "%s\n", filename);
   open_read(eigenin, filename);
   Looplj(length)
      {
      fscanf(eigenin,"%f\n",&u[j][i]);
      }
   fclose(eigenin);
   }
fclose(train);

/**********************************************************
Now loop through the data, taking one full vector at a time and
pulling the klt coefficients.
**********************************************************/
Loopli(2) fscanf(userfile, "%f", &waste);
Looplk(num_train)
   {

/********* Load the speaker vector into voice_vector *********/

   Loopli(length)
      fscanf(userfile, "%f", &voice[i]);

/****** Subtract off the average voice ******/

   Loopli(length)
      I[i]= voice[i] - average_voice[i];

/****** Calculate the KL Coefficients ******/

   w[1] = 0.0;
   Looplij(num_coefs, length)
      w[i] = u[j][i] * I[j]+ w[i];

/****** Write the Coefficients to <user>_s.klt.dat ******/
```

```
   Loopli(num_coefs)
      fprintf(datfile, "%f ", w[i]);

   fprintf(datfile, "\n");
   }
fclose(userfile);

free_matrix(u,1,length,1,num_coefs);
free_vector(voice, 1, length);
free_vector(average_voice, 1, length);
free_vector(w, 1, num_coefs);
free_vector(I, 1, length);
}
```

A.6.5 klt_routine.c.

```
/*********************************************************************
Program:        klt_routine.c

Description:   This routine is based around the Jacobi rotation routine
found in Numerical Recipes in C.  Given a data matrix, it will first calculate
the covariance matrix, and then the eigenvectors.  The eigenvector matrix
is then put into descending eigenvalue order and returned to the calling
program.  The eigenvectors may also be printed out here by un-commenting
the appropriate lines.  The routine can be used with either face or speaker data.

Author:        John G. Keller/Dennis Krepp

Date:      1 Sep 93
*********************************************************************/
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "jkmacros.h"

klt_routine(char file_list[], int type, int num_eigvectors, int dim, int num_train, int increment)
{

FILE   *datfile, *outfile, *average, *train, *code;
int    length,
   waste,
   num_codewords,
   num_classes,
   num_keep_dimensions,
   dim_number;

int    i,j, N, k, M,
   nrot;

float      **matrix(),
    *vector(),
    **A,
    **A_trans,
    **u,
    **L,
    **v,
    *d,
    *average_temp,
    temp;

void    free_vector(), free_matrix(), eigsrt(), jacobi();

char    type_name[30],
    avg_file[30],
    msg[30],
    msg1[30],
    filename[40],
    file[40],
    data_filename[40];

open_read(train, file_list);
if (type == 0)  /***If speaker verification***/
    {
    fscanf(train, "%s", data_filename);
    fscanf(train, "%d", &num_classes);
    fscanf(train, "%d", &num_codewords);
    fscanf(train, "%d", &length);
    num_train = num_classes * num_codewords;
    }
else if (type == 1) /***If face verification***/
    {
    length = dim * dim;
    }
i = 0;
strcpy(type_name, "");
while (file_list[i] ≠ '.' || file_list[i] ≠ '_')
    {
    if (file_list[i] == '.' || file_list[i] == '_')
        {
        type_name[i] = 0;
        break;
        }
    type_name[i] = file_list[i];
    ;
    i++;
    }

/****** Allocate memory  ******/

A_trans = matrix(1,num_train,1,length);
A = matrix(1,length,1,num_train);
average_temp = vector(1,length);
```

```
if (type == 0) /***Speaker***/
   {
   L = matrix(1,length,1,length);
   d = vector(1,length);
   v = matrix(1,length,1,length);
   }
else if (type == 1)
   {
   L = matrix(1,num_train,1,num_train);
   d = vector(1,num_train);
   v = matrix(1,num_train,1,num_train);
   }


/****** Initalize matrices and vectors ******/

if (type == 0)  /***If speaker***/
   {
   Loop1i(num_train)
      {
      Loop1j(length)
         A_trans[i][j] = A[j][i] = 0.0;
      }
   Loop1i(length)
      {
      d[i] = average_temp[i] = 0.0;
      Loop1j(length)
         L[i][j] = v[i][j] = 0.0;
      }
   }
else if (type == 1)
   {
   Loop1i(num_train)
      {
      d[i] = average_temp[i] = 0.0;
      Loop1k(num_train)
         v[i][k] = L[i][k] = 0.0;
      Loop1j(length)
         A_trans[i][j] = A[j][i] = 0.0;
      }
   }

if (type == 0) /***Speaker***/
   {
   printf("\nThe users being trained on are :\n\n");
   open_read(code, data_filename);
   Loop1i(num_train)
      {
      if (i == 1 || ((num_codewords + 1) % i) == 0)
         {
         fscanf(train, "%s\n", filename);
```

```
            printf("\t\t%s\n", filename);
            }
         Loop1j(length)
            {
            fscanf(code, "%f", &A[j][i]);
            }
         }
      }
   else if (type == 1)
      {
      printf("The files being trained on are :\n\n");
      Loop1i(num_train)
         {
         fscanf(train, "%s\n", filename);
         printf("\t\t%s\n", filename);
         open_read(code, filename);
         Loop1j(length)
            fscanf(code, "%f\n", &A[j][i]);
         fclose(code);
         }
      }

   if (type == 0) fclose(code);
   fclose(train);

   /***************Calculate average vector*******************/

   sprintf(avg_file, "avg_%s.dat", type_name);
   open_write(average, avg_file);
   Loop1i(length)
      {
      temp = 0.0;
      Loop1j(num_train)
         {
         temp += A[i][j];
         }
      average_temp[i] = temp/num_train;
      fprintf(average, "%f\n", average_temp[i]);
      }
   fclose(average);

   /**************Subtract average vector**********************/

   Loop1j(num_train)
      Loop1i(length)
         A[i][j] = A[i][j] - average_temp[i];

   free_vector(average_temp, 1, length);

   /*************Make transpose matrix***********************/

   Loop1j(num_train)
```

```
    Loop1i(length)
        A_trans[j][i] = A[i][j];

/******************Matrix multiply A by itself***************/

if (type == 0) /***Speaker***/
    {
    Loop1i(length)
        Loop1j(length)
            {
            temp = 0.0;
            Loop1k(num_train)
                temp = temp + A_trans[k][i] * A[j][k];
            L[i][j] = temp;
            }
    }
else if (type == 1)
    {
    Loop1i(num_train)
        Loop1j(num_train)
            {
            temp = 0.0;
            Loop1k(length)
                temp = temp + A_trans[i][k] * A[k][j];
            L[i][j] = temp;
            }
    }
free_matrix(A_trans, 1, num_train, 1, length);

/**************Do Jacobi rotation and sort eigenstuff***********/
if (type == 0) /***Speaker***/
    {
    jacobi(L, length, d, v, &nrot);
    eigsrt(d, v, length);
    }
else if (type == 1)
    {
    jacobi(L, num_train, d, v, &nrot);
    eigsrt(d, v, num_train);
    }

/****************Find eigenbases*******************************/

u = matrix(1, length, 1, num_train);
Loop1i(num_train)
    Loop1j(length)
        u[j][i] = 0.0;

if (type == 0) /***Speaker***/
    {
    Loop1i(num_train)
        Loop1j(length)
```

```
            Loop1k(length)
                u[j][i] = v[k][j] * A[k][i] + u[j][i];

    }
else if (type == 1)
    {

    Loop1i(num_eigvectors)
        Loop1j(num_train)
            Loop1k(length)
                u[k][i] = v[j][i] * A[k][j] + u[k][i];

    }

/***********Write file containing list of eigenvector names*********/

sprintf(msg, "%s_train.out", type_name);
open_write(outfile, msg);
sprintf(msg1, "eigen%s", type_name);
Loop1i(num_eigvectors)
    {
    sprintf(file, "%s%d.dat", msg1, i);
    fprintf(outfile, "%s\n", file);
    }
fclose(outfile);

/***********Write out the eigenbases*****************************/

open_read(outfile, msg);
Loop1i(num_eigvectors)
    {
    fscanf(outfile, "%s", &file);
    open_write(datfile, file);
    Loop1j(length)
        fprintf(datfile, "%g\n", u[j][i]);

    fclose(datfile);
    }
fclose(outfile);

free_matrix(A, 1, length, 1, num_train);
free_matrix(u, 1, length, 1, num_train);
if (type == 0)
    {
    free_matrix(L, 1, length, 1, length);
    free_matrix(v,1, length,1, length);
    free_vector(d, 1,length);
    }
else if (type == 1)
    {
    free_matrix(L, 1,num_train,1,num_train);
    free_matrix(v,1,num_train,1,num_train);
```

```
   free_vector(d, 1,num_train);
   }

}
```

A.6.6 fom_routine.c.

```
/*************************************************************************
Program:   fom_routine.c


Description:  The purpose of this program is to allow a comparison
between the eigenvectors with the highest eigenvalues (found during
the KL transformation) and the in-class importance of those eigen-
dimensions when each class is individually projected into the
eigenspace.  The program will:

   - read in data from a sourcefile containing multiple classes and
   multiple data points of some dimension n
   - read in an eigenspace defined (via the klt process) from the
    raw data points
   - calculate the in-class and across-class means and variances
   of the eigen-dimensions
   - calculate a figure of merit (FOM) for each set of eigen-
   dimensions

   The program requires a user-specified input data file containing class data.  The first three lines of
the data file must contain the number of classes within the file, the number of vectors per class, and the
dimensionality of each vector.  The listing of vectors, by class, will follow.
   The output of the program will be a listing of the above
statistics.  The output eigen-dimensions will be listed in order of
decreasing eigenvalues, and we will be able to make a direct comparison
between the FOMs and those eigenvalues to determine which eigen-dimensions
appear to be most important for the given data set.

Author:      John G. Keller

Date:     29 Aug 93

 *************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "nrutil.h"
#include "jkmacros.h"


fom_routine(char file_list[], int type, int num_coefs, int dim, int num_train, int start_classes, int num_protos)
{
char    command[40], msg[40], msg1[40], data_filename[40], type_name[30], file[40];
```

```
int     vectors_per_class,
    num_eigcoefs,
    num_eigvectors,
    num_classes,
    num_ftrs,
    total_vectors,
    i, j, k;

float    temp1,
    temp2,
    sum_eigvalue,
    sum_fom,
    *fom,
    *across_class_variance,
    *across_class_mean,
    *mean_vector,
    *eigvalue,
    *fom_ordered_vector,
    *mean_of_var,
    *var_of_means,
    *fom_error,
    *cum_fom_error,
    *eig_error,
    *cum_eig_error,
    **eigenmatrix,
    **fom_ordered_eigmatrix,
    **data_matrix,
    **newdata_matrix,
    **class_mean,
    **class_variance,
    **u;

FILE    *datfile, *outfile, *fhandle, *train, *avg_file;


/*****************************************************************************
Read in initialization data from source files
*****************************************************************************/

open_read(train, file_list);
if (type == 0)  /***If speaker verification***/
    {
    fscanf(train, "%s", data_filename);
    fscanf(train, "%d", &num_classes);
    fscanf(train, "%d", &vectors_per_class);
    fscanf(train, "%d", &num_ftrs);
    num_eigvectors = num_eigcoefs = num_ftrs;
    }
else if (type == 1) /***If face verification***/
    {
    num_ftrs = dim * dim;
    num_eigvectors = num_eigcoefs = num_train;
    num_classes = start_classes;
    vectors_per_class = num_protos;
    }
i = 0;
strcpy(type_name, "");
while (file_list[i] ≠ '.' || file_list[i] ≠ '_')
    {
    if (file_list[i] == '.' || file_list[i] == '_')
        {
        type_name[i] = 0;
        break;
        }
    type_name[i] = file_list[i];
    ;
    i++;
    }

total_vectors = num_classes * vectors_per_class;

/*****************************************************************************
Declare/initialize matrices and vectors
*****************************************************************************/

fom = vector(1, num_eigvectors);
across_class_variance = vector(1, num_classes);
across_class_mean = vector(1, num_classes);
mean_vector = vector(1, num_ftrs);
eigvalue = vector(1, num_eigvectors);
mean_of_var = vector(1, num_ftrs);
var_of_means = vector(1, num_ftrs);
fom_ordered_vector = vector(1, num_ftrs);
eig_error = vector(1, num_ftrs);
cum_eig_error = vector(1, num_ftrs);
fom_error = vector(1, num_ftrs);
cum_fom_error = vector(1, num_ftrs);

eigenmatrix = matrix(1, num_eigcoefs, 1, num_eigvectors);
fom_ordered_eigmatrix = matrix(1, num_eigcoefs, 1, num_eigvectors);
data_matrix = matrix(1, num_ftrs, 1, total_vectors);
newdata_matrix = matrix(1, num_ftrs, 1, total_vectors);
class_mean = matrix(1, num_ftrs, 1, num_classes);
class_variance = matrix(1, num_ftrs, 1, num_classes);
u = matrix(1, num_ftrs, 1, total_vectors);

Loop1i(num_eigcoefs)
    {
    fom[i] = eigvalue[i] = mean_vector[i] = 0.0;
    Loop1j(num_eigvectors)
        eigenmatrix[i][j] = 0.0;
    Loop1j(num_classes)
        class_mean[i][j] = class_variance[i][j] = 0.0;
    }
```

```
Loopli(num_classes)
   across_class_variance[i] = across_class_mean[i] = 0.0;
Loopli(num_ftrs)
   {
   mean_vector[i] = mean_of_var[i] = var_of_means[i] = 0.0;
   Looplj(total_vectors)
      data_matrix[i][j] = newdata_matrix[i][j] = 0.0;
   }

/***************************************************************************
KL transform the source data set to form the eigenspace.
***************************************************************************/

find_klspace(data_filename, eigenmatrix, eigvalue, mean_vector, data_matrix, num_classes, vectors_per_class,
num_ftrs, type);

/***Bring the eigenvalue back from find_klspace in case we want it later***/

/****************************************************************************
Load mean_vector, eigenmatrix, eigvalue, and data_matrix
****************************************************************************/

Loopli(num_eigvectors)
   Looplj(num_eigcoefs)
      fom_ordered_eigmatrix[j][i] = eigenmatrix[j][i];

/****************************************************************************
Save the mean vector to the file ___avg.dat for use by the coefficient
extraction routine (_net_coefficients).
****************************************************************************/

if (type == 0)
   {
   open_write(avg_file, "avg_speaker.dat");
   }
else if (type == 1)
   {
   open_write(avg_file, "avg_face.dat");
   }

Loopli(num_ftrs)
   fprintf(avg_file, "%f ", mean_vector[i]);
fclose(avg_file);

/****************************************************************************
Calculate newdata_matrix
****************************************************************************/

if (type == 0)
   {
   Loopli(total_vectors)
      Looplj(num_eigvectors)
```

```
         Loop1k(num_ftrs)
            newdata_matrix[j][i] += data_matrix[k][i] * eigenmatrix[k][j];
   }
else if (type == 1)
   {
   Loop1i(total_vectors)
      Loop1j(total_vectors)
         Loop1k(num_ftrs)
            newdata_matrix[k][i] += data_matrix[k][j] * eigenmatrix[j][i];
   }

/***************************************************************************
Calculate class_mean and class_variance matrices
****************************  *********************************************/

Loop1i(num_classes)
   Loop1j(num_ftrs)
      {
      temp1 = 0.0;
      Loop1k(vectors_per_class)
         {
         temp1 += newdata_matrix[j][k + ((i - 1) * vectors_per_class)];
         }
      class_mean[j][i] = temp1/vectors_per_class;
      }

Loop1i(num_classes)
   Loop1j(num_ftrs)
      {
      temp2 = 0.0;
      Loop1k(vectors_per_class)
         {
            temp2 += (newdata_matrix[j][k + ((i - 1) * vectors_per_class)] - class_mean[j][i]) * (new-
data_matrix[j][k + ((i - 1) * vectors_per_class)] - class_mean[j][i]) ;
         }
      class_variance[j][i] = temp2/vectors_per_class;
      }


/***************************************************************************
Calculate across_class_mean matrix
***************************************************************************/

Loop1i(num_ftrs)
   {
   temp1 = 0.0;
   Loop1j(num_classes)
      temp1 += class_mean[i][j];
   across_class_mean[i] = temp1/num_classes;
   }
```

```
/***************************************************************************
Calculate mean_of_var and var_of_mean vectors
***************************************************************************/

Loop1i(num_ftrs)
   {
   temp1 = 0.0;
   temp2 = 0.0;
   Loop1j(num_classes)
      {
      temp1 += class_variance[i][j];
      temp2 += (class_mean[i][j] - across_class_mean[i]) * (class_mean[i][j] - across_class_mean[i]);
      }
   mean_of_var[i] = temp1/num_classes;
   var_of_means[i] = temp2/num_classes;
   }

/***************************************************************************
Calculate fom (Figure of Merit) vector
***************************************************************************/

Loop1i(num_ftrs)
   fom_ordered_vector[i] = fom[i] = var_of_means[i]/mean_of_var[i];

/***************************************************************************
Reorder the eigenvectors in order of decreasing FoM.
***************************************************************************/

eigsrt(fom_ordered_vector, fom_ordered_eigmatrix, num_eigvectors);

/***For testing, print out the eigvalue ordered vector vs the
FOM ordered vector
***/

open_write(outfile, "eigvsfom.out");
fprintf(outfile, "#\tEigvalue\tFoM Value\t%fOrdered FoM\n\n");
Loop1i(num_eigvectors)
   {
   fprintf(outfile, "%d)\t%f\t%f\t\t%f\n", i, eigvalue[i], fom[i], fom_ordered_vector[i]);
   }
fclose(outfile);


exit(0);

/****************Find eigenbases*******************************/

u = matrix(1, num_ftrs, 1, num_coefs);
Loop1i(num_coefs)
   Loop1j(num_ftrs)
      u[j][i] = 0.0;
```

```
if (type == 0) /***Speaker***/
   {
   Loop1i(num_coefs)
      Loop1j(num_ftrs)
         Loop1k(num_ftrs)
            u[j][i] = fom_ordered_eigmatrix[k][j] * data_matrix[k][i] + u[j][i];
   }
else if (type == 1)
   {
   Loop1i(num_coefs)
      Loop1j(total_vectors)
         Loop1k(num_ftrs)
            u[k][i] = (fom_ordered_eigmatrix[j][i] * data_matrix[k][j]) + u[k][i];
   }

/***********Write file containing list of eigenvector names*********/

sprintf(msg, "%s_train.out", type_name);
open_write(outfile, msg);
sprintf(msg1, "eigen%s", type_name);
Loop1i(num_eigvectors)
   {
   sprintf(file, "%s%d.dat", msg1, i);
   fprintf(outfile, "%s\n", file);
   }
fclose(outfile);

/******************Write out eigenbasis files**********************/

open_read(outfile, msg);
Loop1i(num_coefs)
   {
   fscanf(outfile, "%s", &file);
   open_write(datfile, file);
   Loop1j(num_ftrs)
      fprintf(datfile, "%g\n", u[j][i]);
   fclose(datfile);
   }
fclose(outfile);


/***************************************************************************
Free up data structure memory.
***************************************************************************/

free_matrix(u, 1, num_ftrs, 1, num_coefs);
free_vector(fom, 1, num_eigcoefs);
free_vector(across_class_variance, 1, num_classes);
free_vector(across_class_mean, 1, num_classes);
free_vector(mean_vector, 1, num_ftrs);
free_vector(eigvalue, 1, num_eigvectors);
free_vector(mean_of_var, 1, num_ftrs);
free_vector(var_of_means, 1, num_ftrs);
free_vector(fom_ordered_vector, 1, num_ftrs);
free_vector(fom_error, 1, num_ftrs);
free_vector(cum_fom_error, 1, num_ftrs);
free_vector(eig_error, 1, num_ftrs);
free_vector(cum_eig_error, 1, num_ftrs);

free_matrix(eigenmatrix, 1, num_eigcoefs, 1, num_eigvectors);
free_matrix(fom_ordered_eigmatrix, 1, num_eigcoefs, 1, num_eigvectors);
free_matrix(data_matrix, 1, num_ftrs, 1, total_vectors);
free_matrix(newdata_matrix, 1, num_ftrs, 1, total_vectors);
free_matrix(class_mean, 1, num_eigcoefs, 1, num_classes);
free_matrix(class_variance, 1, num_eigcoefs, 1, num_classes);


}
```

A.6.7 find_klspace.c.

```
/*********************************************************************
Program:        find_klspace.c

Description:  This routine is based around the Jacobi rotation routine
found in Numerical Recipes in C, and will calculate the eigenspace associated
with a set of data vectors.  Given the vector matrix, it will first calculate
the covariance matrix, and then the eigenvectors.  The eigenvector matrix
is then put into descending eigenvalue order and returned to the calling
program.  The eigenvectors may also be printed out here by un-commenting
the appropriate lines.  The code was written to be used with either face
or speaker data.  This routine was written to be used by the program fom_routine.

Author:       John G. Keller

Date:      1 Sep 93

*************************************************************************/

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


find_klspace(char infilename[], float **eig_vectors, float *eig_values, float *average_temp, float **data_matrix,
int num_classes, int vectors_per_class, int num_ftrs, int type)
{

FILE   *datfile,
   *outfile,
   *average,
   *infile,
   *train,
    *protofile;

int     i,j, N, k, M,
    nrot,
    num_train;

float       **matrix(),
    *vector(),
    **data_matrix_trans,
    **u,
    **L,
    temp;

void      free_vector(),
    free_matrix(),
    eigsrt(),
    jacobi();

char    file[40],
    filename[40];

if (type == 0)
    {
    open_read(datfile, infilename);
    }
num_train = num_classes * vectors_per_class;


/****** Allocate memory  ******/

data_matrix_trans = matrix(1,num_train,1,num_ftrs);

if (type == 1) /***If face verification***/
    {
    L = matrix(1, num_train, 1, num_train);
    }
else if (type == 0) /***If voice verification***/
    {
    L = matrix(1, num_ftrs, 1, num_ftrs);
    }

/****** Initalize matrix and vectors  ******/

if (type == 1) /***If face verification***/
    {
    Loop1i(num_train)
        {
        Loop1j(num_ftrs)
            data_matrix_trans[i][j] = data_matrix[j][i] = 0.0;
        }
    open_read(train, "face_list");
    Loop1i(num_classes * vectors_per_class)
```

```
        {
        fscanf(train, "%s", filename);
        open_read(protofile, filename)
        Loop1j(num_ftrs)
           {
           fscanf(protofile, "%f", &data_matrix[j][i]);
           }
        fclose(protofile);
        }
     fclose(train);
     }
  else if (type == 0) /***If voice verification***/
     {
     Loop1i(num_train)
        {
        Loop1j(num_ftrs)
           data_matrix_trans[i][j] = data_matrix[j][i] = 0.0;
        }
     Loop1i(num_train)
        Loop1j(num_ftrs)
           fscanf(datfile, "%f", &data_matrix[j][i]);
     fclose(datfile);
     }

  /***************Calculate average vector*******************/

  Loop1i(num_ftrs)
     {
     temp = 0.0;
     Loop1j(num_train)
        {
        temp += data_matrix[i][j];
        }
     average_temp[i] = temp/num_train;
     }

  /**************Subtract average vector************************/

  Loop1j(num_train)
     Loop1i(num_ftrs)
        data_matrix[i][j] = data_matrix[i][j] - average_temp[i];


  /*************Make transpose matrix************************/

  Loop1j(num_train)
     Loop1i(num_ftrs)
        data_matrix_trans[j][i] = data_matrix[i][j];

  /******************Matrix multiply data_matrix by itself***************/

  if (type == 1) /****If face verification***/
```

```
    {
    Loop1i(num_train)
        Loop1j(num_train)
            {
            temp = 0.0;
            Loop1k(num_ftrs)
                temp = temp + data_matrix_trans[i][k] * data_matrix[k][j];
            L[i][j] = temp;
            }
    }
else if (type == 0) /***If speaker verification***/
    {
    Loop1i(num_ftrs)
        Loop1j(num_ftrs)
            {
            temp = 0.0;
            Loop1k(num_train)
                temp = temp + data_matrix_trans[k][i] * data_matrix[j][k];
            L[i][j] = temp;
            }
    }

free_matrix(data_matrix_trans, 1, num_train, 1, num_ftrs);

/***************Do Jacobi rotation and sort eigenstuff***********/

if (type == 1) /***If face verification***/
    {
    jacobi(L, num_train, eig_values, eig_vectors, &nrot);
    eigsrt(eig_values, eig_vectors, num_train);
    }
else if (type == 0) /***If voice verification***/
    {
    jacobi(L, num_ftrs, eig_values, eig_vectors, &nrot);
    eigsrt(eig_values, eig_vectors, num_ftrs);
    }

if (type == 1)
    free_matrix(L, 1,num_train,1,num_train);
else if (type == 0)
    free_matrix(L, 1,num_ftrs,1,num_ftrs);

/***Can print out eigenvectors if desired by uncommenting here

open_write(outfile, outfilename);
fprintf(outfile, "%d\n", num_ftrs);
Loop1i(num_ftrs)
    {
    fprintf(outfile, "%f ", average_temp[i]);
    }
Loop1i(num_ftrs)
    {
```

```
   fprintf(outfile, "\n%f\n", eig_values[i]);
   Loop1j(num_ftrs)
      fprintf(outfile, "%f ", eig_vectors[j][i]);     }
fclose(outfile);
***End of print eigenvector section****/

free_vector(average_temp, 1, num_ftrs);
}
```

A.6.8 verify_face_net.c.

```
/******************************************************************
Program:      verify_face_net.c

Description:  This program performs face recognition. The program grabs an image of the person sitting
in front of the camera, processes that image, extracts the KLTcoefficients and finds the closest match from
the faces in the training set.

Author:   Kenneth Runyon

Date:      8 July 92 - 31 Aug 92

Modified by:   John G. Keller

Date:      1 Sep 93

Modification Description:   Added capability for accepting command line arguments.   Added command
line option for specifying use of the net built for the nonlinear transformation for testing here.

*******************************************************************/
#include <stdio.h>
#include "vfc_lib.h"
#include "jkmacros.h"
#include "globals.h"

#define   NUM_LAYRS   2
#define   WT_SED       1918940490
#define PART_SED   1191645590
#define   RNDM_SED   123456789
#define MAX_ITS       600
#define   OUT_INT       100
#define   ETA_IN      0.15
#define   ETA_OUT       0.3
#define ETA_1_2       0.0
#define   ALPHA       0.5
#define   BAT_SZ       1
#define   TRAIN_PCT   0.0
#define   NORM       1

int    dimension, j,
   num_coefs,
   num_train_faces,
    done,
    num_protos,
    happy,
    num_class;

FILE    *fparam,
    *fhandle,
    *fweights,
    *fset,
    *ftable;

char    wt_file[10],
    dat_file[10],
    hid_nodes,
    hid_nodes2,
    username[30],
    user1[30],
    answer[4],
    waste[2];

#define TRUE 1
#define FALSE 0



extern void center();
extern void gwind();
extern void ver_net_coefficients();

main(int argc, char *argv[])
{
FILE    *vprob, *fprob, *nonlindata;

char    command[30],
    user[30];

float    yesprob,
    yesoutput,
    nooutput,
    sum,
    yesvoice,
    novoice,
    yesvoiceprob,
    fusedprob,
    fl_waste;

int    i,
    PROCESS,
    NONLIN,
    arg_count,
    USE_FILE,
    type,
    num_not_person,
    SAME,
    int_waste;

#define SYNTAX "Usage: verify_face_net <claimed identity>\n                    ['filename'][noprocess][

if (argc == 1)
    {
     printf("\n%s\n", SYNTAX);
    exit(0);
    }

PROCESS = TRUE;
NONLIN = FALSE;
USE_FILE = FALSE;
arg_count = argc - 1;


if (arg_count == 1)
    {
    PROCESS = TRUE;
    NONLIN = FALSE;
    }

if (arg_count == 2)
    {
    if (strcmp(argv[2], "noprocess") == 0)
        {
        PROCESS = FALSE;
        NONLIN = FALSE;
        }
    else if (strcmp(argv[2], "nonlin") == 0)
        {
        PROCESS = TRUE;
        NONLIN = TRUE;
        }
    else if (fopen(argv[2], "r") == NULL)
        {
        printf("\nCan't open the file %s.\n", argv[2]);
        exit(0);
        }
    else USE_FILE = TRUE;
    }
if (arg_count == 3)
    {
    if ((strcmp(argv[2], "noprocess") == 0) || (strcmp(argv[3], "noprocess") == 0))
        {
        PROCESS = FALSE;
        }
    if ((strcmp(argv[2], "nonlin") == 0) || (strcmp(argv[3], "nonlin") == 0))
        {
        NONLIN = TRUE;
        }
    if ((fopen(argv[2], "r") == NULL) && (fopen(argv[3], "r") == NULL) && ((PROCESS == TRUE)
|| (NONLIN == FALSE)))
        {
        printf("\nCan't open either %s or %s.  Make sure you're using the proper syntax. \n%s",
argv[2], argv[3], SYNTAX);
        exit(0);
        }
     if ((PROCESS == FALSE && NONLIN == FALSE) || (PROCESS == TRUE && NONLIN ==
TRUE))
        {
        USE_FILE = TRUE;
        }
    }
if (arg_count == 4)
    {
    PROCESS = FALSE;
    NONLIN = TRUE;
    USE_FILE = TRUE;
    if (fopen(argv[2], "r") == NULL)
        {
        printf("\nCan't open file %s.\n", argv[2]);
        exit(0);
        }
    }


sprintf(user, "%s_f_klt.wts", argv[1]);

if ((ftable = fopen(user, "r")) == NULL)
        {
        printf("\nI can't open the file %s_f_klt.wts.\n", argv[1]);
        exit(0);
        }
    else
        fclose(ftable);


/*****************************************************
*   read the parameters from train_params file
*****************************************************/

open_read(fparam, "train_f_params");
fscanf(fparam,"%d",&dimension);
fscanf(fparam,"%d",&num_coefs);
fscanf(fparam,"%d",&num_train_faces);
fscanf(fparam,"%d",&num_protos);
fscanf(fparam,"%d",&num_class);
fclose(fparam);

/************ Create the setup file for the neural network **********/
```

```
sprintf(wt_file, "%s_f_klt.wts", argv[1]);
strcpy(dat_file,"klt_f.dat");
hid_nodes = 2 * num_coefs;
hid_nodes2 = 0;


fset = fopen("setup.mlp","w");
fprintf
(fset, "%d\n%d\n%d\n%d\n%s                    -store weights\n%d\n",NUM_LAYRS,WT_SED,PART_SED,RNDM_SED,wt_fil
MAX_ITS);
fprintf(fset,"%d %d %d %d\n",num_coefs,hid_nodes,hid_nodes2,num_class);
fprintf(fset,"%s                    -data\n%d\n%f\n%f\n%f\n%f\n%d\n%f\n%d\n",
dat_file,OUT_INT,ETA_IN,ETA_OUT,ETA_1_2,ALPHA,BAT_SZ,TRAIN_PCT,NORM);
fclose(fset);

/*************************************************************************
Either use an existing image file or grab a new one.  Can either grab
a single image or use the segmentation algorithm.
*************************************************************************/


if (fopen("user.gra", "r") ≠ NULL) system("rm user.gra");
happy = 0;
if (USE_FILE == TRUE)
   {
   sprintf(command, "cp %s user.gra", argv[2]);
   system(command);
   }
else
   {
   while (!happy)
      {
      autograb("user");
      sprintf(command, "display%d user.gra stay", SM_WIDTH);
      system(command);
      printf("\nIs the picture satisfactory (y/n)? ");
      gets(answer);
      if ((answer[0] == 'y') || (answer[0] == 'Y'))
         break;
      }
   }

if (PROCESS == TRUE)
   {
   center(dimension,"correlate.ref","user.gra");
   gwind(dimension,"user.gra");
   center(dimension,"wind.ref","user.gra");
   }
```

```
/****** create the data file and store the kl coefficients**********/

open_write(ftable, "waste");
open_write(fweights, "klt_f.dat");

fprintf(fweights,"%d\n%d\n",num_coefs,num_class);
sprintf(user, "user.gra");
type = 1;
if (NONLIN == FALSE)
   face_net_coefficients(dimension, num_coefs, user, fweights, ftable, num_class, type, type);
else
   face_net_coefficients(dimension, num_train_faces, user, fweights, ftable, num_class, type, type);
fclose(fweights);

if (NONLIN == TRUE)
   {
   open_read(fweights, "klt_f.dat");
   open_write(nonlindata, "facetest.dat");
   fprintf(nonlindata, "%d\n1\n1\n", num_train_faces);
   Loop1i(3)
      fscanf(fweights, "%d", &int_waste);
   Loop1j(num_train_faces) /***Num of features***/
      {
      fscanf(fweights, "%f ", &fl_waste);
      fprintf(nonlindata, "%f ", fl_waste);
      }
   fclose(fweights);
   fclose(nonlindata);
   system("xfeatures facetest.dat");
   open_read(nonlindata, "newtest.dat");
   open_write(fweights, "klt_f.dat");
   fscanf(nonlindata, "%d ", &int_waste);
   fprintf(fweights, "2\n2\n0 ");
   Loop1i(num_coefs)
      {
      fscanf(nonlindata, "%f ", &fl_waste);
      fprintf(fweights, "%f ", fl_waste);
      }
   fprintf(fweights, "0.90000 0.10000\n");

   fclose(nonlindata);
   fclose(fweights);
   }


/****** find the best matching training face ******/

#ifdef RESULTS
   system("mlp_fuse_file");

/******fuselist.c has the other file writing stuff.  The actual net
```

```
*******outputs are written to the file 'face_prob' by dkmain.c com-
*******piled as mlp_fuse_list*************************************/

#else
   system("mlp_fuse");
#endif

/******If not writing to a file, output to the screen***************/

#ifndef RESULTS
j = 0;
   while(argv[2][j] ≠ '\0')
      {
      if (isalpha(argv[2][j]))
         j++;
      else
         argv[2][j] = 0;
      }

   if (strcmp(argv[1], argv[2]) == 0)
      SAME = TRUE;
   else
      SAME = FALSE;



open_read(fprob, "node_out");
fscanf(fprob, "%f%f", &yesoutput, &nooutput);
fclose(fprob);
num_not_person = (num_train_faces/num_protos) - 1;
yesprob = (yesoutput * (num_not_person/2.0))/((yesoutput * num_not_person/2.0) + nooutput * 0.5);

printf("\n\nThe post-probability based on face that this is %s is %f.\n\n", argv[1], yesprob);

if (SAME == TRUE)
   printf("\nClaimed ID is true ID.  Verification probability is %f\n", yesprob);
else if (SAME == FALSE)
   printf("\nImposter.  Verification probability is %f\n", yesprob);


open_write(fprob, "face_prob");
fprintf(fprob, "%f", yesprob);
fclose(fprob);

#endif

/****** remove trash files *******/

/*system("rm test_coefs");
if (fopen("*.rle", "r") ≠ NULL) system("rm *.rle");
if (fopen("*.red", "r") ≠ NULL) system("rm *.red");
if (fopen("*.rec", "r") ≠ NULL) system("rm *.rec");  */
if (fopen("waste", "r") ≠ NULL) system("rm waste");

}
```

A.6.9 verify_voice_net.c.

```
/*****************************************************
Program:    verify_voice_net.c

Description:  This routine is based on verify_face_net, written by Ken Runyon.  It verifies the identity of
a speaker via a file, and can be modified to capture a speaker's voice live.  It also maintains the capability
of concatenating raw speaker cepstral vectors (to retain temporal information) or using codebook vectors
for verification.

Author:      John G. Keller

Date:     1 Sep 93
*****************************************************/
#include <stdio.h>
#include "vfc_lib.h"
#include "jkmacros.h"
#include "globals.h"

#define   NUM_LAYRS   2
#define   WT_SED       1918940490
#define PART_SED   1191645590
#define   RNDM_SED   123456789
#define MAX_ITS       1000
#define   OUT_INT       100
#define   ETA_IN      0.15
#define   ETA_OUT       0.3
#define ETA_1_2       0.0
#define   ALPHA       0.5
#define   BAT_SZ       1
#define   TRAIN_PCT   0.0
#define   NORM       1

int    length, j,
   num_coefs,
   num_speaker_vectors,
   done,
   num_protos,
   happy,
   num_in_chunk,
   num_chunks,
   num_class,
   total_num_coefs,
   num_not_person,
   temp;

FILE    *fparam,
   *fhandle,
    *fweights,
    *fset,
    *ftable,
    *handle,
    *fdat,
    *fdat2;

char    wt_file[10],
    dat_file[10],
    hid_nodes,
    hid_nodes2,
    username[30],
    user1[30],
    answer[4],
    waste[2];
float   flnumber, temp1, temp2;

#define TRUE 1
#define FALSE 0


main(int argc, char *argv[])
{
FILE    *vprob, *prob, *nonlindata;

char    command[30],
    user[30];

float   yesprob,
    yesoutput,
    nooutput,
    fusedprob,
    fl_waste;

int   i,
    USE_FILE,
    SAME,
    NONLIN,
    int_waste;

#define SYNTAX "Usage: verify_voice_net <claimed identity> ['filename'] ['nonlin']\n"

if (argc == 1 || argc > 4)
    {
     printf("\n%s\n", SYNTAX);
    exit(0);
    }

USE_FILE = FALSE;

if (argc == 3 || argc == 4)
    USE_FILE = TRUE;
```

```
if (argc == 4)
   NONLIN = TRUE;

sprintf(user, "%s_s_klt.wts", argv[1]);
open_read(ftable, user);

/***************************************************************************
Either use an existing speaker file or grab a new one.
***************************************************************************/

   /***If verifying from file, copy file to 'user.speech' ***/

if (USE_FILE == TRUE)
   {
   sprintf(command, "cp %s user.speech", argv[2]);
   system(command);
   }
else
   {
/***insert speaker capture routines here.  Put resultant speech in
user.speech***/
   }


/*****************************************************
*   read the parameters from train_s_params file
*****************************************************/

open_read(fparam, "train_s_params");
fscanf(fparam,"%d", &num_coefs);
fscanf(fparam,"%d",&num_class);
fscanf(fparam, "%d", &num_in_chunk);
fscanf(fparam, "%d", &num_not_person);
fclose(fparam);

open_read(fhandle, "user.speech");
fscanf(fhandle, "%d", &num_protos);
fscanf(fhandle,"%d",&length);
rewind(fhandle);

/****** create the data file and store the kl coefficients**********/

open_write(fweights, "klt.dat");

fprintf(fweights,"%d\n%d\n",num_coefs,num_class);
sprintf(user, "user.speech");
if (NONLIN ≠ TRUE )
   voice_net_coefficients(length, num_coefs, fweights, fhandle, num_class, num_protos);
else
   voice_net_coefficients(length, length, fweights, fhandle, num_class, num_protos);
fclose(fweights);
system("rm user.speech");
```

```
/************************************************************************
We now have the klt.dat file containing all the sets of klt coefficients
for this user.  We next will rewrite the klt.dat file to represent
concatenated "chunks" of vectors.  A chunk of '1' will be a single frame's
coefficients.
************************************************************************/

num_chunks = (int) (num_protos/num_in_chunk);
total_num_coefs = num_in_chunk * num_coefs;
open_read(fweights, "klt.dat");
open_write(handle, "temp.dat");

fscanf(fweights, "%d", &temp);
fprintf(handle, "%d\n", total_num_coefs);
fscanf(fweights, "%d", &temp);
fprintf(handle, "%d\n", temp);


Loop1i(num_chunks)
   {
   fprintf(handle, "%d ", i - 1);
   Loop1j(total_num_coefs)
      {
      fscanf(fweights, "%f", &flnumber);
      fprintf(handle, "%f ", flnumber);
      }
   fprintf(handle, "0.10000 0.10000\n");
   }
fclose(handle);
fclose(fweights);
system("mv temp.dat klt_s.dat");

if (NONLIN == TRUE)
   {
   open_read(fweights, "klt_s.dat");
   open_write(nonlindata, "spkrtest.dat");
   fprintf(nonlindata, "%d\n1\n64\n", length );
   Loop1i(3)
      fscanf(fweights, "%d", &int_waste);
   Loop1j(length) /***Num of features***/
      {
      fscanf(fweights, "%f ", &fl_waste);
      fprintf(nonlindata, "%f ", fl_waste);
      }
   fclose(fweights);
   fclose(nonlindata);
   system("xfeatures spkrtest.dat");
   open_read(nonlindata, "newtest.dat");
   open_write(fweights, "klt_s.dat");
   fscanf(nonlindata, "%d ", &int_waste);
   fprintf(fweights, "2\n2\n");
   Loop1j(64)
      {
      fprintf(fweights, "%d ", j - 1);
      Loop1i(num_coefs)
         {
         fscanf(nonlindata, "%f ", &fl_waste);
         fprintf(fweights, "%f ", fl_waste);
         }
      fprintf(fweights, "0.90000 0.10000\n");
      }
   fclose(nonlindata);
   fclose(fweights);
   }

/************ Create the setup file for the neural network **********/


sprintf(wt_file, "%s_s_klt.wts", argv[1]);
strcpy(dat_file,"klt_s.dat");
hid_nodes = 2 * total_num_coefs;
hid_nodes2 = 0;

fset = fopen("setup.mlp","w");
fprintf
(fset, "%d\n%d\n%d\n%d\n%s                    -store weights\n%d\n",NUM_LAYRS,WT_SED,PART_SED,RNDM_SED,wt_fil
MAX_ITS);
fprintf(fset,"%d %d %d %d\n",total_num_coefs,hid_nodes,hid_nodes2,num_class);
fprintf(fset,"%s                    -data\n%d\n%f\n%f\n%f\n%f\n%d\n%f\n%d\n",
dat_file,OUT_INT,ETA_IN,ETA_OUT,ETA_1_2,ALPHA,BAT_SZ,TRAIN_PCT,NORM);
fclose(fset);

/****** find the best matching training speaker ******/

if (fopen("node_out", "r") ≠ NULL) system("rm node_out");

#ifdef RESULTS
   system("mlp_fuse_file");

/******fuselist.c has the other file writing stuff. The actual net
*******outputs are written to the file 'face_prob' by dkmain.c com-
*******piled as mlp_fuse_list*************************************/

#else
   system("mlp_fuse");
#endif

/******If not writing to a file, output to the screen***************/

#ifndef RESULTS
j = 0;
   while(argv[2][j] ≠ '\0')
      {
```

```
      if (isalpha(argv[2][j]))
         j++;
      else
         argv[2][j] = 0;
      }

   if (strcmp(argv[1], argv[2]) == 0)
      SAME = TRUE;
   else
      SAME = FALSE;

open_read(vprob, "node_out");
sprintf(command, "%s_prob.dat", argv[1]);
if (SAME == TRUE) open_write(prob, command);
temp1 = temp2 = 0.0;
Loop1i(num_chunks)
   {
   fscanf(vprob, "%f%f", &yesoutput, &nooutput);
  if (SAME == TRUE) fprintf(prob, "%f\n", (yesoutput * (num_not_person/2.0))/((yesoutput * num_not_person/2.0)
+ nooutput * 0.5));
   temp1 += ( yesoutput * (num_not_person/2.0))/((yesoutput * num_not_person/2.0) + nooutput * 0.5);
   temp2 += 1.0 - ( yesoutput * (num_not_person/2.0))/((yesoutput * num_not_person/2.0) + nooutput
* 0.5);
   }
fclose(vprob);
if (SAME == TRUE) fclose(prob);
yesprob = temp1/(temp1 + temp2);
printf("\nThe post-probability based on voice that this is %s is %f.\n\n", argv[1],yesprob);
open_write(vprob, "voice_prob");
fprintf(vprob, "%f", yesprob);
fclose(vprob);

#endif

/****** remove trash files *******/

if (fopen("waste", "r") ≠ NULL) system("rm waste");

}
```

A.6.10 verify_identity.c.

```
/***************************************************************************
Program:   verify_identity.c

Description:   This program will invoke, in turn, the face verifier and the speaker verifier, and will then
fuse the results.

Author:      John G. Keller

Date:     1 Sep 93
***************************************************************************/
#include<stdio.h>
```

```
#include "jkmacros.h"

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

float    faceprob,
   voiceprob,
   identprob;

char    command[30];

FILE    *fhandle;

if (argc < 2 || argc == 3)
   {
   printf("\nSYNTAX:  verify_identity <claimed username> [<face file><voice file>]\n\n");
   exit(0);
   }

if (argc == 2) /***Capture face and voice live***/
   {
   sprintf(command, "verify_face_net %s", argv[1]);
   system(command);
   sprintf(command, "verify_voice_net %s", argv[1]);
   system(command);
   }
else
   {
/***********Make sure the files exist*********************/

   open_read(fhandle, argv[2]);
   open_read(fhandle, argv[3]);
   sprintf(command, "verify_face_net %s %s", argv[1], argv[2]);
   system(command);
   sprintf(command, "verify_voice_net %s %s", argv[1], argv[3]);
   system(command);
   }

/******************************************************************************
Get the face and voice probabilities resultant from running the
individual verifiers.  Fuse those probabilities.
******************************************************************************/

open_read(fhandle, "face_prob");
fscanf(fhandle, "%f", &faceprob);
fclose(fhandle);
open_read(fhandle, "voice_prob");
fscanf(fhandle, "%f", &voiceprob);
fclose(fhandle);

open_write(fhandle, "ident_prob");
for (i = 0;i ≤ 10;i++)
   {
   identprob = 0.1 * i * faceprob + (1 - (0.1 * i)) * voiceprob;
   fprintf(fhandle, "%f  %f\n\t%f\n",(0.1 * i), 1 - (0.1 * i), identprob);
    printf("Probability of verified identity based on %d%% face and %d%% voice = %f\n", 10 * i,
10 * (10 - i), identprob);
   }
fclose(fhandle);
}
```

A.6.11 *xnetpush.c.*

```
/************************************************************************
Program:   xnetpush.c

Description:   This program performs a non-linear transformation on multiple classes of input vectors.
The weight update rule causes the transformed classes to be pushed apart from each other, with the intent
of moving the classes into a reduced and more separable space than the one in which they began. The
weights are saved to a file, as are the final outputs from the net. These outputs are also put into multiple
files for plotting by Gnuplot.

Author:      John G. Keller

Date:     15 Oct 93
************************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include "jkmacros.h"

#define epoch 10

/*******Declare global variables and functions************/

int      num_prototypes,
       num_classes,
      num_features,
   num_out_nodes,
      num_hidden_nodes,
   output_type;

float  **data_matrix,
       **weight12,
       **weight23,
   *temp_hidden_out1,
   *temp_hidden_out2,
   *temp_out1,
   *temp_out2;


float  *vector(), **matrix(), free_vector(), free_matrix(), ran1();
```

```
/***************Begin main program****************/

main(int argc, char *argv[])
{
int   i,j,k, l,m, n,
   max_iterations,
   temp,
   random_pick1,
   random_pick2,
   random_seed,
   EXIT_SWITCH,
   OUT_OF_CLASS,
   max_vectors[5],
   class1,
   class2;


float

   **new_data_matrix,
   *hidden_node_out,
   *delta_out,
   *delta_hidden,
   *temp_total_dist,
   *sum_weight_out,
   *mean_feature,
   *variance_feature,
    *delta_out1,
    *delta_out2,
    *diff_out,
    *sum_out_delta1,
    *sum_out_delta2,
    *delta_hidden_out1,
    *delta_hidden_out2,
   eta_in,
   eta_out,
   temp1,
   temp_hold1,
   temp_hold2,
   temp_hold,
   sum_diff_out,
   norm_factor,
   total_error,
   min_distance,
   scale_factor,
   min_dist,
   **desired,
   *sum_diff,
   sum_weight;

char   command[30];
```

```
FILE  *netinfo, *data, *weightfile, *fhandle, *error_out, *error_in, *newtest;

if (argc ≠ 2)
  {
  printf("\nSYNTAX:  trainxnetbatch <datafile name>\n\n");
  exit(0);
  }

open_read(netinfo, "bpnet.dat");
fscanf(netinfo, "%d\n%d\n%f\n%f\n%d\n%d\n%d", &num_out_nodes, &num_hidden_nodes, &eta_in, &eta_out,
&max_iterations, &random_seed, &output_type);
fclose(netinfo);


random_seed = − random_seed;
open_read(data, argv[1]);
fscanf(data, "%d\n%d\n%d\n", &num_features, &num_classes, &num_prototypes);

num_hidden_nodes++;
num_features += 1; /***Account for augmentation***/

/*****************************************************************************
Declare and initialize matrices and vectors.
*****************************************************************************/

hidden_node_out = vector(1, num_hidden_nodes);
sum_weight_out = vector(1, num_hidden_nodes);
temp_total_dist = vector(1, num_out_nodes);
mean_feature = vector(1, num_features);
variance_feature = vector(1, num_features);
delta_out1 = vector(1, num_out_nodes);
delta_out2 = vector(1, num_out_nodes);
diff_out = vector(1, num_out_nodes);
sum_out_delta1 = vector(1, num_hidden_nodes);
sum_out_delta2 = vector(1, num_hidden_nodes);
delta_hidden_out1 = vector(1, num_hidden_nodes);
delta_hidden_out2 = vector(1, num_hidden_nodes);

weight12 = matrix(1, num_hidden_nodes, 1, num_features);
weight23 = matrix(1, num_out_nodes, 1, num_hidden_nodes);
data_matrix = matrix(1, num_features, 1, (num_prototypes * num_classes));
temp_out1 = vector(1, num_out_nodes);
temp_out2 = vector(1, num_out_nodes);
temp_hidden_out1 = vector(1, num_hidden_nodes);
temp_hidden_out2 = vector(1, num_hidden_nodes);

/*****************************************************************************
Load all the data into a single matrix.  Will be able to extract specific
class vectors later by keeping track of the indices.
*****************************************************************************/
```

```
Loop1i(num_prototypes * num_classes)
  {
  Loop1j(num_features - 1)
  fscanf(data, "%f", &data_matrix[j][i]);
  data_matrix[num_features][i] = 1.0;/***Augment each vector with 1.0***/
  }

/***************************************************************************
Normalize all the data across the features.
***************************************************************************/

normalize(data_matrix, mean_feature, variance_feature);

/***
printf("random_seed = %d\n", random_seed);
***/

/***************************************************************************
Initialize net weights. 'ran1' is a Numerical Recipes routine that returns
a random float between 0.0 and 1.0.
***************************************************************************/

Loop1ij(num_hidden_nodes, num_features)
  {
  weight12[i][j] = 1.0 * (ran1(&random_seed) - 0.5);
 }

Loop1ij(num_out_nodes, num_hidden_nodes)
  {
  weight23[i][j] = 1.0 * (ran1(&random_seed) - 0.5);
  }


   /*********open file for writing output error**********/
open_write(error_in, "error.inclass");
open_write(error_out, "error.outclass");


/***************************************************************************
This is the start of the main loop. Loop until we exceed m iterations.
***************************************************************************/
m = 0;
while (m < max_iterations)
   {

/***************************************************************************
Randomly pick two vectors from the data set and determine to which
classes they belong. If they are in the same class, we wish to push
the outputs together; if they're in different classes, we wish to
push the outputs apart. Set the variable "OUT_OF_CLASS" to indicate same
or different classes.
***************************************************************************/
```

```
   random_pick1 = (int)(ran1(&random_seed) * num_prototypes * num_classes);
  while (random_pick1 == 0) random_pick1 = (int)(ran1(&random_seed) * num_prototypes * num_classes);

   class1 = random_pick1/num_prototypes + 1;
label:
   random_pick2 = (int)(ran1(&random_seed) * num_prototypes * num_classes);
  while (random_pick2 == 0) random_pick2 = (int)(ran1(&random_seed) * num_prototypes * num_classes);

   class2 = random_pick2/num_prototypes + 1;

if (m < max_iterations/4 || m > 3 * max_iterations/4)
   {
   if (class1 == class2) goto label;
   }
else
   if (class1 ≠ class2) goto label;

/**********Compare the two classes and set the flag***************/

   if (class1 == class2) OUT_OF_CLASS = 0;
   else OUT_OF_CLASS = 1;



/***************************************************************************
Loop once through net for each vector.  Save the outputs of each node
for later calculation of the new weights.
***************************************************************************/

   /***First compute the hidden layer outputs for each vector***/


   compute_hidden_nodes(data_matrix, weight12, num_hidden_nodes, random_pick1, temp_hidden_out1);
   compute_hidden_nodes(data_matrix, weight12, num_hidden_nodes, random_pick2, temp_hidden_out2);



   /**********Now compute the output nodes**************/

     compute_output_nodes(weight23, temp_hidden_out1, temp_out1, num_out_nodes, num_hidden_nodes,
output_type);
     compute_output_nodes(weight23, temp_hidden_out2, temp_out2, num_out_nodes, num_hidden_nodes,
output_type);

temp1 = 0.0;
Loop1i(num_out_nodes)
   temp1 += sqr(temp_out1[i] - temp_out2[i]);

if (OUT_OF_CLASS == 1) fprintf(error_out, "%f\n", sqrt(temp1));
else if (OUT_OF_CLASS == 0) fprintf(error_in, "%f\n", sqrt(temp1));
```

```
/**********************Update the weights**************************/

   /********Build some terms for use by the update rule***********/


   Loop1i(num_out_nodes)
      {
      delta_out1[i] = temp_out1[i] * ( 1 - temp_out1[i]);
      delta_out2[i] = temp_out2[i] * ( 1 - temp_out2[i]);
      diff_out[i] = temp_out1[i] - temp_out2[i];
      }

   Loop1i(num_hidden_nodes)
      {
      sum_out_delta1[i] = sum_out_delta2[i] = 0.0;
      Loop1j(num_out_nodes)
         {
         sum_out_delta1[i] += diff_out[j] * weight23[j][i] * delta_out1[j];
         sum_out_delta2[i] += diff_out[j] * weight23[j][i] * delta_out2[j];
         delta_hidden_out1[i] = temp_hidden_out1[i] * ( 1 - temp_hidden_out1[i]);
         delta_hidden_out2[i] = temp_hidden_out2[i] * ( 1 - temp_hidden_out2[i]);
         }
      }

   /***********First update the output layer weights**********/

   Loop1i(num_hidden_nodes)
      Loop1j(num_out_nodes)
         {
         if (OUT_OF_CLASS == 1)
            {
             weight23[j][i] += eta_out * diff_out[j] *(delta_out1[j] * temp_hidden_out1[i] - delta_out2[j] *
temp_hidden_out2[i]);
            }
         else if (OUT_OF_CLASS == 0)
            {
             weight23[j][i] -= eta_out * diff_out[j] *(delta_out1[j] * temp_hidden_out1[i] - delta_out2[j] *
temp_hidden_out2[i]);
            }
         }

   /***********Then update the hidden layer weights***************/

   Loop1i(num_features)
      {
      Loop1j(num_hidden_nodes)
         {
         if (OUT_OF_CLASS == 1)
            {
```

```
            weight12[j][i] += eta_in * sum_out_delta1[j] * delta_hidden_out1[j] * data_matrix[i][random_pick1]
    - sum_out_delta2[j] * delta_hidden_out2[j] * data_matrix[i][random_pick2];
              }
            else if (OUT_OF_CLASS == 0)
              {
            weight12[j][i] -= eta_in * sum_out_delta1[j] * delta_hidden_out1[j] * data_matrix[i][random_pick1]
    - sum_out_delta2[j] * delta_hidden_out2[j] * data_matrix[i][random_pick2];
              }
            }
        }
m++;
if (m ≥ max_iterations) break;

   }
fclose(error_in);
fclose(error_out);

/*****************************************************************************
Save the weights to a file.
*****************************************************************************/

open_write(newtest, "newtest.dat");
        Loop1k(num_prototypes * num_classes)
          {
   /**********Compute the output of the hidden nodes************/

          compute_hidden_nodes(data_matrix, weight12, num_hidden_nodes, k, temp_hidden_out1);

   /**********Now compute the output nodes**************/

        compute_output_nodes(weight23, temp_hidden_out1, temp_out1, num_out_nodes, num_hidden_nodes,
output_type);
          Loop1i(num_out_nodes)
             {
             fprintf(newtest, "%f ", temp_out1[i]);
             }
          fprintf(newtest, "\n");
          }
fclose(newtest);
open_write(weightfile, "bpnet.wts");
fprintf(weightfile, "%d\n%d\n%d\n%d\n", num_features, num_hidden_nodes, num_out_nodes, output_type);
Loop1i(num_features - 1)
  fprintf(weightfile, "%f\n%f\n", mean_feature[i], variance_feature[i]);
Loop1i(num_hidden_nodes)
  {
  Loop1j(num_features)
  fprintf(weightfile, "%f ", weight12[i][j]);
  fprintf(weightfile, "\n");
  }
Loop1i(num_out_nodes)
  {
  Loop1j(num_hidden_nodes)
  fprintf(weightfile, "%f ", weight23[i][j]);
  fprintf(weightfile, "\n");
  }
fclose(weightfile);

printf("\nTotal epochs:  %d\n\n", m);

sprintf(command, "xfeatures %s", argv[1]);
/***system(command);***/
sprintf(command, "gnudata newtest.dat %d", num_classes);
system(command);


/****************************************************************************
Free memory from matrices and vectors.
****************************************************************************/

free_vector(hidden_node_out, 1, num_hidden_nodes);
free_vector(sum_weight_out, 1, num_hidden_nodes);
free_vector(temp_total_dist, 1, num_out_nodes);
free_vector(mean_feature, 1, num_features);
free_vector(variance_feature, 1, num_features);
free_vector(delta_out1, 1, num_out_nodes);
free_vector(delta_out2, 1, num_out_nodes);
free_vector(delta_hidden_out1, 1, num_hidden_nodes);
free_vector(delta_hidden_out2, 1, num_hidden_nodes);
free_vector(diff_out, 1, num_out_nodes);
free_vector(sum_out_delta1, 1, num_hidden_nodes);
free_vector(sum_out_delta2, 1, num_hidden_nodes);

free_matrix(weight12, 1, num_hidden_nodes, 1, num_features);
free_matrix(weight23, 1, num_out_nodes, 1, num_hidden_nodes);
free_matrix(data_matrix, 1, (num_prototypes * num_classes), 1, num_features);
free_matrix(temp_out1, 1, num_out_nodes);
free_matrix(temp_out2, 1, num_out_nodes);
free_matrix(temp_hidden_out1, 1, num_hidden_nodes);
free_matrix(temp_hidden_out2, 1, num_hidden_nodes);
}



normalize(float **data_matrix, float *mean_feature, float *variance_feature)
{
int   i,j;

float   sum,
     *sum_features,
     *sum_var;

FILE   *fhandle;

sum_features = vector(1, num_features);
```

```
sum_var = vector(1, num_features);

Loop1i(num_features - 1)
  {
  sum_features[i] = 0.0;
  Loop1j(num_prototypes * num_classes)
  {
  sum_features[i] += data_matrix[i][j];
  }
  }
Loop1i(num_features - 1)
  mean_feature[i] = sum_features[i]/(num_prototypes * num_classes);

Loop1i(num_features - 1)
  {
  sum_var[i] = 0.0;
  Loop1j(num_prototypes * num_classes)
  {
  sum_var[i] +=sqr(data_matrix[i][j] - mean_feature[i]);
  }
  }
Loop1i(num_features - 1)
  variance_feature[i] = sum_var[i]/(num_prototypes * num_classes);

Loop1i(num_features - 1)
  Loop1j(num_prototypes * num_classes)
  data_matrix[i][j] = (data_matrix[i][j] - mean_feature[i])/variance_feature[i];
open_write(fhandle, "normfile.dat");
Loop1i(num_prototypes * num_classes)
  {
  Loop1j(num_features - 1)
  fprintf(fhandle, "%f   ", data_matrix[j][i]);
  fprintf(fhandle, "\n");
  }
fclose(fhandle);

free_vector(sum_features, 1, num_features);
free_vector(sum_var, 1, num_features);

}


compute_hidden_nodes(float **data_matrix, float **weight12, int num_hidden_nodes, int vector, float *temp_hidden_out)
{

int   i,j;

float   temp_hold1;

Loop1i(num_hidden_nodes - 1)
    {
    temp_hold1 = 0.0;
```

```
      Loop1j(num_features)
        {
        temp_hold1 += weight12[i][j] * data_matrix[j][vector];
        }
      temp_hidden_out[i] = 1.0/(1.0 + exp(-temp_hold1));
      }
    temp_hidden_out[num_hidden_nodes] = 1.0;/***Account for augmentation***/

}



compute_output_nodes(float **weight23, float *temp_hidden_out, float *temp_out, int num_out_nodes, int
num_hidden_nodes, int output_type)
{

int   i,j;

float   temp_hold1;


Loop1i(num_out_nodes)
   {
   temp_hold1 = 0.0;
   Loop1j(num_hidden_nodes - 1)
      {
      temp_hold1 += weight23[i][j] * temp_hidden_out[j];
      }
   if (output_type == 0)  /***If linear output***/
      {
      temp_out[i] = temp_hold1;
      }
   else if (output_type == 1) /***If nonlinear***/
      {
      temp_out[i] = 1.0/(1.0 + exp(-temp_hold1));
      }
   }
}
```

Appendix B. Source Data

This appendix contains the source data for the performance figures given in Chapter 4. Tables B.1 through B.8 give the data for face verification accuracy, and Tables B.9 through B.16 provide the data for speaker verification.

Table B.1 Face verification accuracy when the subject's claimed identity is his true identity (two dimensions).

Claimed Identity is the True Identity (True Accept Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	100.0 %	100.0 %
dprescot	100.0 %	100.0 %	20.0 %
eingham	100.0 %	100.0 %	80.0 %
jcossent	100.0 %	100.0 %	80.0 %
jkeller	100.0 %	100.0 %	100.0 %
jmiller	100.0 %	100.0 %	100.0 %

Table B.2 Face verification accuracy when the subject's claimed identity is **not** his true identity (two dimensions).

Claimed Identity is Not True Identity(True Reject Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	68.0 %	20.0 %
dprescot	84.0 %	100.0 %	40.0 %
eingham	96.0 %	80.0 %	64.0 %
jcossent	60.0 %	88.0 %	100.0 %
jkeller	20.0 %	68.0 %	16.0 %
jmiller	96.0 %	60.0 %	20.0 %

Table B.3 Face verification accuracy when the subject's claimed identity is his true identity (four dimensions).

Claimed Identity is the True Identity (True Accept Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	100.0 %	100.0 %
dprescot	100.0 %	100.0 %	80.0 %
eingham	100.0 %	100.0 %	80.0 %
jcossent	100.0 %	100.0 %	40.0 %
jkeller	100.0 %	100.0 %	100.0 %
jmiller	100.0 %	100.0 %	100.0 %

Table B.4 Face verification accuracy when the subject's claimed identity is **not** his true identity (four dimensions).

Claimed Identity is Not True Identity (True Reject Accuracy			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	92.0 %	8.0 %
dprescot	100.0 %	100.0 %	64.0 %
eingham	100.0 %	80.0 %	72.0 %
jcossent	96.0 %	100.0 %	96.0 %
jkeller	64.0 %	80.0 %	48.0 %
jmiller	88.0 %	60.0 %	8.0 %

Table B.5 Face verification accuracy when the subject's claimed identity is his true identity (six dimensions).

Claimed Identity is the True Identity (True Accept Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	100.0 %	80.0 %
dprescot	100.0 %	100.0 %	100.0 %
eingham	100.0 %	100.0 %	80.0 %
jcossent	100.0 %	100.0 %	100.0 %
jkeller	100.0 %	100.0 %	0.0 %
jmiller	100.0 %	100.0 %	100.0 %

Table B.6 Face verification accuracy when the subject's claimed identity is **not** his true identity (six dimensions).

Claimed Identity is Not True Identity (True Reject Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	100.0 %	80.0 %
dprescot	100.0 %	100.0 %	80.0 %
eingham	100.0 %	100.0 %	16.0 %
jcossent	100.0 %	100.0 %	60.0 %
jkeller	80.0 %	88.0 %	72.0 %
jmiller	84.0 %	88.0 %	88.0 %

Table B.7 Face verification accuracy when the subject's claimed identity is his true identity (eight dimensions).

Claimed Identity is the True Identity (True Accept Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	100.0 %	100.0 %
dprescot	100.0 %	100.0 %	100.0 %
eingham	100.0 %	100.0 %	80.0 %
jcossent	100.0 %	100.0 %	40.0 %
jkeller	100.0 %	100.0 %	100.0 %
jmiller	100.0 %	100.0 %	100.0 %

Table B.8 Face verification accuracy when the subject's claimed identity is **not** his true identity (eight dimensions).

Claimed Identity is Not True Identity (True Reject Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	100.0 %	28.0 %
dprescot	96.0 %	100.0 %	88.0 %
eingham	96.0 %	96.0 %	44.0 %
jcossent	100.0 %	96.0 %	72.0 %
jkeller	88.0 %	84.0 %	44.0 %
jmiller	100.0 %	100.0 %	88.0 %

Table B.9 Speaker verification accuracy when the subject's claimed identity is his true identity (four dimensions).

Claimed Identity is the True Identity (True Accept Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	25.0 %	50.0 %	0.0 %
dprescot	100.0 %	100.0 %	100.0 %
jkeller	0.0 %	0.0 %	0.0 %
jmiller	100.0 %	100.0 %	0.0 %
jtreleav	0.0 %	100.0 %	100.0 %
kmccrae	50.0 %	0.0 %	100.0 %
mchin	75.0 %	25.0 %	25.0 %
rmacdona	0.0 %	0.0 %	100.0 %
wgool	0.0 %	100.0 %	100.0 %

Table B.10 Speaker verification accuracy when the subject's claimed identity is **not** his true identity (four dimensions).

Claimed Identity is Not True Identity (True Reject Accuracy			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	59.4 %	84.4 %	78.1 %
dprescot	46.9 %	84.4 %	81.3 %
jkeller	93.8 %	65.6 %	100.0 %
jmiller	31.3 %	0.0 %	53.1 %
jtreleav	84.4 %	25.0 %	28.1 %
kmccrae	34.4 %	71.9 %	56.3 %
mchin	40.6 %	87.5 %	78.1 %
rmacdona	68.8 %	81.3 %	37.5 %
wgool	62.5 %	25.0 %	28.1 %

Table B.11 Speaker verification accuracy when the subject's claimed identity is his true identity (six dimensions).

Claimed Identity is the True Identity (True Accept Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	50.0 %	0.0 %	0.0 %
dprescot	100.0 %	100.0 %	100.0 %
jkeller	100.0 %	0.0 %	75.0 %
jmiller	100.0 %	50.0 %	0.0 %
jtreleav	100.0 %	100.0 %	100.0 %
kmccrae	100.0 %	25.0 %	100.0 %
mchin	50.0 %	0.0 %	25.0 %
rmacdona	0.0 %	75.0 %	50.0 %
wgool	100.0 %	0.0 %	100.0 %

Table B.12 Speaker verification accuracy when the subject's claimed identity is **not** his true identity (six dimensions).

Claimed Identity is Not True Identity (True Reject Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	87.5 %	31.3 %	87.5 %
dprescot	46.9 %	87.5 %	87.5 %
jkeller	50.0 %	78.1 %	87.5 %
jmiller	21.9 %	59.4 %	28.1 %
jtreleav	43.8 %	31.3 %	15.6 %
kmccrae	81.3 %	37.5 %	65.6 %
mchin	96.9 %	93.8 %	68.8 %
rmacdona	84.4 %	37.5 %	50.0 %
wgool	40.6 %	37.5 %	31.3 %

Table B.13 Speaker verification accuracy when the subject's claimed identity is his true identity (eight dimensions).

Claimed Identity is the True Identity (True Accept Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	50.0 %	75.0 %	0.0 %
dprescot	75.0 %	100.0 %	100.0 %
jkeller	100.0 %	50.0 %	50.0 %
jmiller	100.0 %	100.0 %	25.0 %
jtreleav	100.0 %	100.0 %	100.0 %
kmccrae	100.0 %	0.0 %	100.0 %
mchin	75.0 %	0.0 %	25.0 %
rmacdona	0.0 %	0.0 %	100.0 %
wgool	100.0 %	100.0 %	100.0 %

Table B.14 Speaker verification accuracy when the subject's claimed identity is **not** his true identity (eight dimensions).

Claimed Identity is Not True Identity (True Reject Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	78.1 %	78.1 %	68.8 %
dprescot	34.4 %	81.3 %	78.1 %
jkeller	53.1 %	93.8 %	93.8 %
jmiller	40.6 %	12.5 %	28.1 %
jtreleav	28.1 %	28.1 %	25.0 %
kmccrae	75.0 %	65.6 %	31.3 %
mchin	100.0 %	93.8 %	68.8 %
rmacdona	81.3 %	81.3 %	37.5 %
wgool	40.6 %	31.3 %	37.5 %

Table B.15 Speaker verification accuracy when the subject's claimed identity is his true identity (ten dimensions).

Claimed Identity is the True Identity (True Accept Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	100.0 %	50.0 %	25.0 %
dprescot	100.0 %	75.0 %	100.0 %
jkeller	100.0 %	0.0 %	25.0 %
jmiller	100.0 %	0.0 %	100.0 %
jtreleav	100.0 %	100.0 %	100.0 %
kmccrae	100.0 %	100.0 %	100.0 %
mchin	100.0 %	0.0 %	25.0 %
rmacdona	75.0 %	0.0 %	75.0 %
wgool	25.0 %	100.0 %	100.0 %

Table B.16 Speaker verification accuracy when the subject's claimed identity is **not** his true identity (ten dimensions).

Claimed Identity is Not True Identity (True Reject Accuracy)			
Claimed ID	*Min_error*	*Eigenvalue*	*FoM*
cmartin	53.1 %	75.0 %	68.8 %
dprescot	50.0 %	87.5 %	84.4 %
jkeller	59.4 %	93.8 %	87.5 %
jmiller	78.1 %	78.1 %	37.5 %
jtreleav	81.3 %	25.0 %	25.0 %
kmccrae	87.5 %	18.8 %	37.5 %
mchin	75.0 %	93.8 %	68.8 %
rmacdona	65.6 %	62.5 %	34.4 %
wgool	78.1 %	25.0 %	34.4 %

Vita

Captain John G. Keller was born on 14 Sep 1957 in Moses Lake, Washington. He enlisted in the United States Air Force in 1981, and upon completion of Basic Training and Technical Training School was assigned to the 2167 Communication Squadron in Kalkar, West Germany, as a Ground Radio electronics technician. In 1984 he was assigned to the Joint Communication Support Element at MacDill Air Force Base, Florida, as a Ground Radio technician/operator with the Joint Airborne Communication Center/Command Post ("Jackpot"). Captain Keller was accepted into the Airman Education and Commissioning Program in 1987, and completed his Bachelor of Science degree in Electrical Engineering at the University of Florida in May 1989. After completion of Officer Training School in August 1989, he was commissioned as an officer in the United States Air Force and was assigned to the 552 Airborne and Control Wing, Tinker Air Force Base, Oklahoma, as an E-3 project management engineer. In 1991 he was assigned to the Oklahoma City Air Logistics Center (also at Tinker Air Force Base) as a project management engineer for the E-3 System Program Manager, and during this tour received his Master of Arts degree in Computer Resource Management from Webster University. In May of 1992 he entered into the School of Engineering, Air Force Institute of Technology, at Wright-Patterson Air Force Base, Ohio, to pursue a Master of Science degree in Electrical Engineering.

Permanent address: 3423 Calumet Drive
Orlando Florida 32810

Bibliography

1. Atal, Bishnu S. "Automatic Recognition of Speakers From their Voices." *Proceedings of the IEEE64(4)*. 460–475. 1976.
2. Atal, B.S. "Effectiveness of linear prediction characteristics of the speech wave for automatic speaker identification and verification," *Journal of the Acoustic Society of America*, *50(2)*:1304–1312 (June 1974).
3. Bennani, Younes and Patrick Gallinari. "On the Use of TDNN-Extracted Features Information in Talker Identification." *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. 385–388. 1991.
4. Bennani, Younes et. al. "A Connectionist Approach for Automatic Speaker Identification." *Proceedings of the 1990 International Conference on Acoustics, Speech and Signal Processing*. 265–268. 1990.
5. Bouattour, et al. "Neural Nets for Human Face Recognition," *IEEE IJCNN*, *III*:700–704 (June 1992).
6. Burns, Thomas J. *A Non-Homogeneous Wavelet Multiresolution Analysis and its Application to the Analyis of Motion*. PhD dissertation, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1993.
7. Chair, Z. and P. K. Varshney. "Distributed Bayesian Hypothesis Testing with Distributed Data Fusion," *IEEE Transactions on Systems, Man, and Cybernetics*, *18:5*:695–699 (1988).
8. Chui, Charles K. *An Introduction to Wavelets*. Boston, MA: Academic Press, Inc., 1992.
9. Colombi, John et al. "Multiple Day User Recognition Using Speech and Faces," *Unpublished* (1992).
10. Colombi, John M. *Cepstral and Auditory Model Features for Speaker Recognition*. MS thesis, AFIT/GE/ENG/92D-11, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.
11. Cottrell, Garrison W. and Janet Metcalfe. *EMPATH: Face, Emotion and Gender Recognition Using Holons*. San Mateo, CA: Morgan Kauffmann Publishers, Inc, 1991.
12. Cybenko, G. "Approximations by Superpositions of Sigmoidal Functions," *Mathematical Controls, Signals, and Systems* (1989).
13. Davis, Steven B. and Paul Mermelstein. "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," *IEEE Transactions ASSP*, *28(4)*:357–366 (August 1980).
14. Fleming, Michael K. and Garrison W. Cottrell. "Categorization of Faces Using Unsupervised Feature Extraction." *IEEE International Joint Conference on Neural Networks2*. 65–70. 1990.
15. Furui, Sadaoki. "Cepstral Analysis Techniques for Automatic Speaker Verification," *IEEE Transactions ASSP*, *29(2)*:254–272 (April 1981).

16. Gaganelis, D.A. and E.D. Frangoulis. "A novel approach to Speaker Verification." *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. 373–376. 1991.

17. Gay, Kevin P. *Autonomous Face Segmentation.* MS thesis, AFIT/GE/ENG/92S-06, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1992.

18. Goble, James R. *Face Recognition using the Discrete Cosine Transform.* MS thesis, AFIT/GE/ENG/91D-21, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1991.

19. Goble, James R. et al. "A Facial Feature Communications Interface for the Non-Vocal," *IEEE Engineering in Medicine and Biology*, 1–4 (September 1993).

20. Gordon, Gaile G. "Face Recognition Based on Depth Maps and Surface Curvature," *SPIE, Geometric Methods in Computer Vision*, *1570*:234–247 (1991).

21. Govindaraju, et al. "Locating Human Faces in Newspaper Photographs," *IEEE Computer Vision and Pattern Recognition*, 549–554 (1989).

22. Hattori, Hiraoki. "Text-independent Speaker recognition using Neural Networks." *Proceedings of the 1992 International Conference on Acoustics, Speech and Signal Processing*. 153–156. 1992.

23. Higgins, A. L. and L. G. Bahler. "Text-Independent Speaker Verification By Discriminator Counting." *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. 405–408. 1991.

24. Hush, Don R. and Bill G. Horne. "Progress in Supervised Neural Networks: What's New Since Lippmann?," *IEEE Signal Processing Magazine*, 8–39 (January 1993).

25. Jia, Xiaoguang and Mark S. Nixon. "Profile feature extraction via the Walsh transform for face recognition," *SPIE, Intelligent Robots and Computer Vision*, *1825*:46–52 (1992).

26. Kao, Yu-Hang, et al. "Free-Text Identification over long distance telephone channel using hypothesized phonetic segmentation." *Proceeedings of the 1992 International Conference on Acoustics, Speech and Signal Processing*. 177–180. 1992.

27. Krepp, Dennis L. *Face Recognition with Neural Networks.* MS thesis, AFIT/GE/ENG/GE-92D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.

28. Krzysztofowicz, Roman and Dou Long. "Fusion of Detection Probabilities and Comparison of Multisensor Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, *20:3*:665–677 (June 1990).

29. Levinson, Stephen E. and David B. Roe. "A Perspective on Speaker Recognition," *IEEE Communications Magazine*, 28–34 (January 1990).

30. Linde, Y. et al. "An Algorithm for Vector Quantizer Design," *IEEE Transactions on Communication*, *28*:84–94 (Jan 1980).

31. Liu, W. et al. "Study of Line Spectrum Pair Frequencies for Speaker Recognition." *Proceeedings of the 1990 International Conference on Acoustics, Speech and Signal Processing*. 277–280. 1990.

32. MacDonald, Robert P. *Optical Wavelet Transform for Fingerprint Identification*. MS thesis, AFIT/GE/ENG/GEO-93D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1993.

33. Manjunath, BS. "A Feature Based Approach to Face Recognition," *IEEE Computer Vision and Pattern Recogntion*, 373–378 (1992).

34. Mannaert, Herwig and Andre Oosterlinck. "Self-organizing System for Analysis and Identification of Human Faces," *SPIE, Applications of Digital Image Processing XIII*, *1349*:227–232 (1990).

35. Matsui, Tomoko and Sadaoki Furui. "A Text-Independent Speaker Recognition Method Robust Against Utterance Variations." *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. 377–380. 1991.

36. McCrae, Kimberley A. *Multispectral Image Processing*. MS thesis, AFIT/GE/ENG/GEO-93D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1993.

37. Oglesby, J. and J.S. Mason. "Optimisation of Neural Models for Speaker Identification." *Proceedings of the 1990 International Conference on Acoustics, Speech and Signal Processing*. 261–264. 1990.

38. Oglesby, J. and J.S. Mason. "Radial Basis Function Networks for Speaker Recognition." *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. 393–396. 1991.

39. Parsons, Thomas W. *Voice and Speech Processing*. New York, NY: McGraw-Hill, Inc., 1987.

40. Poritz, Alan B. "Linear Prediction of Hidden Markov Models." *Proceedings of the 1982 International Conference on Acoustics, Speech and Signal Processing*. 1291–1294. 1982.

41. Prescott, D. Neale. *Speaker Identification Using Data Fusion with Auditory Models*. MS thesis, AFIT/GE/ENG/GE-94M, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1994.

42. Prokoski, Francine J., et al. "Identification of Individuals by Means of Facial Thermography," *International Carnahan Conference on Security Technology: Crime Countermeasures*, 120–125 (1992).

43. Riebman, A. R. and L. W. Nolte. "Design and Performance Comparison of Distributed Detection Networks," *IEEE Transactions on Aerospace Electronics Systems*, *23:6*:789–797 (1987).

44. Riebman, A. R. and L. W. Nolte. "Optimal Detection and Performance of Distributed Sensor Systems," *IEEE Transactions on Aerospace Electronics Systems*, *23:1*:24–30 (1987).

45. Rogers, Steven K. and Matthew Kabrisky. *An Introduction to Biological and Artificial Neural Networks*. Bellingham, Washington: SPIE Optical Engineering Press, 1991.

46. Rose, Richard C. and Douglas A. Reynolds. "Text Independent Speaker Identification Using Automatic Acoustic Segmentation." *Proceedings of the 1990 International Conference on Acoustics, Speech and Signal Processing*. 293–296. 1990.

47. Rose, Richard C. et al. "Robust Speaker Identification in Noisy Environments Using Noise Adaptive Speaker Models." *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. 401–404. 1991.

48. Rosenburg, Aaron E., et al. "Sub-Word Unit Talker Verification Using Hidden Markov Models." *Proceedings of the 1990 International Conference on Acoustics, Speech and Signal Processing*. 269–272. 1990.

49. Rosenburg, Aaron E., et al. "Connected Word Talker Verification Using Whole Word Markov Models." *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. 381–384. 1991.

50. Ruck, Dennis et al. "Characterization of Multilayer Perceptrons for Target Recognition." *Proceedings of the Sixth Annual Conference on Aerospace Applications of Artificial Intelligence*. October 1990.

51. Rudasi, Laszlo and Stephen A. Zahorian. "Text-Independent Talker Identification With Neural Networks." *Proceedings of the 1991 International Conference on Acoustics, Speech and Signal Processing*. 389–392. 1991.

52. Runyon, Kenneth R. *Automated Face Recognition System*. MS thesis, AFIT/GE/ENG/GE-92D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.

53. Samal, Ashok and Prasana A. Iyengar. "Automatic Recognition and Analysis of Human Faces and Facial Expressions: A Survey," *Pattern Recognition*, *25*:65–77 (1992).

54. Savic, M. and J. Sorenson. "Phoneme Based Speaker Verification." *Proceedings of the 1992 International Conference on Acoustics, Speech and Signal Processing*. 165–168. 1992.

55. Savic, Michael and Sunil K. Gupta. "Variable Parameter Speaker Verification System based on Hidden Markov Models." *Proceedings of the 1990 International Conference on Acoustics, Speech and Signal Processing*. 281–284. 1990.

56. Schwartz, T. and others. "The application of Probability Density Estimation to Text-independent Speaker Identification." *Proceedings of the 1982 International Conference on Acoustics, Speech and Signal Processing*. 1649–1652. 1982.

57. Seitz, Peter and Martin Bichsel. "The digital doorkeeper-Automatic face recognition with the computer," *IEEE International Carnahan Conference on Security Technology*, 77–83 (1991).

58. Shartle, Gary F. *Handwritten Word Recognition Based on Fourier Coefficients*. MS thesis, AFIT/GE/ENG/GEO-93D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1993.

59. Soong, et al. "A Vector Quantization Approach to Speaker Recognition." *Proceedings of the 1985 International Conference on Acoustics, Speech and Signal Processing*. 387–390. 1985.

60. Soong, Frank K. and Aaron E. Rosenburg. "On the Use of Instantaneous and Transitional Spectral Information in Speaker Recognition," *IEEE Transactions ASSP*, *36(6)*:871–879 (June 1988).

61. Suarez, Pedro. *Face Recognition with the Karhunen-Loève Transform*. MS thesis, AFIT/GE/ENG/GE-91D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.

62. Thomopoulos, Stelios C. A. "Sensor Integration and Data Fusion," *SPIE Sensor Fusion II: Human and Machine Strategies*, *1198*:178–191 (1989).

63. Tou, Julius T. and Rafael C. Gonzalez. *Pattern Recognition Principles*. Reading, MA: Addison-Wesley Publishing, 1974.

64. Tseng, Belle, et al. "Continuous Probabilistic Acoustic Map for Speaker Recognition." *Proceedings of the 1992 International Conference on Acoustics, Speech and Signal Processing*. 161–164. 1992.

65. Turk, Matthew A. and Alex P. Pentland. "Recognition in Face Space," *SPIE, Intelligent Robots and Computer Vision IX*, *1381*:43–54 (1990).

66. Xu, Lei, et al. "Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, *22(3)*:418–435 (1992).

59. Soong, et al. "A Vector Quantization Approach to Speaker Recognition." *Proceedings of the 1985 International Conference on Acoustics, Speech and Signal Processing*. 387–390. 1985.

60. Soong, Frank K. and Aaron E. Rosenburg. "On the Use of Instantaneous and Transitional Spectral Information in Speaker Recognition," *IEEE Transactions ASSP*, *36(6)*:871–879 (June 1988).

61. Suarez, Pedro. *Face Recognition with the Karhunen-Loève Transform*. MS thesis, AFIT/GE/ENG/GE-91D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1991.

62. Thomopoulos, Stelios C. A. "Sensor Integration and Data Fusion," *SPIE Sensor Fusion II: Human and Machine Strategies*, *1198*:178–191 (1989).

63. Tou, Julius T. and Rafael C. Gonzalez. *Pattern Recognition Principles*. Reading, MA: Addison-Wesley Publishing, 1974.

64. Tseng, Belle, et al. "Continuous Probabilistic Acoustic Map for Speaker Recognition." *Proceedings of the 1992 International Conference on Acoustics, Speech and Signal Processing*. 161–164. 1992.

65. Turk, Matthew A. and Alex P. Pentland. "Recognition in Face Space," *SPIE, Intelligent Robots and Computer Vision IX*, *1381*:43–54 (1990).

66. Xu, Lei, et al. "Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, *22(3)*:418–435 (1992).

REPORT DOCUM[illegible]

1. AGENCY USE [illegible]

December 1993 Master's Thesis

4. TITLE AND SUBTITLE

IDENTITY VERIFICATION THROUGH
THE FUSION OF FACE
AND SPEAKER RECOGNITION

6. AUTHOR(S)

John G. Keller

7. PERFORMING [illegible]

Air Force Institute of Technology, WPAFB OH 45433-6583

AFIT/GE/ENG/93D-20

9. SPONSORING [illegible]

Lt Col Rodney Winter
NSA/R221
Fort Meade, MD 20755-6000

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION [illegible]

Distribution Unlimited

13. ABSTRACT [illegible]

Abstract

In this research, face recognition and speaker identification systems are each converted into verification systems. The two verification systems are then fused to form a single identity verification system. Finally, the use of the Karhunen-Loéve Transform (KLT) for dimensional reduction is examined for suitability in the verification task.

The base face recognition system used the KLT for feature reduction and a back-propagation neural net for classification. Verification involved training a net for each individual in the database for two classes of outputs, 'Joe' or 'not Joe.' The base speaker identification system used Cepstral analysis for feature extraction and a distortion measure for classification. Verification in this case involved performing the KLT on the Cepstral coefficients and then classifying using a two-class neural net for each individual, similarly to the face verifier implementation.

KLT feature reduction is compared to alternative linear and non-linear methods, and the KLT is found to provide superior performance. The fusion of the two base verification systems is shown to provide superior performance over either system alone.

14. SUBJECT [illegible]

Pattern Recognition, Recognition, Verification, Biometry

185

17. SECURITY [illegible]
OF REPORT

UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN [illegible]