(1)

AUTONOMOUS FACE RECOGNITION MACHINE
USING A FOURIER FEATURE SET

THESIS

Barbara C. Robb, B.S.
Captain, USAF

AFIT/GE/ENG/89D-44

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/89D-44

AUTONOMOUS FACE RECOGNITION MACHINE
USING A FOURIER FEATURE SET

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Barbara C. Robb, B.S.

Captain, USAF

December, 1989

Approved for public release; distribution unlimited

Acknowledgements

To my husband Roger. You were brave to marry an AFIT student, and your love and encouragement kept me going. Thank you.

To my advisor, Dr. Matthew Kabrisky. Thank you for your unending patience and constant enthusiasm. Never has learning been so much fun.

Thank you to the other members of my committee, Major Steven Rogers and Dr. Frank Brown, for your comments and support.

Thank you Dan Zambon for your support in the lab. You were always there to help me learn the systems and use the lab to do my research.

A special thanks to my fellow students and other individuals at AFIT for letting me take your pictures so I could gather all of the faces I needed for my data base.

Table of Contents

List of Figures

<u>List of Tables</u>

AFIT/GE/ENG/89D-44

Abstract

This thesis demonstrates Fourier coefficients as a reliable feature set for face recognition, using the Autonomous Face Recognition Machine developed at AFIT over the past several years (Routh, 1985; Russel, 1985; Smith, 1986; Lambert, 1987; Sander, 1988).

The Fourier transform portion of the system was examined and improved. The code was made more efficient. Two Fourier transform routines (a fast Fourier transform and a classical Fourier transform) were tested and compared. A voting scheme was incorporated for examining multiple looks at test faces. To further demonstrate performance, the number of faces in the data base was doubled.

Recognition scores of up to 87% were achieved, compared to 63% for Sander's process with Fourier coefficients as a feature set and 67% for Lambert's process with a center-of-mass feature set, (Sander, 1988:32).

This thesis includes complete system documentation, to assist those doing further research in this area.

ENHANCED AUTONOMOUS FACE RECOGNITION MACHINE

I. Introduction

General Issue

For the past several years, much effort has been made at the Air Force Institute of Technology to produce an autonomous face recognition machine. The goal is for a machine to recognize human faces quickly and accurately, without human intervention. The current Autonomous Face Recognition Machine (AFRM) is the work of many people, including Routh, Russel, Smith, Lambert, and Sander. A brief history of the development of that machine follows next. (Routh, 1985; Russel, 1985; Smith, 1986; Lambert, 1987; Sander, 1988)

Background

Routh's Theory (Routh, 1985). Routh proposed the theory on which the AFRM was based when he published his Cortical Thought Theory (CTT). Routh explored the idea of a "gestalt", or essence, of an image as that which the human brain saves to remember objects. He examined applications in speech recognition.

Russel's AFRM (Russel, 1985). Russel applied this theory and built the original AFRM. It was designed to have a computer, with a video system attached, recognize faces.

The AFRM was trained to recognize individuals by processing four different video images of the same person's face. The presumed gestalt of the face was calculated and saved in a data base. Recognition was then accomplished by presenting still another video image of a face to the computer. The gestalt for the face was calculated and then compared, using Euclidean distance, to the averages of the gestalts of each person in the data base in an attempt to identify the person.

The process Russel used to form the gestalt from the video image is shown in Figure 1. He acquired the image of a face against a plain background. Face location was accomplished by an operator centering the face in a frame. The system could then easily separate the face from the plain background. Next, the face image was preprocessed by performing contrast enhancement. The face was then divided in different ways to form six windows. Windowing was necessary because thin symmetric faces were not separable from wide symmetric faces in this system. Windows divided the face into just the left half, just the right half, just the top half, and so on. For each window, the gestalt was calculated. Russel's algorithm for calculating the feature set, or gestalt, was basically the two-dimensional center-of-mass of the window, where the darkest portions were given the highest values. The feature set for each face therefore consisted of six ordered pairs, representing the coordinates

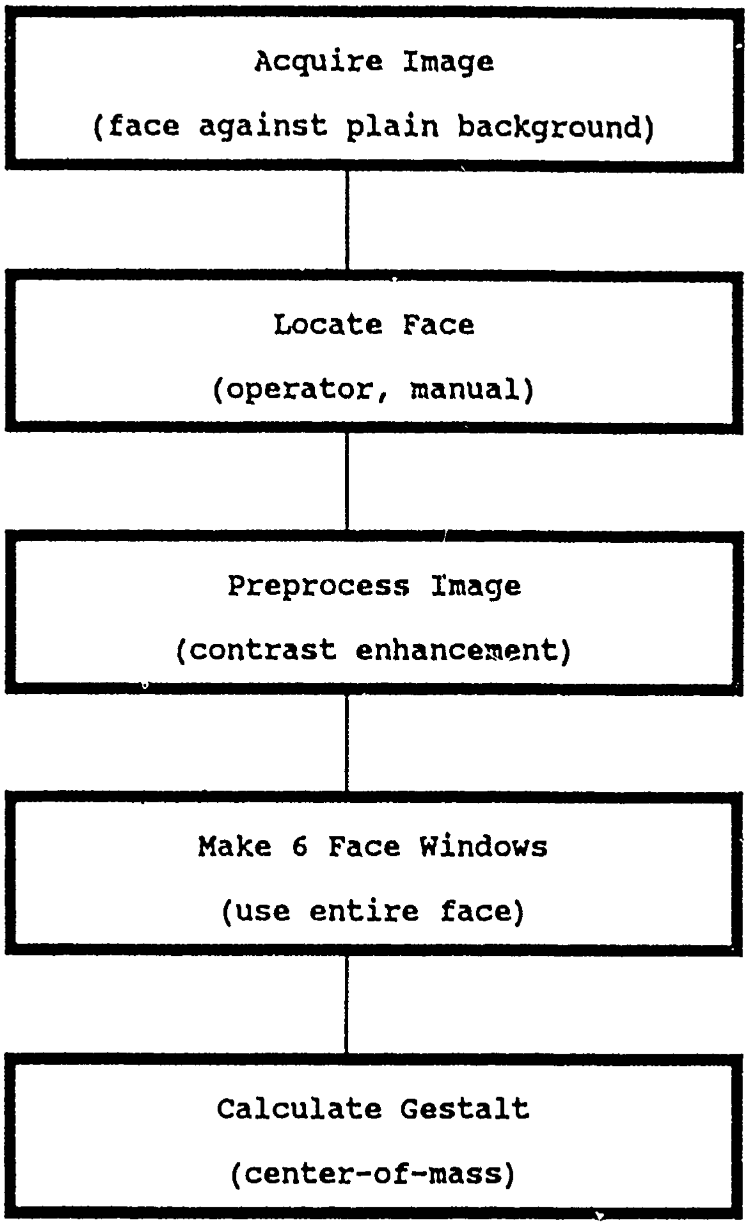

Figure 1. Russel's Image Processing (Russel, 1985)

of the center-of-mass of each of the six windows. This was saved in the data base.

Smith's Improvements (Smith, 1986). Russel's system was limited by requiring a plain background behind the face (to assist in finding the edges of the face) and by requiring human intervention (an operator had to manually place the face in a frame). Smith eliminated both of these restrictions by implementing a new face locator. The process he used is shown in Figure 2. His goal was to locate faces in images with varying backgrounds. His process looked for the brightness "signature" of various facial features. For instance, if a line is drawn horizontally across a picture through a person's eyes, the "signature" shows two approximately equal dark spots with brightness on either side of each of them. Smith used a different set of windows than Russel, since with the uncontrolled background he could reliably locate only the internal features of the face and not the edges. Smith's AFRM did not perform as well as Russell's system, resulting in lower recognition scores.

Lambert's Enhancements (Lambert, 1987). Lambert made several enhancements to the AFRM. His process is shown in Figure 3.

Lambert improved the speed of the system, primarily by rehosting it to its current environment on a Micro-VAX II.

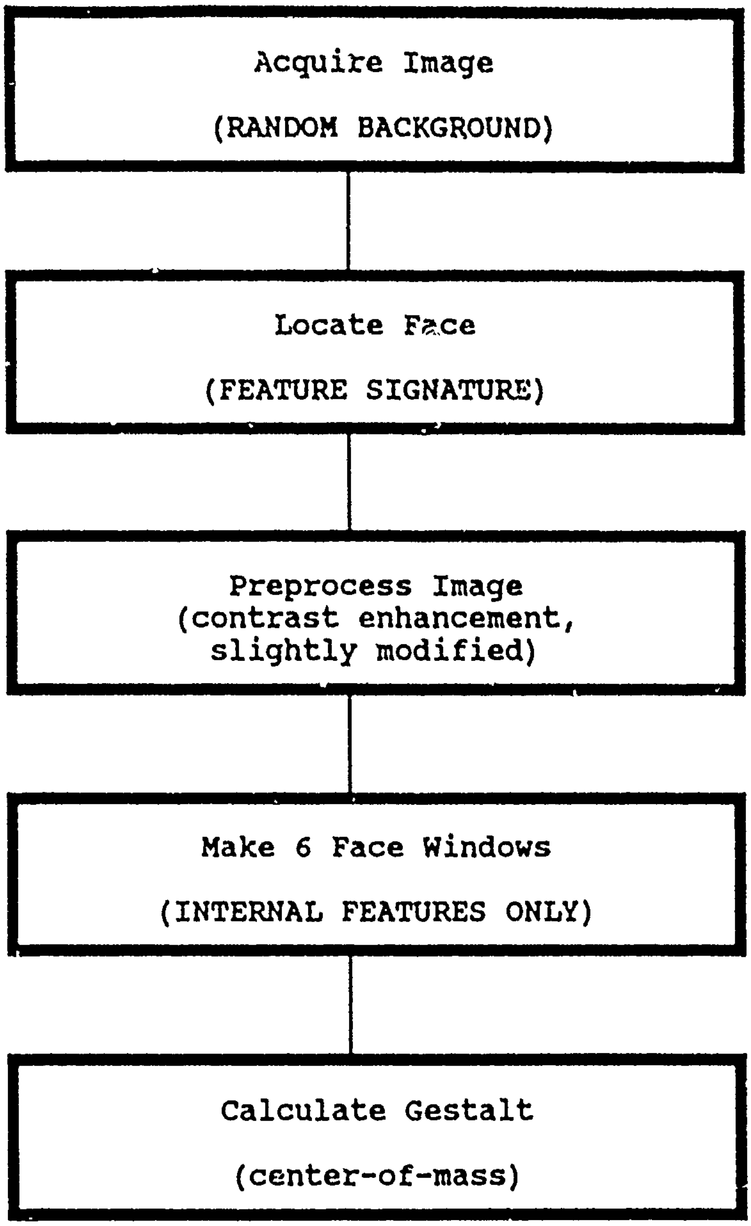

Figure 2. Smith's Image Processing (Smith, 1986)

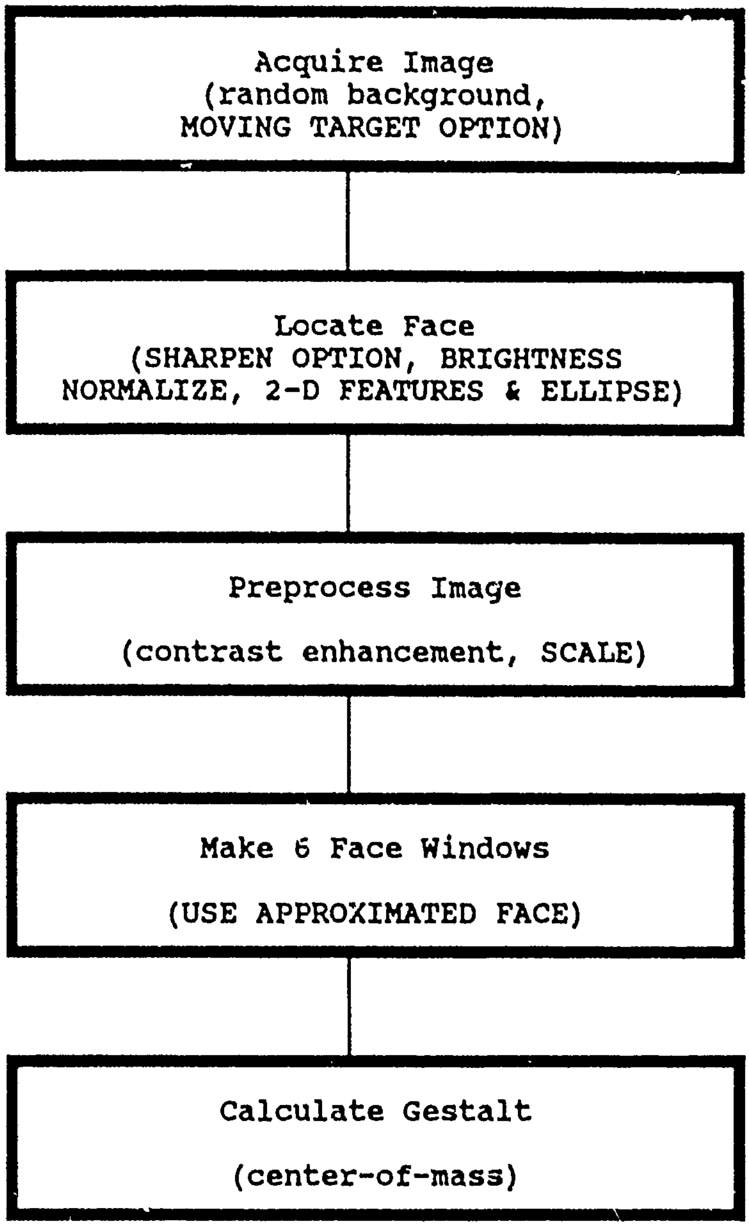

Figure 3. Lambert's Image Processing (Lambert, 1987)

He improved the recognition scores of the AFRM by revising the front-end of the system, where the face location process occurs.

Lambert developed a technique to normalize the brightness of an image. He used the local brightness to adjust the contrast of the image pixels. This allowed for variation in the overall brightness of the scene.

Lambert looked for facial features which he defined to be two-dimensional dark spots, in a specified size range, surrounded by brightness. He then looked for features that together could form two eyes, a nose, and a mouth. This method of finding faces produced fewer false alarms (that is, declaring something to be a face that is not a face) than did Smith's system. Finally, Lambert drew an ellipse around the putative internal features to form a face. Where Smith used the internal portions of the face (features only), Lambert's ellipse approximated the edges of the face and allowed more of the face to be processed. This gave higher recognition scores than did Smith's system. Lambert used still another set of windows, since his ellipse allowed him to use more of the actual face than did Smith's view.

To speed the search for a face in an image, Lambert provided the option of looking for a moving target. The person to be recognized moves into the camera range. The search for a face is limited to the area where movement was noted.

Sander's Further Enhancements (Sander, 1988). In the next step in the evolution of the AFRM, Sander made further improvements to the system. His process is shown in Figure 4.

To improve recognition performance, Sander changed the feature set to use the coefficients of a two-dimensional discrete Fourier transform (2DDFT) of each window of the face, instead of the darkness center-of-mass. 2DDFTs had previously been used successfully for other pattern recognition problems. Sander saved the DC component and the first and second harmonics, resulting in a 5 X 5 array of values for each window, giving 150 values for each face image.

Sander's recognition algorithm continued to use the shortest Euclidean distance between a test image feature set and the trained face feature sets.

To reduce recognition time as the data base of faces grows, Sander used a back-propagation neural network to train the data base on faces and to perform recognition. If successful, this would keep the recognition time a constant, as a result of the distributed memory and computerized properties of the network, regardless of the size of the data base. The previous AFRM would respond progressively more slowly as the data base of faces grew.

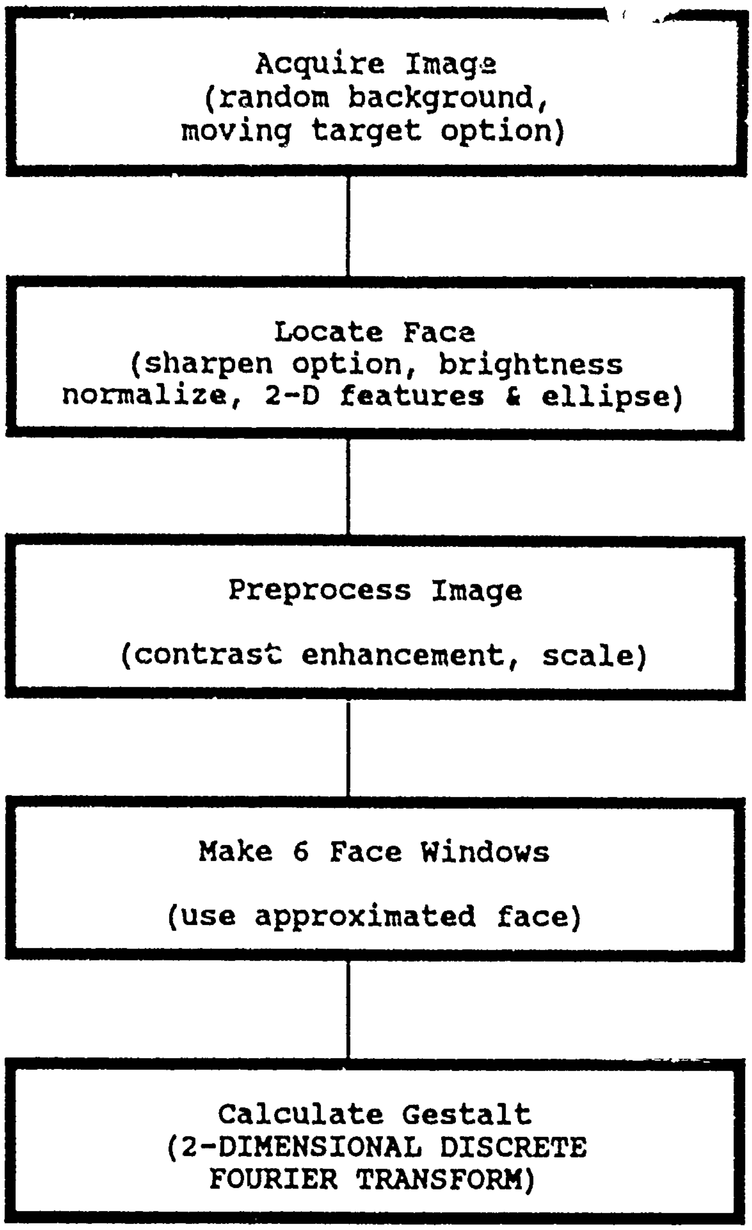

Figure 4. Sander's Image Processing (Sander, 1988)

Problem Statement

The AFRM is not sufficiently fast and accurate, with a large number of faces, to be of practical operational use.

Research Objectives/Methodology

In order to make the AFRM run as quickly and accurately as possible, with a large number of faces, the following steps were taken in this thesis:

1. Examine the 2DDFT feature set. The substitution of a Fourier transform for the center-of-mass calculation did not perform as well as expected in the previous thesis (Sander, 1988:29,30).

   a. Sander's 2DDFT program was examined for possible errors, and corrections were made.

   b. The choice of the number of windows and their contents was examined and reconsidered. Windows were previously based on the center-of-mass calculation (Sander, 1988:8,9). Sander did not examine the choice of windows when changing from the center-of-mass calculation to the use of Fourier transforms (Sander, 1988:25).

   c. Another Fourier transform routine was substituted for the one used by Sander. System performance and processing time were evaluated.

   d. Using the third harmonic, as well as the current DC term and first two harmonics, was considered.

Improved performance was weighed against increased processing time.

2. Test the AFRM with more faces. This system had previously only been trained to recognize 24 faces (Sander, 1988:29). The data bases were expanded and the system tested with 50 faces.

3. Test the use of multiple images of a person for recognition. Experiments were run with the AFRM taking multiple looks at a person, with a voting scheme for resolution. Improved performance was weighed against increased processing time.

4. Improve system documentation. The user's manual was updated and expanded (Appendix A). Complete system documentation was developed (Appendices A, B, and C), to assist further research on the AFRM.

<u>Assumptions</u>

Sander's assumptions were not changed in this thesis. They are as follows:

1. The subject(s) are looking squarely at the camera (the head is not tilted or rotated.

2. The subject(s) are not wearing glasses.

3. The subject(s) have relaxed expressions (the face is not deliberately contorted).

4. Four pictures are sufficient to characterize a person in the data base. (Sander, 1988:2)

Standards

Sander's standards were not changed in this thesis. They are standards that have been met, and must continue to hold as enhancements are made to the AFRM. The standards are as follows:

1. The AFRM should demonstrate "human like" classification of faces.

2. Recognition performance of the AFRM must remain at least as good as that obtained by Russel.

3. No operator interaction is allowed in the face location, windowing, and recognition processes.

4. The AFRM should be able to process scenes with a random, uncontrolled background.

5. The AFRM must be able to process scenes with multiple faces in them. (Sander, 1988:2-3)

The term "human like" implies success or failure during the recognition process that would reflect the capability of a human observer of the same data.

Scope/Limitations

The scope of this thesis was to meet the research objectives previously described. The 2DDFT was examined, corrected, and enhanced to assure proper recognition performance. The system was tested against a larger data base of people. The recognition portion of the system was changed to incorporate a voting scheme using multiple test images of a face. System performance for each modification was weighed against processing time. Documentation of the entire system was developed. No other suggested improvements to the AFRM were addressed in this thesis.

Equipment

The following equipment was used for this thesis:

1. Micro-VAX Computer System
2. Imaging Technology Series 100 Image Processing Board
3. Sander's and Lambert's Computer Files (listed in Appendix A)
4. Dage 650 Video Camera
5. Panasonic WV-5490 Monochrome Monitor
6. Tektronix 4632 Video Hard Copy Unit
7. MicroVMS Version 4.6
8. VAX C Version 2.4
9. ITEX Software Version 1.2

Support

Support was needed from Systems Engineering throughout this research. Support was also needed from fellow students and others at AFIT to allow me to take their pictures to increase my data base of faces.

Summary

This chapter provided a brief background and summary of the work that preceded this thesis effort. Details can be found in each contributor's dissertation or thesis. This background formed the basis for this research project.

The parameters of this thesis were defined by the problem statement, research objectives/methodology, assumptions, standards, scope/limitation, equipment, and support.

The remaining chapters of this thesis describe further evaluation and enhancements to the AFRM. Chapter 2 discusses the methodology used for this effort, and Chapter 3 explains the implementation techniques. In Chapter 4, the results of this thesis are given. Chapter 5 gives conclusions of the research and recommendations for further study.

II. Methodology

Introduction

This is the fifth thesis research effort in the development of the AFIT AFRM, and as the total research effort is very large, there are likely to be subsequent efforts to make the AFRM a practical, operating system.

Previous work using the substitution of a Fourier transform for the center-of-mass calculation did not perform as well as expected (Sander, 1988:29,30). A working feature set for faces is critical to the face recognition process; therefore, examination and correction of the use of Fourier coefficients as a feature set was a next logical step.

Adding faces to the database was also an important step. Making the data base as large as possible helps to prove that the concepts behind the system are sound.

Multiple looks at a person's face, that is multiple face images, with some sort of averaging or voting scheme, were added in an attempt to improve recognition performance.

Previous research efforts included little system documentation. Each succeeding researcher spends increasing amounts of time examining the preceding work and figuring out what the program does and how it does it. Time taken to document the system was an important part of this thesis. It will save follow-on research efforts precious time in reviewing the AFRM program and associated files.

Feature Set

Justification of Method Selected. "Finding an appropriate feature set is one of the most difficult tasks in the pattern recognition process" (Sander, 1988:22). Sander briefly supports the choice of Fourier coefficients as a feature set. Additional support can be found in the literature:

> "Transform theory has played a key role in image processing for a number of years, and it continues to be a topic of interest in theoretical as well as applied work in this field. Two-dimensional transforms are used ... for image enhancement, restoration, encoding, and description." (Gonzalez and Wintz, 1977:36)

> "The discrete Fourier transform often proves to be a powerful tool for the characterization of picture signals, the analysis of imaging systems, and the design of algorithms for image signal processing." (Wahl, 1987:36)

Several applications of image processing are given, such as analysis of fingerprints (Gonzalez and Wintz, 1977:xiii; Wahl, 1987:1) and face profile processing (Wahl, 1987:1). Fourier coefficients have been used as a feature set in research here at AFIT, with much success in alphabet analysis (Bush, 1977) and text recognition (O'Hair, 1984).

Sander chose the coefficients of a 2DDFT (two-dimensional discrete Fourier transform) as the feature set, specifically an FFT (fast Fourier transform). The use of an FFT significantly reduces the number of calculations required to complete the transform, compared to a classical Fourier transform calculation (Gonzalez and Wintz, 1977:79, Wahl, 1987:41-42).

Research Methodology. Sander's test results with the FFT as a feature set were disappointing. "Better recognition was expected from FaceDFT." (Sander, 1988:30)

In an attempt to demonstrate the 2DDFT as a good feature set for faces, this researcher examined Sander's program for possible errors. The code was checked to verify that the inputs to the Fourier transform were correct. The outputs of the Fourier transform, saved in the feature set file, were checked to make sure that the correct values, the DC component and first and second harmonics, were the values saved.

Finally the transform subroutine itself was explored. No citation was given for the transform routine although "an already existing routine was used" (Sander, 1988:25). The source of the code has since been discovered to be an AFIT doctoral student (Fretheim, 1988). Freitheim confirmed that his code had been thoroughly tested (Fretheim, 1989).

Sander's tests were then rerun to verify the results. Since the results of the retest were not the same as Sander's published results, it was assumed that some parameter of his experiments had changed; most likely the face images he left in his directories are not the ones he used for testing.

Freitheim's Fourier transform program is an FFT (fast Fourier transform), which, as mentioned before, runs much faster than a classical transform by reducing the number of required calculations. The input array must have dimensions that are a power of two. Sander's input array to the FFT was always 128 X 128. Since faces vary in size, any unused portions of the array were filled with zeros. This gives good results as long as a large portion of the array is filled with data and not zeros, since only the DC component and two harmonics were saved. An examination of the code showed that the input data was never larger than 64 X 64, meaning that one-fourth or less of the input array of the FFT was used for actual data.

It was also noted that in filling these input arrays, in preparation for the transform, often every other pixel from the image was used. This saves processing time with little or no effect on the accuracy of the result (Kabrisky, 1989).

The code was changed to use a 64 X 64 array as input to the FFT. Where the use of every other pixel resulted in 32 or less values along a dimension, the code was changed to use every pixel, to better fill the array with true data. All face images were then rerun to recalculate their feature sets, and all faces were tested to reevaluate AFRM performance.

Since this method still did not completely fill the input array (faces vary in size), a classical Fourier transform was used, which could intrinsically compute the transform of data files of arbitrary dimensions and does not require input array dimensions to be a power of two. Normally, a classical Fourier transform results in a much slower program. However, since the Fourier transform routine need only calculate the DC component plus two harmonics, some processing speed could be regained. Another AFIT doctoral student provided such a routine (O'Hair, 1989), which was implemented in the AFRM, replacing the FFT. Tests of all faces were rerun and performance compared to the FFT.

Another research objective in examination of the feature set, was to evaluate the utility of particular windows. Since the program allows the six windows of the face to be weighted (the default is to weight them all equally), each window was run alone, to see if perhaps one window gave the same results (or better) than the whole system. Then only that window would be needed in feature set calculations, saving processing time.

The choice of window location and size was reevaluated. Windows were originally chosen considering a center-of-mass feature set, and not reconsidered with the substitution of the Fourier transform (Sander, 1988: 25). Further research showed that these windows were originally chosen because humans found these to be good windows for face recognition (Lambert, 1987:2-12). This choice of windows seems to be a good method and turns out not to be related to the center-of-mass calculation algorithm. Therefore, the window contents were not changed in this thesis.

The final research objective in this area was to consider adding coefficients of the third harmonic of the Fourier Transform to the feature set, weighing increased processing time against improved performance. The increased processing time would not only occur in the calculation of the transform, but also in the time needed to actually recognize faces. When using a neural network for recognition, Sander found the processing time (using a locally available computer) with the third harmonic to be prohibitive (Sander, 1988:31). Since the recognition performance was improved significantly with the other modifications described in this thesis, the third harmonic was not needed as part of the feature set.

Increasing Data Base Size

"A higher dimensioned feature vector should improve the recognition capabilities of the system, by increasing the separation of the template vectors stored for each person" (Sander, 1988:2). This increased separation should mean that the AFRM can run with comparable performance with a larger data base of faces.

The AFRM was previously tested with 24 faces (Sander, 1988:29). In this thesis, the data base was increased to 50 faces.

Multiple Looks

Trained face values are the average of the feature sets of four different images of a person's face. This helps to smooth the data, hopefully finding the middle of the range of values for each feature set entry. Test face values, however, only use one face image and therefore only one set of features. An averaging strategy is also logical for the recognition process, so multiple looks at a face, with some sort of averaging or voting, was examined.

As new people were added to the data base, additional images were taken to allow for a voting strategy. A total of eight pictures of each individual were taken, the four needed for training and four for voting.

After training the AFRM for the new people, each of the four test images was tested individually against the training data base. The results were examined manually to see if voting was likely to improve recognition performance.

Next the four feature sets for the four test images for each person were averaged in the same way training feature sets are handled. This average feature set for each test face was then tested against the training data base and performance measured and examined.

Documentation

Justification of Method Selected. Proper documentation is critical in any system that is to be maintained. "Programs are not used once and discarded, nor are they run forever without change. They evolve" (Kernighan and Plauger, 1978:25). The research on face recognition is not complete; therefore, it is likely that the programs developed in this and preceding theses will continue to evolve.

Documentation is particularly important when those maintaining a program are not those who developed it. (Kernighan and Plauger, 1978:64-65). Comments can provide important information to help someone understand the intent of the program (Pressman, 1982:421). The AFRM software is written in the C programming language, which with capabilities such as complex data structures, requires the programmer to "comment anything that is not obvious" (Darnell and Margolis, 1988:27).

Lack of documentation causes follow-on researchers to spend excessive resources understanding the work that preceded them. Also without documentation, the researcher may misunderstand complex code and make changes that have unintended effects on the operation of the system. A lot of time is then wasted in debugging the program.

Research Methodology. Lambert's User Manual (Lambert, 1987:Appendix C) was updated and included in this thesis as Appendix A. The first chapter, which discusses normal operation of the AFRM, required only a few modifications to reflect additions and changes to the user's view of the AFRM. The second chapter, which covers information for the AFRM maintainer or developer, was expanded substantially. Important file name linkages were explained. A full listing of system files was added, with descriptions of those files. Also, a full listing of system files left by Sander was included, also with descriptions of those files, many of which may still be of use.

Appendix B gives a detailed list of all changes made to Sander's FACEDFT program for this thesis. Many changes were needed to make the program run correctly and to make the menus and messages consistent and more meaningful. Many changes were of a software engineering nature, making the code more readable and maintainable. Warnings are given for potential problems discovered but not changed. Some changes were not made because drastic changes, for documentation proposes but not much affecting the execution of the application, would make it difficult for those trying to follow the development of code from one thesis to another. Finally changes were made to the program to correct and enhance the Fourier transform processing as described in a previous section of this chapter.

The documentation for the program FACEDFT is given in Appendix C. Sander wrote three versions of the face program: FACEDFT implements the FFT only, FACENET implements the neural network only, and FACENETDFT implements both the FFT and the neural network (Sander, 1988:Appendix A). This thesis deals only with changes to FACEDFT. However, many subroutines are duplicated in these multiple versions. Therefore, documentation that is traditionally included in the program as comments is provided here as a separate appendix. Future research may use any of these versions of the face program. The documentation in Appendix C, while specific to FACEDFT, should be very useful when dealing with the other versions. The documentation contains a list of all of the many external files used by the program, with descriptions of those files' contents. Naming conventions for those files are given. The linkage of the program to those files, via DEFINE statements, is described. References to external libraries, needed to link edit the program, is discussed. The major global variables are described. Finally, high-level pseudo-code for the entire program is provided.

Appendix D gives the final program listing for the FACEDFT program.

Appendix E gives substitution code to use the classical Fourier transform in place of the FFT.

Summary

This chapter described the changes made to the AFRM for this research effort, listing those changes and giving the rationale behind them. The next chapter discusses how these changes were implemented.

III. Implementation

Introduction

The implementation of the changes described in the previous chapter was done in an evolutionary fashion. Most of these modifications could be made incrementally, allowing each to be coded, tested, and evaluated separately.

Feature Set

Changes were made carefully and incrementally to the Fourier transform portion of the program. First the existing code was studied and tests run on that system. Next, changes were made to correct and enhance the existing Fourier transform (FFT). In parallel, new faces were added to the system. Then tests were run on all faces. Performance for previous faces, new faces, and the combination of all faces could then be examined. Next the existing Fourier transform routine was replaced with a classical Fourier transform. Tests on all faces were repeated. Once again, performance was evaluated on all faces, combined and separated into old and new.

The program is documented to allow future researchers to choose the enhanced FFT or the classical Fourier transform.

Sander's software versions still exist on backup tape. This includes his original version of the FFT as a feature set, a version that uses a Neural Network for recognition, and a version that combines the FFT feature set and Neural Network. This allows comparisons to still be made between Sander's feature set and the two feature set variations used in this thesis. (This also allows work to be continued on the Neural Network at a future date.)

Increasing Data Base Size

As soon as Sander's system was evaluated, new faces were added to the system, so that the larger data base would be available to evaluate all changes to the AFRM.

Multiple Looks

Voting was looked at whenever testing with new faces. This testing could not be done on face images left by previous researchers, because there weren't sets of four test images per person to use for voting. Voting was examined for the changed version of the FFT coefficients feature set and the classical Fourier transform coefficients feature set, on new faces added to the system.

The code for voting was marked by comments, making it easy to choose to use the voting scheme or not.

Documentation

The documentation was also done in an evolutionary fashion. The first cut covered that part of the code that had to be understood to start research on the other thesis objectives. The documentation was corrected and expanded as work progressed on those other objectives. Finally, remaining gaps were filled in to provide as complete a set of documentation as possible.

Many errors and potential problems were found and corrected. These are listed in Appendix B. Some potential problems were noted but not corrected. Appendix B lists these as warnings and give workarounds where possible.

The overall structure of the program, variable names, etc., were left intact, to avoid loss of continuity with previous and parallel versions of the program. This meant leaving inconsistencies in the structure of the code, particularly noticeable in MENU1, where some processes are broken out into subroutines while some rather long processes are included inline. This also meant leaving such unmeaningful names as MENU1, MENU2, MENU3, sx, sy, and many more.

Documentation of the material left by Sander is given in the User's Manual (Appendix A).

<u>Summary</u>

This chapter described how the changes described in Chapter 2 were implemented in the AFRM. The next chapter examines the results of these changes.

IV. Results

Introduction

This chapter presents the results of this thesis research. Results are compared to previous theses where applicable.

Feature Set

Sander's Results. Sander's FACEDFT program and data were retested for comparison with his results. Table 1 shows the results of these tests. The FACE column shows the performance Sander obtained when testing his data against Lambert's program. The AFRM recognized 16 of 24 people, or 67%. The FACEDFT column gives the results of Sander's FFT version against that same data, where 15 of 24 people, or 63%, were correctly recognized.

Sander proposed that some of the problem may be that Fourier transforms are more sensitive to tilt than are the center-of-mass calculations used by Lambert (Sander, 1988:30). However, in adding more faces to the data base, it was found that the face location algorithm was very sensitive to tilt. Therefore faces with tilt are pretty much eliminated by the face locator and never have feature sets calculated. This researcher does not believe that tilt explains the disappointing performance of the Fourier transform in the AFRM.

Table 1. Comparison of FACE and FACEDFT

Name	Statistics from Sander Thesis (Sander, 1988:32)		Rerun of FACEDFT Test		
	FACE	FACEDFT	Retest of FACEDFT Ver 1	Ver 2	Ver 3
rmaple	2	y	y		
mkabrisky	2	14	2		
mlambert	y	y	y		
llambert	y	y	y	y	
dlambert	2	2	2		
srogers	y	y	y		
ecrawford	y	4	4		
mmayo	y	y	y		
jsillart	y	y	y		
dbane	2	y	y		
druck	y	3	3		
kcox	y	2	2	3	2
efretheim	y	y	y	y	y
lroberts	3	y	y	y	
mdrylie	y	7	y		
gtarr	y	y	y	y	
csabick	y	y	y		
mohair	y	2	4	y	y
ppleva	2	5	y		
dbridges	4	y	y	y	
ddoak	y	y	y		
glorimor	6	y	y		
rmorales	y	y	y	y	
gdawson	y	14	y		

key: - y means yes, face recognized correctly (first choice)
- number means face not recognized, number is placement of correct person in list of choices
- >15 means correct person not in list of top 15 choices

Retest of Sander Data. The remainder of Table 1 shows a retest of Sander's program and Sander's data. Where multiple test images existed, all were used, with simple voting, as there was no way of knowing which Sander may have used. The retest results are not identical to Sander's test, recognizing 19 of 24 people. It is assumed that some parameter of his experiments changed between theses; most likely the face images left by Sander are not the same used by him for his testing. It was decided to ignore these retest results since the parameters of the test were not well understood. Recognition performance in this thesis is compared only with the results Sander claims.

At the same time, the faces were retested using each window alone. Previously all windows were combined and weighted equally for recognition calculations. These tests were run before it was realized that the source of the data could not be traced. The results of the individual window testing are shown in Table 2, with the combined test also presented for comparison. No single window showed superior performance. Regardless of the source of the data, it appears that combining all six windows gives the best feature set for faces.

Test of Modified FACEDFT. Table 3 shows the results obtained when testing the program with the modified FACEDFT program. This is the version where the input array to the Fourier transform is 64 X 64 and that array is filled as

Table 2. Comparison of FACEDFT
And Individual Windows

Name	Ver	Rerun of FACEDFT	Rerun of FACEDFT with single window					
		test	1	2	3	4	5	6
rmaple	1	y	y	y	y	y	y	y
mkabrisky	1	2	4	3	6	3	y	y
mlambert	1	y	y	4	y	y	y	y
llambert	1	y	y	y	y	2	2	y
	2	y	y	y	y	2	2	y
dlambert	1	2	y	y	3	y	2	2
srogers	1	y	y	y	y	y	y	y
ecrawford	1	4	4	9	9	2	2	3
mmayo	1	y	y	y	y	y	y	y
jsillart	1	y	2	2	2	y	y	y
dbane	1	y	y	y	y	y	y	y
druck	1	3	4	4	4	2	3	3
kcox	1	2	2	4	2	y	6	2
	2	3	3	4	4	2	5	2
	3	2	y	2	2	2	4	2
efretheim	1	y	y	y	y	2	y	y
	2	y	y	y	y	y	y	y
	3	y	y	y	y	2	y	y
lroberts	1	y	3	2	y	y	y	2
	2	y	2	y	y	y	y	y
mdrylie	1	y	y	y	y	2	y	y
gtarr	1	y	y	y	y	y	y	y
	2	y	y	y	y	y	y	y
csabick	1	y	y	y	y	y	y	y
mohair	1	4	5	3	7	y	2	10
	2	y	2	3	y	4	2	y
	3	y	y	y	3	y	y	5
ppleva	1	y	3	y	3	2	y	y
dbridges	1	y	3	y	y	3	y	2
	2	y	3	y	y	3	y	2
ddoak	1	y	y	y	y	y	3	3
glorimor	1	y	y	y	y	y	y	y
rmorales	1	y	y	y	y	y	y	y
	2	y	y	3	y	3	y	y
gdawson	1	y	y	2	y	y	y	y

key: - y means yes, face recognized correctly (first choice)
- number means face not recognized, number is placement of correct person in list of choices
- >15 means correct person not in list of top 15 choices

Table 3. Changed FACEDFT

	Name	Ver 1	Ver 2	Ver 3	Ver 4	Averaged
	mkabrisky	y	y	y	y	y
	mmaneely	y	y	y	2	y
	rfiler	y	y	y	y	y
	vmilholen	y	y	y	y	y
	wrecla	y	y	y	y	y
	pwhalen	3	y	y	y	y
	jbrill	y	y	y	y	y
*	rjackson	9	5	6	5	y
	mreinig	7	y	2	y	y
	jhamilton	5	5	15	15	7
*	rricart	>15	10	10	>15	10
	ssablan	y	y	y	y	y
	dbossert	y	y	y	y	y
	thamilton	y	y	2	6	2
	chogan	2	y	y	y	y
	rsmith	10	>15	9	7	6
	tmanely	y	y	y	y	y
	pmarshall	y	y	y	y	y
	jlong	>15	>15	>15	>15	>15
	mleahy	y	y	y	y	y
	syarost	5	7	y	y	y
	tdavis	y	y	y	y	y
	mjohnson	y	y	y	2	y
*	dumphress	5	4	4	4	3
	jsrubar	2	y	2	y	y
	fbrown	2	y	y	y	y
	brobb	y	y	y	>15	y
	gmiracle	y	y	y	y	y
	ssheldon	y	y	y	y	y
	sberger	y	y	y	y	y
	sfiler	y	y	y	y	y
	bconway	y	y	y	2	y
	dbarr	y	y	y	y	y
	kfife	y	y	y	y	y

key: - y means yes, face recognized correctly (first choice)
- number means face not recognized, number is placement of correct person in list of choices
- >15 means correct person not in list of top 15 choices

Table 3. Changed FACEDFT (continued)

	Name	Ver 1	Ver 2	Ver 3
	rmaple	y	y	y
	mlambert	3		
	dlambert	y		
	srogers	y		
	ecrawford	6		
*	mmayo	y		
	jsillart	y		
	dbane	y	2	y
*	druck	>15		
	kcox	7		
	mdrylie	4		
	gtarr	y	y	
	mohair	3	y	
	dbridges	2	2	
	glorimor	y		
	rmorales	y	y	

key: - y means yes, face recognized correctly (first choice)
- number means face not recognized, number is placement of correct person in list of choices
- >15 means correct person not in list of top 15 choices

much as possible with actual data `ther values are set to zero). This table is divided into two sections, one for the faces added in this thesis and one for the previously digitized faces.

It was necessary to reprocess the previously digitized faces to calculate their feature sets with the altered program. This portion of the data base now contains only 16 faces, rather than the previous 24. Eight faces could not be successfully reprocessed, as the face locator could not find four face images to use for training. These eight sets of faces were discarded and 34 new people added to total 50 faces.

Performance of the change to Sander's code alone can be evaluated for the added faces by supposing only one test image was taken, say version 1, and then considering those results.

The previously digitized faces still used simple voting, resulting in a tie for mohair.

The tabulated results are shown in Table 4, lines a, b, and c. The recognition performance is in the same range as those seen by Sander.

The processing time for the classical Fourier transform was compared to the FFT. This was done informally, while watching the AFRM recalculate features sets, since an approximate comparison was all that was needed. There was no noticeable difference in the time needed to calculate feature sets.

Table 4. Tabulated Results

Modified FACEDFT

	50 faces	Raw Score	Percentage
a)	Previously digitized faces	9 or 10/16	56 or 63
b)	Added faces ver 1	22/34	65
c)	Combined with existing faces	31 or 32/50	62 or 64
d)	Added faces simple vote	27 or 28/34	79 or 82
e)	Combined with existing faces	36 or 38/50	72 or 76
f)	Added faces averaged	28/34	82
g)	Combined with existing faces	37 or 38/50	74 or 76

	45 faces (revision of above scores)	Raw Score	Percentage
h)	Previously digitized faces	8 or 9/14	57 or 64
i)	Added faces ver 1	22/31	71
j)	Combined with existing faces	30 or 31/45	67 or 69
k)	Added faces simple vote	27 or 28/31	87 or 90
l)	Combined with existing faces	35 or 37/45	78 or 82
m)	Added faces averaged	27/31	87
n)	Combined with existing faces	35 or 36/45	78 or 80

Note: Where voting schemes result in a tie, two scores are given separated by "or". The higher score is for the tie counting as recognition and the lower score is for the tie counting as failure to recognize.

Table 4. Tabulated Results (continued)

FACEFT

	50 faces	Raw Score	Percentage
o)	Previouly digitized faces	11/16	69
p)	Added faces ver 1	23/34	68
q)	Combined with existing faces	34/50	68
r)	Added faces simple vote	26 or 27/34	76 or 79
s)	Combined with existing faces	37 or 38/50	74 or 76
t)	Added faces averaged	28/34	82
u)	Combined with existing faces	39/50	78

	45 faces (revision of above scores)	Raw Score	Percentage
v)	Previouly digitized faces	11/14	79
w)	Added faces ver 1	22/31	71
x)	Combined with existing faces	33/45	73
y)	Added faces simple vote	25 or 26/31	81 or 84
z)	Combined with existing faces	36 or 37/45	80 or 82
aa)	Added faces averaged	27/31	87
bb)	Combined with existing faces	38/45	84

Note: Where voting schemes result in a tie, two scores are given separated by "or". The higher score is for the tie counting as recognition and the lower score is for the tie counting as failure to recognize.

Face Location Problems. It was later discovered that some faces, although located and processed, were not located correctly. For example, sometimes a dark area under the chin was used as the mouth instead of the mouth itself. This would occur on only some of the faces. When four faces are averaged for training, some using correct features and some not, the feature set is distorted. The faces found to have this problem are denoted in the tables by asterisks in front of the name.

If these faces are discarded, because they were not located and processed as intended by the algorithm, the recognition performance scores must be revisited. The resulting scores can be seen in Table 4, lines h, i, and j. The scores for the previously digitized faces remained about the same. There was an improvement in recognition scores for added faces (67% or 69%, depending on resolution of the tie) and for all faces (71%).

Test of FACEFT. FACEFT used a classical Fourier transform instead of an FFT. The results of testing against FACEFT are shown in Table 5. Once again the table is divided into two portions, previously digitized faces and added faces.

The results for all 50 faces are tabulated in Table 4, lines o, p, and q. Recognition scores are improved over the FFT, at 68% and 69%. The results, after discarding the faces not located and processed correctly, are shown in

Table 5. FACEFT

Name	Ver 1	Ver 2	Ver 3	Ver 4	Averaged
mkabrisky	y	y	y	y	y
mmaneely	y	y	y	8	y
rfiler	y	y	y	y	y
vmilholen	y	y	y	y	y
wrecla	y	y	y	y	y
pwhalen	2	y	y	y	y
jbrill	y	y	y	y	y
* rjackson	y	y	2	y	y
mreinig	2	y	y	y	y
jhamilton	y	3	y	y	y
* rricart	5	6	4	6	5
ssablan	y	y	y	y	y
dbossert	y	y	y	y	y
thamilton	y	y	2	2	y
chogan	2	2	1	y	y
rsmith	7	3	10	13	8
tmanely	y	y	y	y	y
pmarshall	y	y	y	y	y
jiong	>15	>15	>15	>15	>15
mleahy	3	3	3	3	3
syarost	5	4	2	y	2
tdavis	y	y	y	y	y
mjohnson	y	y	y	4	y
* dumphress	5	5	5	9	7
jsrubar	2	y	2	y	y
fbrown	2	y	3	y	y
brobb	y	>15	y	y	y
gmiracle	y	y	y	y	y
ssheldon	y	y	y	y	y
sberger	y	y	y	y	y
sfiler	y	y	y	y	y
bconway	y	y	y	y	y
dbarr	y	y	y	y	y
kfife	y	y	y	y	y

key: - y means yes, face recognized correctly (first choice)
- number means face not recognized, number is placement of correct person in list of choices
- >15 means correct person not in list of top 15 choices

Table 5. FACEFT (continued)

	Name	Ver 1	Ver 2	Ver 3
	rmaple	y	y	y
	mlambert	6		
	dlambert	y		
	srogers	y		
	ecrawford	13		
*	mmayo	3		
	jsillart	y		
	dbane	y	2	y
*	druck	11		
	kcox	y		
	mdrylie	y		
	gtarr	y	y	
	mohair	3	2	
	dbridges	y	y	
	glorimor	y		
	rmorales	y	y	

key: - y means yes, face recognized correctly (first choice)
- number means face not recognized, number is placement of correct person in list of choices
- >15 means correct person not in list of top 15 choices

Table 4, lines v, w, and x. These are the best scores yet; the best score being 79% for previously digitized faces, added faces and combined faces scoring 71% and 73%, respectively.

Increasing Data Base Size

The results of this change are not evaluated separately; the value is in showing how program changes perform against a larger data base of faces.

There were unanticipated problems found when adding new faces to the data base. The Signal Processing Lab had moved to a new location and the overhead lighting was different. Many seemingly good face images could not be found by the face locator. The overhead lighting was strong enough to "wash out" the upper half of many faces, while creating dark shadows on the lower portions of the face.

This situation was alleviated somewhat by placing a desk light upside down on the floor near the camera. The problem was also correctable on some faces by adjusting the brightness threshold (an option on the main menu of the AFRM).

Another problem, previously noted in this chapter, is that the face locator would occasionally use an incorrect part of the face as a feature. Most common was a dark shadow between the mouth and chin being used as the mouth.

Multiple Looks

There are many ways to implement voting and then many ways to score performance. Tables 3 and 5 (for programs FACEDFT and FACEFT) show the results of testing with four test images of each individual for new faces added to the data base. Two voting schemes were evaluated.

First, a simple vote was used, counting how many times a person was correctly identified. In cases with an even number of faces, ties were broken by examination of who ranked above an individual not successfully recognized. Occasionally, there was still a tie, such as the tests on mohair and jsrubar.

The results, before the face location problem was discovered, are given in Table 4, lines d and e (79% or 82% for added faces only and 72% or 76% for all faces) for FACEDFT and lines r and s (76% or 79% for added faces only and 74% or 76% all faces) for FACEFT. The results after discarding the problem faces are shown in Table 4,lines k and l (87% or 90% for added faces and 78% or 82% for 45 faces) for FACEDFT and lines y and z (81% or 84% for added faces and 80% or 82% for 45 faces) for FACEFT. These are improved scores over not using a voting scheme.

The second voting scheme is given in the "averaged" column, referring to averaging the four test feature sets in the same way training feature sets are done. By averaging the four corresponding values for each feature, not only is the closest face important, but how far off any missed faces are is a factor in recognition. This was only tested with new faces, as only those faces have four values for averaging.

Performance on all added faces using averaging is shown in Table 4, lines f and g (82% for added faces and 74 or 76% for all faces) for FACEDFT and lines t and u (82% for added faces and 78% for all faces) for FACEFT. The result, after discarding the faces not located and processed correctly, is shown in Table 4, lines m and n (87% for added faces and 78% or 80% for 45 faces) for FACEDFT and lines aa and bb (87% for added faces and 84% for 45 faces) for FACEFT.

Documentation

The results of documenting the system are difficult to measure. This researcher referred often to notes that were the basis for this documentation. But the more important measure is how useful it is to those modifying this system in the future.

Summary

This chapter showed the results of the research efforts of this thesis. The next chapter discusses conclusions that may be reached from these results.

V. Conclusions and Recommendations

Introduction

This chapter presents conclusions that can be reached as a result of this thesis research. Recommendations for further development of the AFRM are provided.

Feature Set

Correcting and enhancing the use of the FFT to form the feature set improved performance of the system, after the incorrectly located faces were eliminated. Sander's recognition score was 63%; here 71% recognition was achieved for the added faces. Performance scores for previously digitized faces was not improved, but were not totally controlled for this thesis (previous processing of image not well documented or understood). The recognition score for all 45 faces was 67% or 69%, depending on how the tie is resolved.

Substitution of the classical Fourier transform gave equal or better overall recognition than the FFT. The recognition score was 71% for added faces. Scores for previously digitized faces improved significantly, to 79%. The recognition score for all 45 faces was 73%.

When the windows of the face were examined individually, no one window showed performance superior to the combination of all windows. This can be seen in Table 2.

The choice of window contents was briefly evaluated and no change is recommended.

Addition of the third harmonic to the feature set was reconsidered. The possible increase in recognition performance was not believed to be worth the extra processing time, since recognition performance was significantly improved by other means.

Increasing Data Base Size

Increasing the data base size was important for properly evaluating other objectives in this thesis. Adding more people to the training and test data bases gives higher confidence in the results.

Multiple Looks

Adding multiple looks at a face for recognition, both by simple voting and mathematical averaging, increased performance of the AFRM for both of the Fourier transforms. Since the averaging technique gave equal or better performance than simple voting, the averaging technique is preferred.

The best recognition score, 87% (assuming ties cannot be resolved as recognition), was achieved by the combination of either modifying the FFT or using the classical Fourier transform, using the averaging technique for voting, and eliminating faces not corrected located.

When all of these changes are combined, the FFT and classical Fourier transform seem to perform equally well to each other. Both the enhanced FFT and the classical Fourier transform gave higher recognition scores than Sander achieved.

Documentation

The system documentation is now extensive and complete to support future research efforts.

Summary

This thesis demonstrates that the coefficients of the Fourier transform are a good feature set for face recognition, with 87% recognition scores realized.

Recommendations for Further Research

Many recommendations of Lambert, not yet investigated, remain as potentially valuable areas to explore. These include:

- use of color images
- use of binocular images
- use of a parallel processor
- development of a better set of facial features for face location

Other recommendations of Lambert also remain valid. Those listed above should be given higher priority (Lambert, 1987;6-2 through 6-5).

Many of Sander's recommendations were explored in this thesis. The remaining suggestions that were not explored but still quite valid are:

- search for a feature set that is scale and rotation independent, such as use of the log z transform

- evaluate the implemented neural network and compare performance to other existing neural networks

- move the chosen neural network to a parallel processing or vector processing machine (Sander, 1988:35-36)

As a result of this thesis the following recommendations are made:

- As mentioned above, Lambert suggested improvements to the face location algorithm. This was found to be a significant problem when adding faces to the AFRM. With the possible selection of incorrect features that are physically close to the correct features, manual observation of the face location process is required.

- Perhaps controlled lighting should be added to list of assumptions for the AFRM (Chapter 1), if the program cannot be made to overcome the lighting problems.

- Further experimentation may demonstrate whether the classical Fourier transform or the FFT is superior for faces. Perhaps still another transform can be found to improve performance further.

Appendix A

AFRM - Autonomous Face Recognition Machine

USER'S MANUAL

This is an updated version of the User's Manual written by Lambert (Lambert, 1987:Appendix C). Those portions not changed were copied verbatim. Some portions were rewritten and material was added and deleted to reflect the current system.

Table of Contents

Introduction

The information presented in this manual is divided into 2 parts: Chapter 1 gives enough information for a casual user to operate the AFRM (Autonomous Face Recognition Machine), and Chapter 2 gives information needed to maintain the AFRM.

User-friendliness was a primary concern when developing the AFRM. The AFRM is menu-driven, giving the user choices of actions. Prompts tell the user exactly what is required from each keyboard (user) entry. The system is as much as possible fault tolerant. The AFRM code has been documented thoroughly in Appendix C. The program is written in the C programming language (Appendix D). The goal was to write the code as efficiently as necessary, and then as readable as possible.

It is hoped that future modification to the AFRM will maintain an easy user interface and complete system documentation.

I. Operation

Logging On and Off

The easiest way to get to know the AFRM is to sit down and use it. It is located on the Micro-VAX II designated IMAGER, or SMV2A, in the AFIT Signal and Information Processing Lab. To run the AFRM, log onto IMAGER and execute the program. For example:

```
run [kabrisky.brobb.code]facedft
```

The system will run automatically; it will perform several seconds of hardware and software initialization, remind you to turn on the camera and video monitor, and present the main menu. When you are done using the AFRM, return to this main menu and select the QUIT option. This will get you out of the program, reminding you to turn off the camera and video monitor.

Things You Should Not Do

1. The AFRM needs to create temporary files now and then as a normal part of its operation. It will delete these files as soon as they are no longer needed. Since these files are created and deleted without informing the user, the user should avoid saving files with these temporary file names. Never save faces in files named:

BNORM.IMG
ORIG.IMG

At some unannounced time YOU WILL LOSE THEM.

2. The AFRM has been designed to be fault tolerant. You can enter anything you want, at any prompt you want, and the AFRM should handle it. The AFRM will inform you if your input is invalid. The only entries that should not be entered are CTRL-C and CTRL-Y, which terminate the program without going to the main menu option QUIT. These should not be used because the AFRM will not save updated database files.

Menus

The menus allow the AFRM user to do the following:

0: QUIT
This is the proper way to exit the AFRM. Updated database files are saved at this time.

1: ACQUIRE IMAGES
There are several ways to input images into the AFRM and there is a sub-menu for all of the options.

- 0: RETURN TO MAIN MENU

- 1: STATIONARY TARGET
  Allows acquisition of a 512 X 480 image from the camera.

- 2: MOVING TARGET
  Acquires a background scene from the camera (nobody in it), then acquires a second scene (with subject). The AFRM will provide the rectangular area which bounds the region that changed between the two scenes. This target area is all that is processed by the face locator (if locator is selected) and so the face locator will be faster than it would be for a full size scene as well as being less likely to acquire a false target.

- 3: LOAD IMAGE FROM MEMORY
  Allows user to load a previously stored image (up to 200 X 200) to the video screen.

- 4: SAVE IMAGE IN DATABASE
  Allows user to save a full screen image (512 X 480).

- 5: SET CAMERA PORT
  The default is port (0). This allows connecting additional cameras to ports (1) and (2).

- 6: CAMERA CHECK
  Allows continuous acquisition of images so the camera can be positioned and focused.

- 7: RE-INITIALIZE HARDWARE
  Go back to default camera port, clear the video screen, etc.

8: LOAD LARGE IMAGE FROM MEMORY
Allows user to load a previously stored full-screen image (512 X 480).

2: FIND FACES
The face location algorithm will look for faces in the image on the screen and save all it finds to temporary files. There is an option to sharpen the scene that is normally not needed but sometimes helps the face finding process.

3: GESTALT AND IDENTIFY / SAVE
This option only works after a face(s) was found by option #2. If no face(s) was found then this option will return to the main menu. This option runs the gestalt algorithm of the first face found by option #2. Then it runs the recognition algorithm on that face. During recognition the user is allowed to save the face and its gestalt data in the database. There is no other time when a new face and its gestalt data are available for saving in the database, so save it NOW if desired, otherwise it will have be gestalted again (faces are easily deleted from the database if later not needed). If more than one face was found in option #2, then all faces will be gestalted and identified in the order found.

4: DISPLAY CONTENTS OF DATABASE
Shows names of faces for which the AFRM has been trained and names and version numbers of faces that can be used for recognition.

5: DELETE A SUBJECT
This option deletes the training file for this subject. The actual images and gestalt values will still be saved as faces that can be used for recognition and the AFRM can be retrained with this subject later.

6: DELETE AN IMAGE
This option allows the deletion of single images (files that are used for recognition). The actual images are saved as images that are neither trained or used for recognition, but can later be reused for either purpose.

7: TRAIN
This allows the user to train the database with 4 files from the .IMG section (faces used for

recognition) of the database. The files must all have the same name and must have different version numbers. To exit this option at any time, enter a zero version number.
Fault tolerance is really evident in this section of the AFRM because it is so important to maintain a correct database. The AFRM constantly checks user inputs for validity and gives out pertinent information when it finds a mistake. For example, suppose it is decided to train the AFRM with the name Smith, version numbers 1, 2, 3, and 4. The AFRM will verify that the name entered exists in the .IMG section and that it does not exist in the trained section. It will verify that files exist for all the versions selected and that the same version number was not selected more than once. If a mistake is made, the user may exit at any time.

8: DEMONSTRATION

0: RETURN TO MAIN MENU

1: IDENTIFY A PERSON
This option allows the user to demonstrate the recognition capabilities with previously gestalted and saved images, trained and not (those used for recognition). This option can also be used to obtain recognition scores so that the AFRM can be evaluated.

2: TOTAL SYSTEM
This option allows the user to run all AFRM algorithms together starting at image acquisition (moving target) and ending with recognition. It is advised that the user first select the camera port and do a "camera check". Then this option is chosen. When the screen is blank, have the subject walk into the field of view of the camera, turn and stare at the camera, and stand still for a few seconds. As soon as the AFRM "sees" the subject, it will snap a picture and begin to look for a face. If a face is found, then the AFRM will gestalt it and try to recognize the individual. (There is no option to save the face.)

9: CHANGE THRESHOLD

This option is used to change the brightness threshold used by the face location algorithm (option #2). The default value is zero. To get a darker image (more pixels), increase the value, using a small (1 through 5 recommended) positive integer. To get a lighter image (less pixels), decrease the value, using a small (-1 through -5 recommended) negative integer.

II. Technical Details

Files

Many files support the AFRM, both for development and use. The files are backed-up on tape in the Signal and Information Processing Lab. It contains the following:

```
[kabrisky.brobb]          - main directory
  editini.edt             - for full-screen edit mode
  login.com               - for logging onto system

[kabrisky.brobb.code]     - code directory
  autotake.c              - program to take 4 pictures of a
  autotake.exe              person and consolidate as one
  autotake.obj              large image
  facedft.c               - updated version of Sander system
  facedft.exe               (see Appendices B, C, and D)
  facedft.obj
  faceft.c                - substitute transform for facedft
                            (see Appendix E)
  options_file.opt        - for compile
  rl.com                  - for link

[kabrisky.brobb.dbase]    - database for face program
  others.dat              - Lambert test face feature sets
  train.dat               - Lambert training face feature sets
  bothersfft.dat          - Sander test face feature sets
  btrainfft.dat           - Sander training face feature sets
  robbothers.dat          - Robb test face feature sets (from
                            latest facedft)
  robbtrain.dat           - Robb training face feature sets
                            (from latest facedft)
  othersft.dat            - Robb test face feature sets
                            (faceft)
  trainft.dat             - Robb training face feature sets
                            (faceft)

    several files of the form:
  name.img;*              - image files - multiple versions of
                              test faces
  name.pic;*              - image files - multiple versions of
                              training faces
  nameset.img;*           - image files - sets of 4 test faces
  nameset.pic;*           - image files - sets of 4 training
                              faces
```

File Linkages

There is an important linkage between the name.img;* files and othersft.dat (or any others file), with a similar important linkage between the name.pic;* files and trainft.dat (or any training file). When images are gestalted and saved (option 3), the user specifies a name for the picture, usually the person's first initial and last name. The AFRM adds this person to the database as name.img;* where * is the next consecutive VMS version. The person is also added to the gestalt file (such as othersft.dat), the version number being the next consecutive gestalt version for this file. THE AFRM ASSUMES THAT THESE VERSION NUMBERS MATCH! This will be true if the user always sticks with the same training and others file (can be changed in the program) and if the user never uses VMS to alter version numbers. NEVER DO A VMS PURGE COMMAND ON THE DATABASE! Pixel images of people will be lost (forever, if not backed up) and the assumed linkage between the image file and gestalt entry version numbers will be lost. Similarly, when the AFRM is trained for a person (option #7), the four entries in the others file are copied to the training file as versions 1, 2, 3, and 4. (There can only be one set of trained faces for a given name.) The four name.img;* files are copied to be name.pic files. This is a simple VMS copy. The AFRM assumes that the version numbers will be 1, 2, 3, and 4.

When a subject or image is deleted from the AFRM, entries are moved between the training and others files, and name.img;*, name.pic;*, and name.pxl;* (pxl files not gestalted) files may be moved around and version numbers changed.

BOTTOM LINE: Never do a VMS purge command on the database library. Do not move, copy, rename, etc. any pixel files, unless doing development work on the AFRM, and taking into consideration the above description. In that case, it is recommended that the developer study the program and documentation (Appendices C and D) to be certain the AFRM will still work correctly. It would be even better to design around these problems and correct them.

Modification

If a change is needed in the AFRM, then the C source code must be edited, recompiled, and linked to the appropriate libraries. The following commands are needed to accomplish this:

```
edit name.c                  (where name is face or facedft
cc name /nooptimize           or whatever version is to be
@rl name                      modified)
```

The nooptimize option on the compile (second command) is necessary. When the optimizer is used (the default for the command) the program will compile and link but not run correctly. The error is usually a floating point error in the subroutine called recognize. This has been reported to Systems Engineering.

The third command runs a command file called rl.com which identifies all the appropriate libraries for you (so you don't have to do all that typing). rl.com and an associated file called options_file.opt are located in the code directory. The contents of these files are as follows:

rl.com

```
$ link 'P1',dua0:[iti100.itex]itex100/library,-
           dua0:[iti100.toolbox]toolbox/library,-
           dua0:[iti100.vms]vms100/library,-
           dua2:[kabrisky.brobb.code]options_file/opt
```

options_file.opt

```
sys$share:vaxcrtl.exe/share
```

When changing the program, make sure that the define statements at the beginning of the program point to the correct database libraries. See Appendices C and D for the documentation and code.

To create a new training or others file, simply create that file with a zero on the first line (means zero faces in file) and an asterisk on the second line. Change the define statement at the beginning of the program to point to the file(s) to be used.

A modification that may be necessary in the future is a change to the declared size of the arrays in the AFRM. The AFRM is presently set to handle up to 100 subjects in the training file (400 gestalt sets, tlist[400]) and 200 images in the others file (200 gestalt sets, ilist[200]).

Sander's Files (Includes Lambert's Files)

These files are backed-up on TK50 tape 007 in the Signal and Information Processing Lab. The save set is KABRISKY.BCK, dated 29 Jan 29. It contains the following:

```
[kabrisky.brobb]      - main directory
  autotake.c          - program to take 4 pictures of
  autotake.exe          individual and save as one image
  autotake.obj
  backup.com          - back up system
  con.com
  edtini.edt          - for full-screen edit
  face.exe            - Lambert system modified to run for
                        Sander
  l.com               - link C program
  login.com           - standard VMS file
  read.me
  rrobb.img           - test image

[kabrisky.brobb.code]
  b.lis               - listing of compile of bfacefft
  bat.com             - run program
  bat1.com            - run program
  face.c              - Lambert system modified to run for
  face.exe              Sander
  face.obj
  facedft.c           - Sander system with DFT
  facefft.c
  facefft.exe
  facemap.c           - subroutine from system
  facenet.c           - Sander system with Neural Network
  facenet.exe
  facenet.jou
  facenet.obj
  facenetdft.c        - Sander system with DFT and Neural
                        Network
  facenetfft.c
  facenetfft.exe
  fft.c               - subroutine from system
  get.com             - copy files
  l.com               - link program
  loadface.c          - program to load face file to screen
  mom.c               - program to calculate distances between
  mom.exe               feature sets of faces
  mom2.c
  mom2.exe
  mom2.out
  myface.c            - piece of system
  myface.exe
```

```
  newface.c          - piece of system
  newface.exe
  nl.com             - link program
  options_file.opt - needed to compile and link C programs
  put.com            - copy files
  readfftfile.c      - subroutine from system
  results.
  results.net
  sander.img
  save_fft.c         - subroutine from system
  sub.doc            - block comment set up to document
                       routines
  test15020.net
  testface.c         - piece of system
  testface.exe
  testsubs.c
  trans.c            - Lambert's code for feature set

[kabrisky.brobb.code.origcode] - Lambert's files -
  autotake.c                       documented in his thesis
  autotake.exe                     (Lambert, 1987:
  bright.c                         Appendix B)
  bright.exe
  face.c
  face_sig.c
  face_sig.exe
  graph.c
  graph.exe
  mti.c
  mti.exe
  newface.c
  sixel.c
  sixel.exe
  sub_demo.c
  sub_demo.exe

[kabrisky.brobb.code.thesisdbase] - empty

[kabrisky.brobb.dbase]
  b.dat              - training files and others (test face)
  bothersfft.dat       files (feature set values) for various
  btrainfft.dat        versions of code
  nothersfft.dat
  ntrainfft.dat
  others.dat
  othersfft.dat
  train.dat
  trainfft.dat
```

```
  several files of the form:
name.img;*        - pixel files - multiple versions of test
                    images
name.pic;*        - pixel files - multiple versions of
                    training images
```

```
[kabrisky.brobb.prntimage]
  baddisp.c
  baddisp.exe
  baddisp.obj
  bat.com
  bhalf.c
  bhalf.exe
  bhalf.obj
  bhalfface.c
  bhalfface.exe
  bhalfface.obj
  faceloc.img
  l.com
  phtest.c
  phtest.exe
  phtest.for
  phtest.obj
  prntbig.exe
  prntbig.for
  prntbig.lis
  prntbig.obj
  prnth.for
  prnthuge.exe
  prnthuge.for
  prnthuge.obj
  prntm.exe
  prntm.for
  prntm.obj
  recognize.out
```

Appendix B

Program Changes

Corrections and Cleanup:

- Hard-coded files names were located throughout the program. These were all moved to the beginning of the program in DEFINE statements. Those files now are the library of picture images for training, the library of picture images for testing, the library for picture images not trained or tested, the training data base, and the testing (others) data base.

- The program documentation stated that the data base files were read into memory at the start of the AFRM and written back to disk upon exit. This was not true. Now it is.

- The program attempted to save the data base files upon exit only if those files changed. The criteria used was not conclusive (same number of entries at start and end). Now the data bases are truly only saved if changed. Flags (TRAIN_CHANGED and OTHERS_CHANGED) keep track of the files status.

- Cleaned up menus:
  - options in ascending order
  - line spacing is consistent
  - all options are listed (0.9 and 1.8 existed but were not shown on the menu)
  - code is ordered by option

- FACEREC has coding errors. Some code was partially repeated and the subroutine would not link correctly. The code was corrected.

- It was found that compilation with the optimizer (the default) created a program that could fail with a floating point overflow error. It was determined that pointers within the structure were not correct. Compilation without the optimizer eliminates the problem. This has been reported to Systems Engineering.

- Some character arrays were too short to contain the necessary data. This occurred when file names were built by concatenating pieces of data. In the C programming language, offsets are used even if out of range. This caused unpredictable results, depending what was accessed. Character arrays were expanded, including padding for possible future expansion.

- Unused code was deleted. This included CORTRAN16, RTRANSA, RTRANSB, NEWLINE, ORIGI, ORIGK, and LOADFACE.

- The recognition list (option 3) showed a ranking of all faces in the data base. As the data base grew, this caused the top rated faces (the most important ones) to roll out of view. The recognition list now displays only the top 15 faces.

- Where user input is limited (such as 10 characters for a name), this limit is enforced.

- Deleting a trained face originally (Lambert) caused the data to be moved from the training data base to the testing (others) data base and the video images to be renamed. This was changed by Sander to truly delete everything associated with the face. This was restored to the Lambert algorithm. Also, when images are deleted from the testing (other) data base, they are now moved to the pixel library.

- Comments have been updated.

- The title displayed on the video screen has been updated to 1989.

- Spacing and indenting has been improved for readability.

- User messages have been updated and improved.

- When a face was displayed (1.3) the name field was not changed. If a name was already up there (from another option), it would remain, possibly causing confusion. This field is now blanked when an image is displayed.

- When a face was displayed (1.3) it was put in the upper left corner of the screen. It is now displayed centered about an inch from the top of the screen.

- The training option (7) asked for version -1 to exit the option. But the program only looks at the first character (not two). This option has been changed to look for version 0 to exit.

- Some options did not allow the user to exit if entered accidentally. This has been corrected where possible.

- The display of test images (others) showed a list of .IMG files. This was misleading since it is really a list of entries in the test (others) data base, which may or may not match the list of .IMG files. This has been changed.

- Option 1.8 was very similar to option 1.3. Option 1.8 is now rewritten to load full-screen pixel files to the video screen.

Warnings:

The following problems were not corrected and should be noted:

- Only the first digit of a version number is used by the program. Version numbers must not exceed 9.

- Never purge the data base libraries. All of those versions are needed by the AFRM.

- Never alter the contents of the data base libraries without a thorough understanding of effects on the AFRM. The program assumes that it is doing all manipulations to this library. User alterations can affect the AFRM.

- The program assumes that the appropriate libraries and files exist, as listed in the DEFINE statements at the beginning of the program.

- FFT and DFT are used interchangeably in both the code and comments.

- The code is very inconsistent in level of detail. For instance, for the main menu, some detailed code is included instream and some is broken out into subroutines. Also some names are not meaningful (such as MENU1 and MENU2). It was decided not to make major changes in this area, because those studying the progression of the program over the past few years would lose the continuity of the program structure and names used in the program.

Enhancements:

- The 2DDFT routine used a 128 X 128 array. It was discovered that nothing larger than 64 X 64 was ever used (the remainder padded with zeroes). The 2DDFT was changed to use up to a 64 X 64 array.

- To build the input array to the 2DDFT, every other pixel in both the horizontal and vertical direction was used. To better fill the 64 X 64 array, sometimes every pixel in one or both direction was used.

- A voting scheme was implemented in subroutine RECOGNIZE. It is marked by comments so the programmer can choose to use voting or not.

- A classical Fourier transform can be substituted for the FFT used by Sander. It is in a separate source file called FACEFT.C. It can be exchanged for comparable code in FACEDFT.

Appendix C

Software Documentation

Table of Contents

Files Used by FACEDFT:

- file containing feature sets for trained faces pointed to by DEFINE of TRAIN

  format is:

    integer - number of face entries in file

    for each entry (there are 4 entries for each trained person):

    ```
    character - name for face (first initial and last
                name)
    integer   - version number
    double    - 6 X 5 X 5
                for each of 6 windows there is a 5 X 5
                array (gestalt) of features, which is
                the 2DDFT (DC and 2 harmonics)
    ```

    * - EOF marker

- file containing feature sets for test (others) faces pointed to by DEFINE of OTHERS

  format is:

    integer - number of face entries in file

    for each entry (there is 1 entry for each face test image):

    ```
    character - name for face (first initial and last
                name)
    integer   - version number
    double    - 6 X 5 X 5
                for each of 6 windows there is a 5 X 5
                array (gestalt) of features, which is
                the 2DDFT (DC and 2 harmonics)
    ```

    * - EOF marker

- name.pic;* in DBASEDIR -- trained face images (pixels), where name is the first initial and last name of the person; there should be four version for each person

- name.img;* in IMAGEDIR -- test (others) face images (pixels), where name is the first initial and last name of the person

- name.pxl;* in PIXELDIR -- any other face images (pixels) not matching trained or test faces

- orig.img;* -- used to temporarily save an original face image

- bnorm.img;* -- used to temporarily save a brightness normalized face image

DEFINE References:

```
DBASEDIR - library with pixel images used for training
IMAGEDIR - library with pixel images used for testing
           (others)
PIXELDIR - library with other pixel images
TRAIN    - file of features for trained faces
OTHERS   - file of featured for test (others) faces
```

INCLUDE References:

These statements are needed to link this program with standard system routines (I/O, etc.), system math routines, and ITEX (video processing) routines.

Important Global Variables:

```
TLIST        - structure list for up to 400 trained face
               entries; with 4 entries per face, this is 100
               trained faces
             - during execution of the AFRM, the TRAIN file
               resides in this structure list to speed
               execution of the system

I            - number of trained face entries in TLIST

ILIST        - structure list for up to 100 test face image
               entries
             - during execution of the AFRM, the OTHERS file
               resides in this structure list to speed
               execution of the system

K            - number of test face image entries in ILIST

THR          - brightness threshold

SX,SY,       - position and dimensions for face
FX,FY

TRAINED_CHANGED - used to determine whether or not TLIST
                  structure should be written back to the
                  TRAIN file when application is terminated

OTHERS_CHANGED  - used to determine whether or not ILIST
                  structure should be written back to the
                  OTHERS file when application is terminated
```

High-Level Pseudo-Code:

MAIN

- initialize system/hardware
- display title screen on video screen
- call READFFTFILE twice to read trained faces and test (OTHERS) faces into main memory (TLIST and ILIST)
- initialize TRAIN_CHANGED and OTHERS_CHANGED flags to false
- display and execute MENU1 to show and perform main menu

MENU1 - display main menu and execute user's choice:

0: quit
- if changed, save trained faces and test faces from main memory (TLIST and ILIST) back into TRAIN and OTHERS files
- return to operating system

1: acquire images
- call MENU2 to give options

2: find faces
- give user option to sharpen faces
- call SHARPEN if option chosen
- call FACEMAP

3: gestalt and identify / save
- call FACEREC

4: display contents of database
- call DISPLAY with trained faces (TLIST)
- call DISPLAY with test (others) face images (ILIST)

5: delete a subject
- call DISPLAY with trained faces (TLIST) to show user choices
- get subject's name from user
- loop through trained faces (TLIST)
  - if subject's name found
    - loop through test (others) face images (ILIST) to find subject's name and largest version number
    - increment that largest version number
    - loop 4 times (4 versions), copying TLIST entries to ILIST as new versions
    - move actual images from DBASEDIR to IMAGEDIR
    - continue to loop through TLIST, moving the remaining images up 4 places (to delete subject)
- if subject not found in TLIST
  - print error message
- set TRAINED_CHANGED and OTHERS_CHANGED flags

6: delete an image
- call DISPLAY with test face images (ILIST) to show user choices
- get subject's name and version number from user

- loop through test face images (ILIST)
  - if subject's name and version number match test face image
    - move actual images from IMAGEDIR to PIXELDIR
    - continue to loop through ILIST, moving each entry up one place (to delete subject)
- if subject not found in ILIST
  - print error message
- set OTHERS_CHANGED flag

7: train
- call DISPLAY with trained faces (TLIST) to show user names of trained people
- call DISPLAY with test face images (ILIST) to give user choices
- get subject's name from user
- loop through TLIST, looking for subject name
- if no match, print error message and exit option
- ask user for 4 version numbers
- loop 4 times (4 versions), verifying unique version numbers and searching ILIST to verify existence of image files
- print error messages as appropriate
- loop 4 times (4 versions), copying entries from ILIST to TLIST, and getting rid of ILIST entries by moving remaining entries up
- move actual images from IMAGEDIR to DBASEDIR
- set TRAINED_CHANGED and OTHERS_CHANGED flags

8: demonstration
- call MENU3

9: change threshold
- allow user to enter new variable threshold (used by BRIGHT_NORM)

MENU2 - display menu and execute user's choice:

0: return to main menu

1: stationary target
- use ITEX routines to put camera image on video screen

2: moving target
- use ITEX routines to put camera image on video screen
- subtract successive images and look for movement (change)
- use ISOLATE to isolate target from surroundings

3: load image from memory
- loads an image file (pixels) of up to 200 X 200
to the screen, centered and near the top

4: save image
- saves an entire video screen image to an image (pixel) file in PIXELDIR

5: set camera port
- allow user to choose camera port 0, 1, or 2

6: camera check
- allow user to adjust camera (focus and such)

7: re-initialize hardware
- initialize hardware and clear video screen

8: load face from memory
- similar to option 3 except loads a full-screen image

MENU3 - display menu and execute user's choice:

0: return to main menu

1: identify a person
- call DISPLAY with test face images (ILIST) to give user choices
- prompt user for person's name and version number
- search ILIST for face entry
- display corresponding picture on video screen
- call RECOGNIZE

2: total system
- call AFRM

FACEMAP - use BRIGHT_NORM to brightness-normalize the image; then use FEATUREMAP to look for possible facial features (eyes, nose, mouth), then draw ellipse and rectangle around features to define face, saving bnorm.img and orig.img

FEATUREMAP - look for possible features

GESTALT - display windows on video terminal
- set up 6 windows for face

- calculate gestalt, or feature set, for each window
  - use CLEAR_CRAY to initialize calculation arrays to zero
  - put values for windows in calculation arrays
  - call FFT2 (2DDFT routine)
  - call SAVE_FFT to save Fourier values (DC and first two harmonics) in ILIST[0] (work space)

SAVE_FFT - save 2DDFT information in ILIST[0] (working space)

- the feature set for each window for each face is a 5 X 5 array (DC value and first two harmonics)
- this data must be copied from the calculation arrays to ILIST[0]:

	0	1	2	3	4
0	crayi [02][02]	crayi [01][02]	crayi [00][02]	crayi [63][02]	crayi [62][02]
1	crayi [02][01]	crayi [01][01]	crayi [00][01]	crayi [63][01]	crayi [62][01]
2	crayi [02][00]	crayi [01][00]	crayr [00][00]	crayr [00][01]	crayr [00][02]
3	crayr [01][62]	crayr [01][63]	crayr [01][00]	crayr [01][01]	crayr [01][02]
4	crayr [02][62]	crayr [02][63]	crayr [02][00]	crayr [02][01]	crayr [02][02]

CLEAR_CRAY - clears (sets to zero) the calculation arrays to be us for the 2DDFT

FFT2 - perform 2DDFT (two-dimensional discrete Fourier transform) with FFT (fast Fourier transform):

- call FFT for each row
- call FFT for each column

FFT - perform one-dimensional fast Fourier transform

BRIGHT_NORM - brightness-normalize image
(Lambert, 1987:3-15 to 3-21)
- use threshold set with main menu, option 9

CONT_ENHANCE - contrast-enhance image
(Lambert, 1987:3-21, 3-22)

SCALE - rescale image if needed

FACEREC - display brightness normalized face (bnorm.img)
- call CONT_ENHANCE (commented out in this version)
- call SCALE (commented out in this version)
- call GESTALT to calculate feature set
- display original face (orig.img)
- if called from main menu option 3
  - ask user whether or not to save face in data base
    - for save,
      - ask for subject name
      - find highest version number for this name and increment
      - add face information to ILIST
      - save face in name.img file in IMAGEDIR
      - set OTHERS_CHANGED flag
  - call RECOGNIZE (using ILIST[0]) to try to recognize face
- delete bnorm.img and orig.img

RECOGNIZE - attempts to recognize the current image (parameter passed which is pointer in ILIST)

- loop through all 6 windows

  - loop through all trained people (sets of 4 faces)

    gix = the average of this feature for the 4 training entries

    gux = the corresponding test feature (can also be averaged if voting scheme used)

    sum the squares of the difference between each feature and gix and divide that sum by 4

    sigix = the square root of the value just calculated

    if sigix < 0.5, set it to 0.5

    c is the running sum of

    $$(gix - gux)^2 / 4 * sigix^2$$

  - $v = e^{-c/10000}$ * the window weight (default 1.0)

  Note: The value of the variable c now gives a measure of closeness for the test feature set to trained feature sets. This last equation, giving v, reverses the values so that a higher score means a closer value and a lower score means a further value. This equation also scales values between zero and one.

- loop through all people
  - the recognition distance for each person against the test image is the sum of the v values for the six windows. This value is between zero and one, with the highest value as the closest to the test image

- sort the recognition distances

- show the three closest faces and print the names and distance values for the closest 15 faces

- if there aren't three faces close enough (settable value), print message

- Note: windows can be weighted by setting values (default is all equal)

ISOLATE - looks for moving target (changed pixels) on top half of screen

AFRM - for demo, run through entire AFRM process, using MTI (moving target indicator) and not saving any images or feature sets

DEL - delete current image - BNORM.IMG and ORIG.IMG

PRTC - prompt user to press return to continue

CLS - clear the terminal screen

READFFTFILE - read a feature set file (first parameter) and load into structure (second parameter)

- first read an integer that gives number of entries in file
- then loop to read entries and put in structure
- for each entry there is a name followed by 6 (windows) X 5 X 5 (features) floating values

- Note: position zero in the structure is a work area and is not used by this procedure

WRITEFFTFILE - save feature set file (first parameter), saving from structure (second parameter)

- reverses read process (described above)

DISPLAY - first parameter gives structure (TLIST or ILIST)
second parameter gives number of entries in structure third parameter is a flag for TLIST or ILIST

- if TLIST, display names of people found in TLIST
- if ILIST, display names and version numbers for people found in ILIST

COPYFILE - build and execute command to copy one file (first parameter) to another file (second parameter)

FACEFT

FACEFT.C is a separate source file. It contains alternate versions of GESTALT, SAVE_FFT, CLEAR_CRAY, and FT2. These routines can be swapped for the comparable code in FACEDFT.C to use a classical Fourier transform in place of the FFT.

Appendix D

FACEDFT Source Listing

```
/***********************************************************************
*
*
*                              This is FACEDFT.C
*
*
************************************************************************
*    Name FACE - AUTONOMOUS FACE RECOGNITION MACHINE
*
*    Author: Laurence C. Lambert - 1987
*    Based on the Data General (Eclipse/Nova) AFRM by E. Smith
*
*       12 JAN 88 L. Lambert
************************************************************************
*
*       Modified in 1988 by D. Sander
*
*       Major change was to use 2DDFT for feature vectors instead
*       of moment calculation
*
************************************************************************
*
*       Modified in 1989 by B. Robb
*
*       Major changes were to add choice of a simple (not fast) Fourier
*       Transform for feature set (see FACEFT.C), averaging of four test
*       images for recognition, and several clean-up changes
*
********************************************************************* */
```

```
#define dbasedir "[kabrisky.brobb.dbase]"
#define imagedir "[kabrisky.brobb.dbase]"
#define pixeldir "[kabrisky.brobb.dbase]"
#define train "[kabrisky.brobb.dbase]robbtrain.dat"
#define others "[kabrisky.brobb.dbase]robbothers.dat"

#include "sys$library:stdio.h"
#include "dua0:[itil00.itex]stdtyp.h"
#include "dua0:[itil00.itex]itex100.h"
#include <math>

static int option,test,sy,sx,fy,fx,nf,x,y,z,thr=0;
int i,k;                          /* i = size of tlist, k = size of ilist */
char train_changed,others_changed;

struct list {
        char name[11];
        int  num;
        double feature[6][5][5];
        };

static struct list tlist[400] = {0,0};
static struct list ilist[200] = {0,0};
static double gauss[257];
```

```
/*************************************************************************/
main()                                                              /* 23 */
{

    unsigned    base = 0x1600;
    long        mem = 0x200000L;
    int         flag = 1,block = 8;

sethdw(base,mem,flag,block);
initialize();
sclear(100,1);
cls();
printf("\n             Initializing hardware and reading dbase files.");
printf("\n            Please turn on the video monitor and the camera.");
text(120,200,0,8,0,"AFRM");
text(110,415,0,1,0," AIR FORCE INSTITUTE OF TECHNOLOGY");
text(110,430,0,1,0,"    SIGNAL PROCESSING LABORATORY");
text(110,445,0,1,0,"AUTONOMOUS FACE RECOGNITION MACHINE");
text(110,460,0,1,0,"               1989");
/* del();*/
i = readfftfile(train,tlist);
k = readfftfile(others,ilist);
train_changed = 'f';
others_change' = 'f';
 " = sx = .y = 0;
   = fy = 511;
   l();
```

```
/***********************************************************************/
menu1()                                                          /* 55 */
{

    char t2[80],t3[80],temp[80],ch[80];      /* j,l and m are counters */
    int ver[5],j,l,m;  /* ver[1-4] - file version numbers to train on. */
    int r,w,w1,w2;
    int rot[5] = {-20,-10,0,10,20};

for(;;) {

   cls();
   printf("\n                 AUTONOMOUS FACE RECOGNITION MACHINE\n ");
   printf("\n                        ***** MAIN MENU  *****\n ");
   printf("\n                      0:QUIT");
   printf("\n                      1:ACQUIRE IMAGES");
   printf("\n                      2:FIND FACES");
   printf("\n                      3:GESTALT AND IDENTIFY / SAVE");
   printf("\n                      4:DISPLAY CONTENTS OF DATABASE");
   printf("\n                      5:DELETE A SUBJECT");
   printf("\n                      6:DELETE AN IMAGE");
   printf("\n                      7:TRAIN");
   printf("\n                      8:DEMONSTRATION");
   printf("\n                      9:CHANGE THRESHOLD\n\n\n\n\n\n<");

   scanf("%d",&option);
   cls();

   switch(option){

   case 0:
      printf("\n\n saving DBASE files...");
      if (train_changed != 'f')
         writefftfile(train,tlist,i);
      if (others_changed != 'f')
         writefftfile(others,ilist,k);
      cls();
      printf("      Please turn off the video monitor and the camera.");
      printf("\n\n\n\n\n\n\n\n\n\n\n");
      return;

   case 1:
      menu2();
      break;

   case 2:
      printf("\n Sharpen image? (Y/N/Quit) >");
      scanf("%s",ch);
      if (ch[0] == 'q' || ch[0] == 'Q') break;
      if (ch[0] == 'y' || ch[0] == 'Y') {
         sharpen(sx,sy,fx-sx,fy-sy,3);
      }
```

```
         facemap();
         break;

      case 3:
         facerec(1);
         break;

      case 4:
         display(tlist,i,8);
         display(ilist,k,6);
         prtc();
         break;

      case 5:
         display(tlist,i,8);
         printf("\n\n                         ***** DELETE SUBJECT *****");
         printf("\n\n Are you sure (Y/N)? >");
         scanf("%s",temp);
         if (temp[0] != 'Y' && temp[0] != 'y') break;
         printf("\n\n Enter subject's name. >");
         scanf("%s",temp);
         printf("\n");

         test = 0;
         for (j=1; j<(i+1); j=j+4) {
            if ((strncmp(tlist[j].name,temp,11)) == 0) {
               l = 0;  /* look for highest existing version of .IMG file. */
               for (m=1; m<(k+1); m++)
                  if (strncmp(ilist[m].name,temp,11)==0 && ilist[m].num>>l)
                     l=ilist[m].num;
               l = l + 1; /* add 1 to highest version to get new version. */

               for (m=1; m<5; m++) {    /* put 4 new versions into ilist. */
                  k = k + 1;
                  ilist[k].name[0] = '\0';
                  strncat(ilist[k].name,temp,11);
                  ilist[k].num = l;
                  for (w=0; w<6; w++)
                     for (w1=0;w1<5;w1++)
                        for (w2=0;w2<5;w2++)
                           ilist[k].feature[w][w1][w2] =
                            tlist[j+(m-1)].feature[w][w1][w2];
                  l++;
               }

               l = l - 4;
               for (m=0; m<4; m++) {
                  t2[0] = '\0';
                  t3[0] = '\0';
                  strcat(t2,dbasedir);
                  strcat(t3,imagedir);
                  strcat(t2,temp);
                  strcat(t3,temp);
```

```
            strcat(t2,".pic;\0");
            strcat(t3,".img;\0");
            ch[1] = '\0';
            ch[0] = m + 1 + '0';
            strcat(t2,ch);
            ch[0] = 1 + '0';
            strcat(t3,ch);
            copyfile(t2,t3);
            delete(t3);
         }

         for (m=j; m<(i-3); m++) tlist[m] = tlist[m+4];
         printf("\n");
         i = i - 4;
         j = i + 2;                /* forces end of loop through tlist */
         test = 1;                 /* indicates that subject was found */
   }}

   if (test != 1) printf("\n\n Subject not found.");
   else {
      printf("\n\n Subject deleted ");
      printf("(Moved from training data base");
      printf(" to image data base)");
      train_changed = 't';
      others_changed = 't';
   }
   prtc();
   break;

case 6:
   display(ilist,k,6);
   printf("\n\n                    ***** DELETE IMAGE *****");
   printf("\n\n Are you sure (Y/N)? >");
   scanf("%s",temp);
   if (temp[0] != 'Y' && temp[0] != 'y') break;
   printf("\n\n Enter subject's name. >");
   scanf("%s",temp);
   printf("\n\n Enter version number. >");
   scanf("%s",ch);
   printf("\n");

   test = 0;
   for (j=1; j<(k+1); j++) {
      if (strncmp(ilist[j].name,temp,11) == 0) {
         if (ilist[j].num == (ch[0] - '0')) {
            t2[0] = '\0';
            t3[0] = '\0';
            strcat(t2,imagedir);
            strcat(t3,pixeldir);
            strcat(t2,temp);
            strcat(t3,temp);
            strcat(t2,".img;\0");
            strcat(t2,ch);
```

```
            strcat(t3,".pxl\0");
            copyfile(t2,t3);
            delete(t2);
            for (m=j; m<k; m++) ilist[m] = ilist[m+1];
            k = k - 1;
            j = k + 2;
            test = 1;
   }}}

   if (test != 1) printf("\n\n Image file not found.");
   else {
      printf("\n\n Image file deleted ");
      printf("(Moved from image data base");
      printf(" to pixel data base)");
      others_changed = 't';
   }
   prtc();
   break;

case 7:
   display(tlist,i,8);                                        /* 194 */
   display(ilist,k,6);
   printf("\n\n                    ***** TRAIN *****");
   printf("\n\n Enter person's name. >");
   scanf("%s",temp);

   test = 0;
   for (l=1; l<(i+1); l=l+4)                           /* test name */
      if ((strncmp(tlist[l].name,temp,11)) == 0) test = 1;
   if (test == 1) {
      printf("\n That name already exists in the training file.");
      prtc();
      break;
   }

   for (l=1; l<(k+1); l++)
      if ((strncmp(ilist[l].name,temp,11)) == 0) test = 2;
   if (test != 2) {
      printf("\n There are no image files with that name.");
      prtc();
      break;
   }

   printf("\n\n You must enter 4 valid (and unique)");
   printf(" file version numbers.");
   printf("\n (Enter 0 (zero) to quit training procedure)");

   for (j=1; j<5; j++) {
      printf("\n Enter version number for training file # ");
      printf("%d%s",j,">");
      scanf("%s",ch);
      ver[j] = ch[0] - '0';
```

```
   if (ver[j] > 0) {
      test = 0;
      for (l=1; l<j; l++) {
         if (ver[l] == ver[j]) {
            display(ilist,k,6);
            printf("\n\n You already selected:");
            for (m=1; m<j; m++) printf("%s%d"," ",ver[m]);
            j = j - 1;
            test = 1;
      }}
      if (test != 1) {
         for (l=1; l<(k+1); l++) {
            if (strncmp(ilist[l].name,temp,11) == 0
             && ilist[l].num == ver[j]) {
                test = 1;
                l = k + 2;
         }}
         if (test == 0) {
            display(ilist,k,6);
            printf("%s%d","\n\n File version #",ver[j]);
            printf(" not found, try another.");
            if (j != 1) {
               printf(" (You already selected:");
               for (l=1; l<j; l++) printf("%s%d"," ",ver[l]);
               printf(")");
            }
            j = j - 1;
   }}}

   else j = 5;
}

if (j == 6) break;

for (j=1; j<5; j++) {
   i = i + 1;
   for (l=1; l<(k+1); l++) {
      if (strncmp(ilist[l].name,temp,11) == 0
       && ilist[l].num == ver[j]) {
          tlist[i] = ilist[l];
                                 /* find proper gestalt file in ilist, */
          tlist[i].num = j;                         /* put in tlist, */
          for (m=1; m<k; m++) ilist[m] = ilist[m+1];
          k = k - 1;                              /* delete from ilist, */
   }}

   t2[0] = '\0';
   t3[0] = '\0';
   strcat(t2.dbasedir);
   strcat(t3,imagedir);
   strcat(t2,temp);
   strcat(t3,temp);
   strcat(t2,".pic;\0");
```

```
            strcat(t3,".img;\0");
            ch[1] = '\0';
            ch[0] = ver[j] + '0';
            strcat(t3,ch);
            ch[0] = j + '0';
            strcat(t2,ch);
            copyfile(t3,t2);
            delete(t3);
         }

         printf("%s%s%s","\n\n The training file now contains <",temp,">");
         printf("(Moved from image data base)");
         train_changed = 't';
         others_changed = 't';
         prtc();
         break;

      case 8:
         menu3();
         break;

      case 9:
         printf("Enter the new variable threshold >");
         scanf("%d",&thr);
         break;

      default:
         break;
   }}
   }
```

```
/*************************************************************************/
menu2()                                                        /* 287 */
{

    int cam;
    char name[80],t1[80],t2[80],temp[80];

for (;;) {

   cls();
   printf("\n                    ***** ACQUISITION OF IMAGES *****\n");
   printf("\n                        0:RETURN TO MAIN MENU");
   printf("\n                        1:STATIONARY TARGET");
   printf("\n                        2:MOVING TARGET");
   printf("\n                        3:LOAD IMAGE FROM MEMORY");
   printf("\n                        4:SAVE IMAGE IN DATABASE");
   printf("\n                        5:SET CAMERA PORT");
   printf("\n                        6:CAMERA CHECK");
   printf("\n                        7:RE-INITIALIZE HARDWARE");
   printf("\n                        8:LOAD LARGE IMAGE FROM MEMORY");
   printf("\n\n\n\n\n\n<");
   scanf("%d",&option);
   cls();

   switch (option) {

   case 0:
      return;

   case 1:
      nf = sx = sy = 0;
      fx = fy = 511;
      getchar();
      printf("\n                    ***** STATIONARY TARGET *****");
      printf("\n\n Acquire new image (Y/N)? >");
      scanf("%s",temp);
      if (temp[0] == 'Y' || temp[0] == 'y') {
         grab(0);
         prtc();
         stopgrab(1);
      }
      break;

   case 2:
      nf = 0;  /* this algorithm sets sx,sy,fx,fy to target's location */
      printf("\n                    ***** MOVING TARGET *****\n\n");
      getchar();
      printf("\n Prepare background image");
      printf(" and press RETURN to continue. >");
      waitvb();             /* The aclear() is used in this routine to */
      grab(0);              /* clear the 1st 16 columns of the image   */
      getchar();            /* because of an X_SPIN delay of the image.*/
      stopgrab(1);          /* Therefore the 256x512 image is really   */
```

```
    setreg(X_SPIN,0);     /* only a 256x496 image.                        */
    snap(1);
    aclear(0,0,16,768,0);
    setreg(SCROLL,256);
    printf("\n\n Prepare subject image");
    printf(" and press RETURN to continue. >");
    waitvb();         /* Scrolling 256 stores the background image      */
    grab(0);          /* off the screen area.  Scroll 0 brings it       */
    getchar();        /* back. I have used the setreg function instead */
    stopgrab(1);      /* of scroll because of the problem with defini- */
    setreg(X_SPIN,0); /* tions in the library. (see the comment        */
    snap(1);            /* obtained when linking this program).        */
    aclear(0,0,16,512,0);
    setreg(SCROLL,0);
    printf("\n\n Subtracting images and locating target.");
    oparea(0,256,512,255,0,0,512,255,5,1);
    if (isolate(20,8,16) == 0) { /* Isolate target from surroundings */
       printf("\n Could not find target. Press RETURN to continue. >");
       getchar();
       sx = sy = 0;
       fx = fy = 511;
       break;
    }
    carea(sx+1,sy+257,fx-sx-1,fy-sy-1,sx+1,sy+1,fx-sx-1,fy-sy-1);
    aclear(0,255,512,256,0);
    break;

 case 3:
    printf("\n                          ***** LOAD IMAGE FROM MEMORY *****");
    printf("\n\n\n Enter complete file specification.");
    printf("\n (filename.ext;version)\n\n >");
    scanf("%s", name);
    t2[0] = '\0';
    strcat(t2,imagedir);
    strcat(t2,name);
    printf("%s\n",t2);
    printf("\n\n\n Loading file...");
    if (readim(200,30,200,200,t2,"nocomm") == -1) {
       printf("\n\n\n File not found.");
       prtc();
    }
    else text(200,10,0,1,200,"          \0");
    nf = sx = sy = 0;
    fx = fy = 511;
    break;

 case 4:
    t1[0] = '\0';
    strcat(t1,imagedir);
    printf("\n Will save ******ENTIRE****** screen as 8-bit image in
");
    printf(" pixel data base.\n\n\n Enter name (including EXT)");
    printf("/n  (Enter Quit to exit)\n\n>");
```

```
      scanf("%s", name);      /* I want to make sure that the DBASE     */
      if(strcmp(name,"Quit\0") == 0) break;
      strncat(t1, name,11);  /* directory is not touched by this save. */
      printf("\n\n\n\n Saving image...");
                                       /* Hence the directory name is */
      saveim(0,0,511,480,0,t1,"nocomm");  /* not allowed to vary.      */
      break;

   case 5:
      printf("\n Select camera port (0,1 or 2) >");
      scanf("%d",&cam);
      if (cam == 0 || cam == 1 || cam == 2)
         setcamera(cam);
      break;

   case 6:
      printf("\n\n                        ***** CAMERA CHECK *****\n\n");
      grab(0);
      prtc();
      stopgrab(1);
      nf = sx = sy = 0;
      fx = fy = 511;
      break;

   case 7:
      initialize();
      nf = sx = sy = 0;
      fx = fy = 511; /****************************************************/
      sclear(100);    /* using 100 gives a clean screen that is not too */
      break;          /* dark to tell whether the monitor is on/off     */

   case 8:
      printf("\n                     ***** LOAD IMAGE FROM MEMORY *****");
      printf("\n\n\n Enter complete file specification.");
      printf("\n (filename.ext;version)\n\n >");
      scanf("%s", name);
      t2[0] = '\0';
      strcat(t2,imagedir);
      strcat(t2,name);
      printf("%s\n",t2);
      printf("\n\n\n Loading file...");
      if (readim(0,0,511,480,t2,"nocomm") == -1) {
         printf("\n\n\n File not found.");
         prtc();
      }
      else text(200,10,0,1,200,"          \0");
      nf = sx = sy = 0;
      fx = fy = 511;
      break;

   default:          /* This is to prevent accidently leaving it on.   */
      break;         /**************************************************/
}}
```

```
}

/**************************************************************************/
menu3()                                                          /* 402 */
{

    char temp[80],ch[80],t3[80];
    int version,j;

for (;;) {

   cls();
   printf("\n                          ***** DEMONSTRATION *****\n");
   printf("\n                          0:RETURN TO MAIN MENU");
   printf("\n                          1:IDENTIFY A PERSON");
   printf("\n                          2:TOTAL SYSTEM\n\n\n\n\n\n\n>");

   scanf("%d",&option);

   switch (option) {

   case 0:
      return;

   case 1:
      display(ilist,k,6);
      printf("\n\n                    ***** IDENTIFY A PERSON *****");
      printf("\n\n Enter person's name (or Quit). >");
      scanf("%s",temp);
      if(strcmp(temp,"Quit\0") == 0) break;
      printf("\n\n Enter version number. >");
      scanf("%s",ch);
      printf("\n");

      test = -1;
      for (j=1; j<(k+1); j++) {
         if (strncmp(ilist[j].name,temp,11) == 0) {
            if (ilist[j].num == (ch[0] - '0')) {
               t3[0] = '\0';
               strcat(t3,imagedir);
               strncat(t3,ilist[j].name,11);
               strcat(t3,".img;\0");
               strcat(t3,ch);
               test = j;
               j = k + 2;
      }}}
      if (test == -1) {
         printf("\n\n Image file not found.");
         prtc();
      }
      else {
         nf = sx = sy = 0;
         fx = fy = 511;
```

```
            sclear(0,1);
            readim(200,30,200,200,t3,"nocomm");
            text(200,10,0,1,200,temp);
            recognize(test);
         }
         break;

      case 2:
         afrm();
         break;

      default:
         break;
   }}
   }
```

```
/*************************************************************************/
static int pix,avg,diff,neigh,threshold,ne,nn,nm;                  /* 469 */
static int col[512];
struct image{
       int data[512];
};
struct feat{
       int sx,sy,fx,fy,xcenter,ycenter,pix,xsize,ysize,used;
};
struct whole{
       int x,y,dx,dy,leye,reye,teye,beye,tnose,cmouth;
       int center,xellipse,yellipse,radius;
};
static struct image  pic[512],norm[512],temp[512];
static struct feat   eye[100],nose[100],mouth[100];
static struct whole  face[10];

/************************************************************************/
int facemap()                                                      /* 485 */
{

     int i,j,k,l;
     char name[30];

del();
cls();
printf("\n processing image...");
bright_norm();
ne = nn = nm = nf = 0;
featuremap();

for (i=1; i<ne; i++) {
 if (eye[i].used == 0) {
  for (j=1; j<ne+1; j++) {                     /* look for a matching eye */
   if (eye[j].sx > eye[i].fx && eye[j].used == 0) {       /* try eye[j] */
    if (abs(eye[j].pix - eye[i].pix) < eye[j].pix/2) {  /* pix numbers */
     if (eye[j].ycenter > eye[i].sy &&                         /* okay */
         eye[j].ycenter < eye[i].fy) {               /* close in height */
      if (eye[j].sx < eye[i].fx+2*eye[i].xsize) {        /* near enough */
       for (k=1; k<nn+1; k++) {                      /* look for a nose */
        if (nose[k].sy > eye[i].fy && nose[k].used == 0) {
                                                       /* try nose[k] */
         if (nose[k].xcenter > eye[i].sx &&
             nose[k].xcenter < eye[j].fx) {             /* between eyes */
          for (l=1; l<nm+1; l++) {                  /* look for a mouth */
           if (mouth[l].ycenter > nose[k].fy &&
               mouth[i].used == 0) {                      /* below nose */
            if (mouth[l].ycenter < eye[i].fy+4*eye[i].ysize) {
                                                        /* near enough */
             if (mouth[l].xcenter > eye[i].sx &&
                 mouth[l].xcenter < eye[j].fx) {        /* between eyes */
              nf = nf+1;
                  /* all features found and conditions met for a face. */
```

```
                eye[i].used = eye[j].used = 1;
                nose[k].used = mouth[l].used = 1;
                face[nf].dx = 9*(eye[j].xcenter - eye[i].xcenter)/4;
                face[nf].dy = 2*(mouth[l].ycenter - eye[i].sy);
                face[nf].x = (eye[j].xcenter+eye[i].xcenter)/2
                       - face[nf].dx/2;
                face[nf].y = mouth[l].ycenter - 4*face[nf].dy/5;
                face[nf].leye = eye[i].sx - face[nf].x;
                face[nf].reye = eye[j].fx - face[nf].x;
                face[nf].teye = (eye[i].sy + eye[j].sy)/2 - face[nf].y;
                face[nf].beye = (eye[i].fy + eye[j].fy)/2 - face[nf].y;
                face[nf].tnose = nose[k].sy - face[nf].y;
                face[nf].cmouth = mouth[l].ycenter - face[nf].y;
                face[nf].center = face[nf].dx/2;
                face[nf].xellipse = face[nf].dx/2 + face[nf].x;
                face[nf].yellipse = face[nf].dy/2 + face[nf].y;
                face[nf].radius = face[nf].dx;
                circle(face[nf].xellipse,face[nf].yellipse,
                       face[nf].radius,1,2,255);
                rectangle(eye[i].sx-1,eye[i].sy-1,eye[j].fx - eye[i].sx,
                          mouth[l].ycenter - eye[i].sy,255);
                l = k = j = 500;
}}}}}}}}}}}}}}

if (nf == 0) return(0);
printf("\n Saving brightness normalized faces to disk...");
for (y=0; y<480; y++) whline(0,y,512,norm[y].data);
name[0] = '\0';
strcat(name,"bnorm.img\0");

for (i=1; i<nf+1; i++) {
 printf("\n %s%s%d",name,";",i);
/* changed 255 to 0 for fill between ellipse and rectangle */
 circle(face[i].xellipse,face[i].yellipse,face[i].radius,1,2,255);
 rectangle(face[i].x,face[i].y,face[i].dx,face[i].dy,255);
 fill(face[i].x+1,face[i].y+1,50,255);
 fill(face[i].x+1,face[i].y+1,0,255);
 fill(face[i].x+face[i].dx-1,face[i].y+face[i].dy-1,50,255);
 fill(face[i].x+face[i].dx-1,face[i].y+face[i].dy-1,0,255);
 fill(face[i].x+1,face[i].y+face[i].dy-1,50,255);
 fill(face[i].x+1,face[i].y+face[i].dy-1,0,255);
 fill(face[i].x+face[i].dx-1,face[i].y+1,50,255);
 fill(face[i].x+face[i].dx-1,face[i].y+1,0,255);
 circle(face[i].xellipse,face[i].yellipse,face[i].radius,1,2,0);
 rectangle(face[i].x,face[i].y,face[i].dx,face[i].dy,0);
 saveim(face[i].x,face[i].y,face[i].dx,face[i].dy,0,name,"nocomm");
}

printf("\n Also saving original faces...");
for (y=0; y<480; y++) whline(0,y,512,temp[y].data);
name[0] = '\0';
strcat(name,"orig.img\0");
```

```
for (i=1; i<nf+1; i++) {
 printf("\n %s%s%d",name,";",i);
 rectangle(face[i].x,face[i].y,face[i].dx,face[i].dy,255);
 saveim(face[i].x,face[i].y,face[i].dx,face[i].dy,0,name,"ndcolor");
}

return(1);
}
```

```
/*************************************************************************/
featuremap()                                                      /* 563 */
{

 int fill,test,ymin,ymax,xmax,i,j,dy,dx,ytest,xtest,bx;
 char type;

for (y=sy+14; y<fy-14; y++) {   /* begin and end with 14 pixel margins */
 test = 0;
 for (x=sx; x<fx-14; x++) {   /* see if line is touching top of object */
  if (pic[y+1].data[x] == 0) {      /* these checks are done like this */
   if (pic[y].data[x] == 100) {                           /* for speed. */
    if (pic[y].data[x-1]+pic[y].data[x+1] == 200) {
     if (pic[y].data[x-2]+pic[y].data[x+2] == 200) {
      if (pic[y].data[x-3]+pic[y].data[x+3] == 200) {
       if (pic[y].data[x-4]+pic[y].data[x+4] == 200) {
        if (pic[y].data[x-5]+pic[y].data[x+5] == 200) {
         test = 1;
         bx = x - 50;
         if (bx < 14) bx = 14;
         x = 512;
 }}}}}}}}

 if (test == 1) {            /* okay, for this line find the object(s) */
  for (x=bx; x<498; x++) {
   test = 0;
   type = 'u';
   if (pic[y].data[x] == 100) {                      /* possible corner */
    ymax = y + 40;
    if (ymax>479) ymax = 479;
    for (i=y; i<ymax+1; i++) {              /* how far is line white? */
     if (pic[i].data[x] == 0) {
      ymax = i - 1;
      i = 512;
    }}
    if (ymax > y+1) {
     for (i=y+1; i<ymax; i++) {
      if (pic[i].data[x+1] == 0) {          /* something touching line */
       dy = i;
       test = 1;
       i = 512;
   }}}}

   if ( test == 1) {                                    /* left side ok */
    xmax = x + 50;
    if (xmax>498) xmax = 498;
    for (i=x; i<xmax+1; i++) {              /* how far is line white? */
     if (pic[y].data[i] == 0) {
      xmax = i-1;
      i = 512;
    }}
    test = 0;
```

```
 if (xmax > x+1) {
  for (i=x+1; i<xmax; i++) {
   if (pic[y+1].data[i] == 0) {          /* something touching line */
    dx = i;
    test = 1;
    i = 512;
}}}}

if (test == 1) {                   /* at the border of unknown object */
 test = 0;
 while (dx < xmax+1 && dy < ymax+1 && test == 0) {
  ytest = 0;
  while (dy < ymax+1 && ytest == 0) {      /* try to go across to dx */
   ytest = 1;                                     /* assume success */
   for (i=x; i<dx+1; i++)
    if (pic[dy].data[i] == 0) ytest = 0;
   if (ytest == 0) dy = dy + 1;
  }
  if (ytest == 1) {
   xtest = 0;
   while (dx<xmax+1 && xtest == 0) {          /* try to go down to dy */
    xtest = 1;                                    /* assume success */
    for (i=y; i<dy+1; i++)
     if (pic[i].data[dx] == 0) xtest = 0;                 /* failed */
    if (xtest == 0) dx = dx + 1;
   }
   if (xtest == 1) {                          /* recheck present dy */
    for (i=x; i<dx+1; i++)
     if (pic[dy].data[i] == 0) ytest = 0;                 /* failed */
    if (xtest == 1 && ytest == 1) test = 1;
}}}}

if (test == 1) {                   /* successfully blocked in object */
 if ((dy-y) > 3*(dx-x)) type = 't';             /* too tall and thin */
 if ((dx-x) < 7) type = 't';                            /* too small */
}
if (test == 1 && type == 'u') {
 fill = 0;
 for (j=y+1; j<dy; j++)
  for (i=x+1; i<dx; i++) if (pic[j].data[i] == 0) fill++;
 if (fill < (dx-x)*(dy-y)*3/10) test = 0;    /* less than 30% solid */
}
if (test == 1 && type == 'u') {
 if (dx-x > 2*(dy-y)) {                            /* possible mouth */
  rectangle(x,y,dx-x,dy-y,0);
  type = 'm';
  nm = nm + 1;
  mouth[nm].xcenter = (dx+x)/2;
  mouth[nm].ycenter = (dy+y)/2;
  mouth[nm].sy = y;
  mouth[nm].used = 0;
}}
```

```
   if (test == 1 && type != 't') {
    fill = 0;
    ymax = dy+(2*(dy-y)/3);
    if (ymax < 480) {
     for (j=dy+1; j<ymax; j++)                /* chk for space below */
      for (i=x; i<dx; i++) if (pic[j].data[i] == 0) fill++;
     if (fill<(ymax-dy+1)*(dx-x)*10/100) {
                                           /* less than 10% of area filled */
      type = 'e';
      ne = ne + 1;
      eye[ne].xcenter = (dx+x)/2;
      eye[ne].ycenter = (dy+y)/2;
      eye[ne].pix = (dx-x) * (dy-y);
      eye[ne].xsize = dx - x;
      eye[ne].ysize = dy - y;
      eye[ne].sx = x;
      eye[ne].fx = dx;
      eye[ne].sy = y;
      eye[ne].fy = dy;
      eye[ne].used = 0;
      rectangle(x,y,dx-x,dy-y,0);           /* (1x1 <= size <= 20x20) */
    }}
    fill = 0;
    ymin = y-(dy-y);
    if (ymin > 0) {
     for (j=ymin; j<y; j++)                   /* chk for space above */
      for (i=x; i<dx; i++) if (pic[j].data[i] == 0) fill++;
     if (fill < (y-ymin)*(dx-x)*10/100) {
                                           /* less than 10% of area filled */
      nn = nn + 1;
      nose[nn].xcenter = (dx+x)/2;
      nose[nn].ysize = dy - y;
      nose[nn].fy = dy;
      nose[nn].sy = y;
      nose[nn].pix = (dx-x) * (dy-y);
      nose[nn].used = 0;
      rectangle(x,y,dx-x,dy-y,0);           /* (1x1 <= size <= 20x20) */
}}}}}}

return;
}
                                                                  /* 767 */
```

```
/***************************************************************************/
static double cray[65][65],crayr[65][65],crayi[65][65],rinp[65];
static int ix,iy;

/***************************************************************************/
gestalt(m)                                  /* Values range from 0 to 128  */

int m;                                                /* m = face number */
 {
       int x,y,i,j;

line(256,0,256,512,0);
line(0,256,512,256,0);
line(384,0,384,512,0);
line(128,256,128,512,0);

/* left half: whole head */
carea(sx,sy,face[m].dx/2,face[m].dy,270,sy,face[m].dx/2,face[m].dy);
/* right half: whole head */
carea(sx+face[m].dx/2,sy,face[m].dx/2,face[m].dy,
                                     400,sy,face[m].dx/2,face[m].dy);
/* top half: top to tnose */
carea(sx,sy,face[m].dx,face[m].tnose,15,sy+256,face[m].dx,face[m].tnose)
;
/* internal features */
carea(sx+face[m].leye,sy+face[m].teye,face[m].reye-face[m].leye,
                                         face[m].cmouth-face[m].teye,
      140+face[m].leye,sy+256+face[m].teye,face[m].reye-face[m].leye,
                                        face[m].cmouth-face[m].teye);
/* left internal features */
carea(sx+face[m].leye,sy+face[m].teye,face[m].center-face[m].leye,
                                         face[m].cmouth-face[m].teye,
      270+face[m].leye,sy+256+face[m].teye,face[m].center-face[m].leye,
                                        face[m].cmouth-face[m].teye);
/* bottom half: tnose to chin */
carea(sx,sy+face[m].tnose,face[m].dx,face[m].dy-face[m].tnose,
      400,sy+256+face[m].tnose,face[m].dx,face[m].dy-face[m].tnose);

line(sx,sy,sx+face[m].dx,sy,0);                                   /*top*/
line(sx+face[m].dx,sy,sx+face[m].dx,sy+face[m].dy,0);            /*right*/
line(sx+face[m].dx,sy+face[m].dy,sx,sy+face[m].dy,0);           /*bottom*/
line(sx,sy+face[m].dy,sx,sy,0);                                  /*left*/
line(sx,sy+face[m].teye,sx+face[m].dx,sy+face[m].teye,0);        /*teye*/
line(sx,sy+face[m].cmouth,sx+face[m].dx,sy+face[m].cmouth,0); /*cmouth*/
line(sx,sy+face[m].tnose,sx+face[m].dx,sy+face[m].tnose,0);     /*tnose*/
line(sx+face[m].leye,sy,sx+face[m].leye,sy+face[m].dy,0);        /*leye*/
line(sx+face[m].center,sy,sx+face[m].center,sy+face[m].dy,0); /*center*/
line(sx+face[m].reye,sy,sx+face[m].reye,sy+face[m].dy,0);        /*reye*/
ix = face[m].dx/2;
iy = face[m].dy/2;

printf("\n calculating fft for window 1.");
clear_cray();                                /* left half: whole head */
```

```
for (y=sy; y<sy+face[m].dy; y+=2)
   for (x=270; x<270+face[m].dx/2; x+=1) {
      crayr[x-269][1+(y-29)/2] = (double) brpixel(x,y)/255;
   }
fft2(crayr,crayi,64,0);
save_fft(0);

printf("\n calculating fft for window 2.");
clear_cray();                                   /* right half: whole head */
for (y=sy; y<sy+face[m].dy; y+=2)
   for (x=400; x<400+face[m].dx/2; x+=1) {
      crayr[(x-399)+face[m].dx/4][1+(y-29)/2] = (double) brpixel(x,y)/255;
   }
fft2(crayr,crayi,64,0);
save_fft(1);

printf("\n calculating fft for window 3.");
clear_cray();                                   /* top half: top to tnose */
for (y=sy+256; y<sy+256+face[m].tnose; y+=1)
   for (x=15; x<15+face[m].dx; x+=2) {
      crayr[1+(x-14)/2][y-285] = (double) brpixel(x,y)/255;
   }
fft2(crayr,crayi,64,0);
save_fft(2);

printf("\n calculating fft for window 4.");
clear_cray();                                       /* internal features */
for (y=sy+256; y<sy+256+face[m].cmouth; y+=2)
   for (x=140; x<140+face[m].reye; x+=2) {
      crayr[1+(x-139)/2][1+(y-285)/2] = (double) brpixel(x,y)/255;
   }
fft2(crayr,crayi,64,0);
save_fft(3);

printf("\n calculating fft for window 5.");
clear_cray();                                   /* left internal features */
for (y=sy+256; y<sy+256+face[m].cmouth; y+=2)
   for (x=270; x<270+face[m].dx/2; x+=1) {
      crayr[x-269][1+(y-285)/2] = (double) brpixel(x,y)/255;
   }
fft2(crayr,crayi,64,0);
save_fft(4);

printf("\n calculating fft for window 6.");
clear_cray();                               /* bottom half: tnose to chin */
for (y=sy+256; y<sy+256+face[m].dy; y+=1)
   for (x=400; x<400+face[m].dx; x+=2) {
      crayr[1+(x-399)/2][y-285] = (double) brpixel(x,y)/255;
   }
fft2(crayr,crayi,64,0);
save_fft(5);

return;
```

```
}

/*************************************************************************/
save_fft(index)

  int index;

{
  int x,y;

for (x=0;x<5;x++)
   for (y=0;y<5;y++)
      ilist[0].feature[index][x][y] = 0.0;
for (x=0;x<3;x++)
   for (y=0;y<3;y++)
      ilist[0].feature[index][x+2][y+2] = crayr[x][y];
for (x=0;x<2;x++)
   for (y=0;y<2;y++)
      ilist[0].feature[index][x+3][y] = crayr[x+1][y+62];
for (x=0;x<3;x++)
   for (y=0;y<2;y++)
      ilist[0].feature[index][x][y] = crayi[2-y][2-x];
ilist[0].feature[index][0][2] = crayi[0][2];
ilist[0].feature[index][1][2] = crayi[0][1];
for (x=0;x<2;x++)
   for (y=0;y<2;y++)
      ilist[0].feature[index][1-y][4-x] = crayi[x+62][y+1];

return;
}

/*************************************************************************/
clear_cray()
{

     int x,y;

for (y=0; y<65; y++)
  for (x=0; x<65; x++) {
    cray[x][y] = 0.0;
    crayr[x][y] = 0.0;
    crayi[x][y] = 0.0;
  }
return;
}
```

```c
/**********************************************************************/

fft2(picc,ipicc,n,dir)

double picc[65][65];
double ipicc[65][65];
int n;
int dir;

{

     double pic[65];
     double ipic[65];
     int i,j;

     for (i = 0; i < n; i++)
          {
          for (j = 0; j < n; j++)
               {
               pic[j+1] = picc[i][j];
               ipic[j+1] = ipicc[i][j];
               }
          fft(pic,ipic,n,dir);

          for (j = 0; j < n; j++)
               {
               picc[i][j] = pic[j+1];
               ipicc[i][j] = ipic[j+1];
               }
          }

     for (j = 0; j < n; j++)
          {
          for (i = 0; i < n; i++)
               {
               pic[i+1] = picc[i][j];
               ipic[i+1] = ipicc[i][j];
               }
          fft(pic,ipic,n,dir);

          for (i = 0; i < n; i++)
               {
               picc[i][j] = pic[i+1];
               ipicc[i][j] = ipic[i+1];
               }
          }

     return;
 }
```

```
fft(fr,fi,n,dir)
double fr[65];
double fi[65];
int n;
int dir;

{
     double tr=0, ti=0;
     double wr=0, wi=0;
     double el=0, a=0;
     int i=0, j=0;
     int mr=0;
     int l=0;
     int k=0;
     int m=0, nn=0;
     int step=0;


     if (dir < 0 )
          for (i = 1; i < n+1; i++)
               fi[i] = -fi[i];
     mr = 0;
     nn = n - 1;
     for (m = 1; m <= nn; m++)
          {
          l = n/2;
          while (l + mr > nn) l = l/2;
          mr = mr%l + l;
          if (mr > m )
               {
               tr = fr[m+1];
               fr[m+1] = fr[mr+1];
               fr[mr+1] = tr;

               ti = fi[m+1];
               fi[m+1] = fi[mr+1];
               fi[mr+1] = ti;
               }
          }
     l = 1;
     while (l < n)
          {
          step = 2*l;
          el = l;
          for (m = 1; m <= l; m++)
               {
               a = 3.1415926535 * (double) (1-m)/ el;
               wr = cos(a);
               wi = sin(a);
               for (i = m; i <= n; i += step)
                    {
                    j = i + l;
                    k = i;
```

```
                    tr = wr*fr[j] - wi*fi[j];
                    ti = wr*fi[j] + wi*fr[j];
                    fr[j] = fr[k] - tr;
                    fi[j] = fi[k] - ti;
                    fr[k] = fr[k] + tr;
                    fi[k] = fi[k] + ti;
                    }
               }
          l = step;
          }

     if ( dir < 0)
          for (i = 1; i < n+1; i++)
               {
               fr[i] = fr[i]/n;
               fi[i] = -fi[i]/n;
               }

     return;
}
```

/* 697 */

```
/*************************************************************************/
bright_norm()
{                      /* norm will contain brightness normalized scene  */
                       /* (bright areas set to 128, dark areas= 128-diff */
int i,j;               /* pic will contain the dark objects of the scene */
                             /* (uses variable threshold, binary output) */
sx = sx - 14;
if (sx < 0) sx = 0;
sy = sy - 14;
if (sy < 0) sy = 0;
fx = fx + 14;
if (fx > 512) fx = 512;
fy = fy + 14;
if (fy > 480) fy = 480;
for (y=0; y<480; y++) for (x=0; x<512; x++) norm[y].data[x] = 0;
for (y=0; y<480; y++) rhline(0,y,512,pic[y].data);
y = sy;

for (i=sx; i<fx; i++) {
 col[i] = 0;                     /*  setup all columns for first y value */
 for (j=y; j<y+30; j++) col[i] += pic[j].data[i];
}

for (y=sy+1; y<fy-30; y++) {      /* now all columns calculated faster  */
 for (i=sx; i<fx; i++) col[i] += (pic[y+29].data[i] - pic[y-1].data[i]);
 x = sx;
 neigh = 0;                                /* setup first neighborhood */
 for (i=x; i<x+30; i++) neigh += col[i];
 for (x=sx+1; x<fx-30; x++) { /* now all other neigh calculated faster */
  neigh += (col[x+29] - col[x-1]);
  avg = neigh/900;
  pix = pic[y+14].data[x+14];

  if (pix < avg) norm[y+14].data[x+14] = 128 - (avg - pix);
  else norm[y+14].data[x+14] = 128;
  if (norm[y+14].data[x+14] < 0) norm[y+14].data[x+14] = 0;

/* added variable threshold */
  threshold = 80*avg/100 + thr;       /* add 7 because noise is +/- 7 */
  if (pix < threshold) temp[y+14].data[x+14] = 0;          /* dark=0 */
  else temp[y+14].data[x+14] = 100;                /* else light=100 */
}}

for (y=sy+14; y<fy-14; y++) {                       /* cleanup noise */
 for (x=sx+14; x<fx-14; x++) {
  pic[y].data[x] = temp[y].data[x];
  if (temp[y].data[x] == 0) {
   if (temp[y].data[x-1]+temp[y].data[x+1]+temp[y].data[x+2] > 0) {
    pic[y].data[x] = 100;
}}}}

for (y=sy+14; y<fy-14; y++) {
 for (x=sx+14; x<fx-14; x++) {
```

```
  if (pic[y].data[x] == 0) {
   if (pic[y].data[x-1]+pic[y-1].data[x]+pic[y+1].data[x]+
       pic[y].data[x+1] > 200) {
    pic[y].data[x] = 100;
}}}}

for (y=fy-14; y>sy+14; y--) {
 for (x=fx-14; x>sx+14; x--) {
  if (pic[y].data[x] == 0) {
   if (pic[y].data[x-1]+pic[y-1].data[x]+pic[y+1].data[x]+
       pic[y].data[x+1] > 200) {
    pic[y].data[x] = 100;
}}}}

for (y=0; y<480; y++) rhline(0,y,512,temp[y].data);
for (y=0; y<480; y++) whline(0,y,512,pic[y].data);
return;
}
```

```
/*********************************************************************/
cont_enhance(m)

     int m; /* face number */

{
     int x,y,z;

  static_luts();
  setlut(RED,5);
  histeq(RED,5,face[m].leye+sx+1,face[m].beye+sy,
         face[m].dx/2 - 2,face[m].cmouth-face[m].beye);
  maplut(RED,5,0,0,256,256);
  linlut(RED,5);
  linlut(GREEN,5);
  linlut(BLUE,5);

  for (y=sy; y<sy+face[m].dy; y++) {                /* threshold result */
   for (x=sx; x<sx+face[m].dx; x++) {
    z = brpixel(x,y);                            /* leave dark areas but */
    if (z < 50) bwpixel(x,y,z);                  /* make skin pure white */
    else bwpixel(x,y,255);
  }}

 return;
 }
```

```
/***********************************************************************/
scale(m)

    int m;

{
    int fact;

  if ((fx-sx)/150 > (fy-sy)/150) fact = 150/(fx-sx);
  else fact = 150/(fy-sy);
  if (fact > 1) {
   repzoom(sx,sy,fx-sx,fy-sy,sx,sy,200,200,fact,fact);
   face[m].dx = face[m].dx * fact;   /* update face[m].lines by 'fact' */
   face[m].dy = face[m].dy * fact;
   face[m].leye = face[m].leye * fact;
   face[m].reye = face[m].reye * fact;
   face[m].teye = face[m].teye * fact;
   face[m].beye = face[m].beye * fact;
   face[m].tnose = face[m].tnose * fact;
   face[m].cmouth = face[m].cmouth * fact;
   face[m].center = face[m].center * fact;
   fx = (fx-sx)*fact + sx;                /* update fx,fy by 'fact' */
   fy = (fy-sy)*fact + sy;
  }

  aclear(0,0,512,sy,255);
  aclear(0,fy,512,480-fy,255);
  aclear(0,sy-1,sx,fy-sy+1,255);
  aclear(fx,sy-1,512-fx,fy-sy+1,255);

 return;
 }
```

```
/***************************************************************************/
facerec(version)

int version;

{
  char ch[80],t2[30],t3[80],t4[30],t5[80];
  int i,j,w,w1,w2,l,m,n,p,dx,dy;

if (nf != 0) {
 cls();
 printf(" trying to recognize faces found...");
 for (m=1; m<nf+1; m++) {
  t2[0] = '\0';                          /* create file names for face # m */
  t4[0] = '\0';
  strcat(t2,"bnorm.img;\0");
  strcat(t4,"orig.img;\0");
  t3[0] = m + '0';
  t3[1] = '\0';
  strcat(t2,t3);
  strcat(t4,t3);
  sx = 60;
  sy = 30;
  printf("\n %s",t2);
  sclear(0,1);
  readim(sx,sy,200,200,t2,"nocomm");       /* display bright_norm face */
  l = sy;
  while(brpixel(sx,l) != 0) l++;                  /* get fx,fy values */
  fy = l - 1;
  l = sx;
  while(brpixel(l,sy) != 0) l++;
  fx = l - 1;
  dx = fx - sx;
  dy = fy - sy;
/*  cont_enhance(m);
  scale(m); */
  text(70,10,0,1,0,t2);
  gestalt(m);                        /* gestalt values put in ilist[0] */
  initialize();
  sclear(0,1);
  readim(200,30,200,200,t4,"nocomm");       /* display original face */
  text(200,10,0,1,200,t4);
  if (version == 1) {
   printf("\n Save in dbase? (Y/N) >");
   scanf("%s",ch);
   if (ch[0] == 'y' || ch[0] == 'Y') {
     printf("\n enter name of subject (up to 10 letters) \n>");
     scanf("%s",t3);
     p = 0;            /* highest existing version # for this subject */
     for (n=1; n<k+1; n++) {
       if (strncmp(ilist[n].name,t3,11) == 0) {
         p = ilist[n].num;
       }
```

```
      }
      k = k + 1;
      p = p + 1;
      ilist[k].name[0] = '\0';
      strncat(ilist[k].name,t3,11);
      ilist[k].num = p;
      for (w=0; w<6; w++)
        for (w1=0; w1<5; w1++)
          for (w2=0; w2<5; w2++)
            ilist[k].feature[w][w1][w2] = ilist[0].feature[w][w1][w2];
      others_changed = 't';
      t5[0] = '\0';
      strcat(t5,imagedir);
      strncat(t5,ilist[k].name,11);
      strcat(t5,".img;\0");
      saveim(200,30,200,200,0,t5,"nocomm");
    }
  }
  else {
    if (m < nf) {
       printf("\n Forget about rest of ");
       printf("faces and return to main menu? (Y/N) >");
       scanf("%s",ch);
       if (ch[0] == 'y' || ch[0] == 'Y') {
         return;
       }
     }
   }
   recognize(0);                    /* pass in gestalt values of ilist[0] */
   delete(t2);
   delete(t4);
   cls();
}}
else {
   printf("\n face not found.");
   prtc();
}
nf = sx = sy = 0;
fx = fy = 511;
return;
}
```

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

static int resultsl[257][5];
static int list[101];    /* list of id#s ordered by distances in list2 */
static double t[101],list2[101];  /* total distances (for all windows) */
static double v[101][7];     /* v[id][w] = distance from person #id to
                    unknown person for window #w  (Russel, 1985:4-40a) */

/**********************************************************************/
recognize(num)                       /*from REMID.FR 06/03/86 by R. Russel */

 int num;        /* the position in ilist[] of gestalt values to use. */

{
char t8[80];
double gix,gux,sigix,c,most;
int id,w,x,y,m,n,j,confid,test;
double p[7] = {1.0,1.0,1.0,1.0,1.0,1.0,6.0}; /* window performance factors
          (update after training and testing with sufficient samples */
                                 /* note: p[0] is used for total of factors */

printf("\n\n\n Now trying to recognize subject in top half of screen.\n");
printf("\n Presently trained with %d subjects.",(i/4));

for (w=0; w<6; w++) {
 for (id=1; id<(i/4)+1; id++) {
   m = id*4 - 3;
   c=0;
   for (x=0; x<5 ; x++) {
     for (y=0; y<5; y++) {
       gix = ( (double) (tlist[m].feature[w][x][y] +
                         tlist[m+1].feature[w][x][y] +
                         tlist[m+2].feature[w][x][y] +
                         tlist[m+3].feature[w][x][y] )/4.0);
       gux = (double) ilist[num].feature[w][x][y];

/*
                   V O T I N G   S C H E M E

      Voting can be done on four consecutive test feature sets
      (consecutive is ilist).  Substitute this assign statement
      for the variable gux.

      gux = ( (double) (ilist[num].feature[w][x][y] +
                        ilist[num+1].feature[w][x][y]
                        ilist[num+2].feature[w][x][y]
                        ilist[num+3].feature[w][x][y] )/4.0);

*/

       sigix = ( (double) (abs(gix-tlist[m].feature[w][x][y])*
               abs(gix-tlist[m].feature[w][x][y])+
```

```
                abs(gix-tlist[m+1].feature[w][x][y])*
                abs(gix-tlist[m+1].feature[w][x][y])+
                abs(gix-tlist[m+2].feature[w][x][y])*
                abs(gix-tlist[m+2].feature[w][x][y])+
                abs(gix-tlist[m+3].feature[w][x][y])*
                abs(gix-tlist[m+3].feature[w][x][y])))/4.0;
         sigix = sqrt(sigix);
         if (sigix < .5) sigix = .5;

         c += (gix-gux)*(gix-gux)/(4*sigix*sigix);

       }
     }

     v[id][w] = exp(-1.0*c/10000) * p[w];

  }
}

for (id=1; id<(i/4)+1; id++) {
   t[id] = 0.000000001;
   for (w=0; w<6; w++) {
     t[id] += v[id][w];
   }
   t[id] = t[id]/p[6];
                     /*max t[id] = 1.0 when distance from id to unknown*/
}                                               /* individual = 0.0 */
/*  printf("\nSorting Distances %2d",i); */

/* now have all distances ordered by id#, need to order id#s by dist */
for (m=1; m<101; m++) {
   list[m] = 0;
   list2[m] = 0.000000001;
}
for (m=1; m<((i/4)+1) && m<16; m++) {
   most = 0.000000001;
   for (j=1; j<(i/4)+1; j++) {
      if (t[j] > most) {
         most = t[j];
         n = j;
      }
   }
   list[m] = n;                                              /* id # */
   list2[m] = t[n];                                      /* distance */
   t[n] = 0.000000001;
}

/* now have ordered list of candidates, need to display them */
test = 0;
if (list2[1] > 0.001) {
   printf("\n\n                Candidate      Distance");
/*   printf("   Confidence");   */
   for (m=1; m<((i/4)+1) && m<16; m++) {
```

```
      if (list2[m] > 0.001) {
         if (m == 1) {
            printf("\n 1st Choice: ");
            t8[0] = '\0';
            strcat(t8,dbasedir);
            strcat(t8,tlist[list[1]*4 - 3].name);
            strcat(t8,".pic\0");
            readim(50,286,200,200,t8,"nocomm");
            text(50,266,0,1,200,tlist[list[1]*4 - 3].name);
            test = 1;
         }
         if (m == 2) {
            printf("\n 2nd Choice: ");
            t8[0] = '\0';
            strcat(t8,dbasedir);
            strcat(t8,tlist[list[2]*4 - 3].name);
            strcat(t8,".pic\0");
            readim(200,286,200,200,t8,"nocomm");
            text(200,266,0,1,200,tlist[list[2]*4 - 3].name);
            test = 2;
         }
         if (m == 3) {
            printf("\n 3rd Choice: ");
            t8[0] = '\0';
            strcat(t8,dbasedir);
            strcat(t8,tlist[list[3]*4 - 3].name);
            strcat(t8,".pic\0");
            readim(350,286,200,200,t8,"nocomm");
            text(350,266,0,1,200,tlist[list[3]*4 - 3].name);
            test = 3;
         }
         if (m == 4) printf("\n     Others: ");
         if (m > 4) printf("\n               ");
/*         confid = based on distance of this candidate and
                           distances to next candidates */
         printf("%lls     %f",tlist[list[m]*4 - 3].name,list2[m]);
 /*        printf("       %d",confid);*/
      }
      else m=200;
   }
}
if (test == 0) {
   printf("\n\n Could not find any close enough candidates.");
   printf("\n The computer has never seen this person before.");
}
if ( test < 3 && test != 0) {
  printf("\n\n Could not find any more close enough candidates so");
  if (test == 1) printf("\n only displayed 1 picture.");
  else printf("\n only displayed 2 pictures.");
}
prtc();
return;
}
```

```
/****************************************************************************/
int isolate(thresh,mode,size)    /* works on top half of screen only! */

    int thresh;                  /* threshold for detection of target */
    int mode;                                    /* 6 bit or 8 bit image */
    int size;  /* determines minimum size of target and affects speed. */

{                                  /* size is either 16 or 32 pixels. */
    int x,y,z;

sx = sy = fx = fy = -1;  /* Find top.    ****************************** */

for (y=size-1; y <= 255; y=y+size){  /* This subroutine finds location */
 for (x = 0; x < 511; x=x+size){     /* of a moving object. If there is*/
  z = brpixel(x,y);                  /* no moving object, or it is too */
  if (mode == 6) z = z & 63;         /* small then (0) is returned. If */
  if (z >= thresh) {                 /* an object is found then sx,sy, */
   sy = y-(size-1);                  /* fx,fy are set and (1) is re-   */
   x = 512;                          /* turned. This is done so that   */
   y = 512;                          /* all future work done on a scene*/
}}}
                                     /* is done on a greatly reduced   */
if (sy == -1) return(0);             /* area of the scene and hence is */
for (y=256-size; y>(sy+size-1); y=y-size){
                                     /* done faster. Thresh is set to*/
 for (x = 0; x <= 511; x=x+size){   /* high enough value to eliminate */
  z = brpixel(x,y);                 /* video noise but low enough to  */
  if (mode == 6) z = z & 63;         /* find small brightness differen-*/

  if (z >= thresh){                  /* ces that may occur between a   */
   fy = y + size-1; /* Find bottom.   * moving object and its bkgnd.   */
   x = 512;                         /*********************************/
   y = -1;
}}}

if (fy < (sy + size)) return(0);
for (x=size-1; x <= 511; x=x+size){                   /* find left side */
 for (y = 0; y < 255; y=y+size){
  z = brpixel(x,y);
  if (mode == 6) z = z & 63;
  if (z >= thresh){
   sx = x - (size-1);
   x = y = 512;
}}}

if (sx == -1) return(0);
for (x = 512-size; x > (sx + size-1); x = x - size){
 for (y = 0; y < 255; y = y + size){                 /* find right side */
  z = brpixel(x,y);
  if (mode == 6) z = z & 63;
  if (z >= thresh){
   fx = x + size-1;
```

```
   x = -1;
   y = 512;
}}}

if (fx < (sx + size)) return(0);
return(1);
}

/*************************************************************************/
#define A0 (short int)a0(i)   /* These are the transformations used in */
#define a0(i) (i & 0x003f)    /* the feedback lut for the real time    */
#define A1 (short int)a1(i)   /* subtraction demo. This software was   */
                              /* created by using the toolbox */
#define a1(i) ((i & 0x0fc0) >> 6)     /* program (see FG-100 user's */
#define D0(i) { data &= 0xffc0; data |= (i & 0x003f); }
                                                /* manual chapt 7)  */
#define D1(i) { data &= 0xf03f; data |= ((i << 6) & 0x0fc0); } /*********/
#define INPUT 0x6000
#define abs(i) (((i) < 0) ? (-(i)) : (i))

xform1(addr, initial)
 unsigned addr,initial;
{
register unsigned short i = addr;
register short int data = initial;
D1(A1);
D0(abs(A1 - A0));
return((unsigned)data);
}

xform2(addr, initial)
 unsigned addr,initial;
{
register unsigned short i = addr;
register short int data = initial;
D1(A0);
D0(abs(A1 - A0));
return((unsigned)data);
}
```

```
/*****************************************************************************/
afrm()      /* A completely Autonomous Face Recognition Machine (AFRM) */
{

 int cam;
 char stop,answer[1];
 register unsigned j;

stop = 'n';
cam = 9;
while (cam != 0 && cam != 1 && cam != 2) {
    printf("\n Select camera port (0,1 or 2) >");
    scanf("%d",&cam);
    }

while (stop == 'n') {
 cls();
 printf(" please wait...");
 rtsubtract(0);
 setcamera(cam);
 setlut(0,0);
 setinmux(6);
 for (j=0; j<0x1000; j++) write_lut(INPUT,j,xform2(j,read_lut(INPUT,j)));
 cls();
 printf(" looking for target.");
 snap(1);
 snap(1);
 while((isolate(8,6,32)) != 1) snap(1);
 printf("\n found target, acquiring 8 bit image.");
 initialize();
 setcamera(cam);
 waitvb();
 snap(1);
 nf = sx = sy = 0;
 fx = 511;
 fy = 255;          /* presently isolate() only looks for target in top */
 if (facemap() == 1) {          /* half so look for faces in top half */
  printf("\n found ");
  printf("%d",nf);
  if (nf == 1) printf(" face.");
  else printf(" faces.");
  facerec(2);
 }
 printf("\n Do you wish to stop? (Y/N) >");
 scanf("%s",answer);
 if (answer[0] == 'Y' || answer[0] == 'y') stop = 'y';
}

return;
}
```

```
/****************************************************************************/
del()
{
printf("\n\n              Deleting files with reserved names.");
system("delete bnorm.img;*");
                          /* these names are reserved for facefinder */
system("delete orig.img;*");
return;
}

/****************************************************************************/
prtc()                                                          /* 455 */
{
printf("\n\n Press RETURN to continue. >");
getchar();
getchar();
return;
}

/****************************************************************************/
cls()
{
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
return;
}
```

```
/*********************************************************************/
int readfftfile(name,str)
   char name[];
   struct list str[];
{
   char c,instring[10];
   FILE *fp;
   int h,i,j,k,l,x;
   double y;

fp = fopen(name,"r");
fscanf(fp,"%d",&h);
for (i=1;i<h+1;i++) {
   fscanf(fp,"%10s",&instring);
   for (j=0;j<10;j++){
         str[i].name[j]=instring[j];
   }
   str[i].name[j] = '\0';
   fscanf(fp,"%d",&x);
   str[i].num = x;

   for (j=0;j<6;j++)
      for (k=0;k<5;k++){
         for (l=0;l<5;l++) {
            fscanf(fp,"%f",&y);
            str[i].feature[j][k][l] = y;
         }
      }
}
fclose(fp);
return(h);
}
```

```
/*************************************************************************/
writeffcfile(name,str,i)     /* used to write updated DAT files to disk */
 char name[]   /* when user is done modifying the database and selects */
 struct list str[];       /* menu option = 0 (Return to main menu).      */
 int i;                 /************************************************/

{
 FILE *fp,*fopen();
 int j,k,l,m;

delete(name);
fp = fopen(name,"w");
fprintf(fp,"%d\n",i);
for (j=1; j<(i+1); j++) {
 fprintf(fp, "%-10s%5d \n", str[j].name, str[j].num);
 for (k=0;k<6;k++)
   for (l=0;l<5;l++) {
     for (m=0;m<5;m++)
       fprintf(fp, "%3.3f ", str[j].feature[k][l][m]);
     fprintf(fp,"\n");
   }
}
fprintf(fp, "*");
fclose(fp);
return;
}
```

```
/*************************************************************************/
display(str,k,m)      /* m = 6 or 8 depending on # columns desired      */
 struct list str[];   /* m = 6 for ilist displays, 8 for tlist displays */
 int k,m;             /* l = present column being printed on screen     */
{                     /* j counts by 1 or 4 depending on value of m     */
 int j,l,n;           /* this is due to format of tlist file; there are */
l = 0;                /* sets of 4 lines all with the same name and the */
if (m == 6) {         /* name only needs to be printed once.            */
 printf("\n\n The AFRM has the following image feature sets:\n");
 printf(" -----------------------------------\n");
}
else {
 printf(" The AFRM is trained on the following subjects:\n");
 printf(" -------------------------------------------\n");
}
if (m==8) n = 4;
else n = 1;
for (j=1; j<(k+1); j=j+n) {
 l = l + 1;
 if (l == m) {
  l = 1;
  printf("\n");
 }
 if (m == 6) printf("%11s%s%d",str[j].name, ",", str[j].num);
 else printf("%11s",str[j].name);
}
return;
}
```

```
/**********************************************************************/
copyfile(src,dest)

    char src[],dest[];
{

    char t9[80];

t9[0] = '\0';
strcat(t9,"copy \0");
strcat(t9,src);
strcat(t9," \0");
strcat(t9,dest);
printf("\n %s",t9);
system(t9);
return;
}

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

Appendix E

FACEFT

```
/* ************************************************************************
*
*                          This is FACEFT.C
*
*          This is a subset of FACEDFT that is concerned with
*          the Fourier Transform used for the feature set.
*
*          These routines use a standard two- dimensional
*          Fourier Transform, but calculate only the dc component
*          and the first two harmonics.
*
*************************************************************************/
```

```
/***************************************************************************/
static double crayr[200][200],sinaray[5][3],cosaray[5][3],rinp[65];
static int ix,iy;

/***************************************************************************/
gestalt(m)                                    /* Values range from 0 to 128  */
int m;                                                   /* m = face number */

{
     int x,y,i,j;
line(256,0,256,512,0);
line(0,256,512,256,0);
line(384,0,384,512,0);
line(128,256,128,512,0);

/* left half: whole head */
carea(sx,sy,face[m].dx/2,face[m].dy,270,sy,face[m].dx/2,face[m].dy);
/* right half: whole head */
carea(sx+face[m].dx/2,sy,face[m].dx/2,face[m].dy,
                                        400,sy,face[m].dx/2,face[m].dy);
/* top half: top to tnose */
carea(sx,sy,face[m].dx,face[m].tnose,15,sy+256,face[m].dx,face[m].tnose)
;
/* internal features */
carea(sx+face[m].leye,sy+face[m].teye,face[m].reye-face[m].leye,
                                          face[m].cmouth-face[m].teye,
      140+face[m].leye,sy+256+face[m].teye,face[m].reye-face[m].leye,
                                          face[m].cmouth-face[m].teye);

/* left internal features */
carea(sx+face[m].leye,sy+face[m].teye,face[m].center-face[m].leye,
                                          face[m].cmouth-face[m].teye,
      270+face[m].leye,sy+256+face[m].teye,face[m].center-face[m].leye,
                                          face[m].cmouth-face[m].teye);
/* bottom half: tnose to chin */
carea(sx,sy+face[m].tnose,face[m].dx,face[m].dy-face[m].tnose,
      400,sy+256+face[m].tnose,face[m].dx,face[m].dy-face[m].tnose);
line(sx,sy,sx+face[m].dx,sy,0);                                    /*top*/
line(sx+face[m].dx,sy,sx+face[m].dx,sy+face[m].dy,0);            /*right*/
line(sx+face[m].dx,sy+face[m].dy,sx,sy+face[m].dy,0);           /*bottom*/
line(sx,sy+face[m].dy,sx,sy,0);                                   /*left*/
line(sx,sy+face[m].teye,sx+face[m].dx,sy+face[m].teye,0);         /*teye*/
line(sx,sy+face[m].cmouth,sx+face[m].dx,sy+face[m].cmouth,0); /*cmouth*/
line(sx,sy+face[m].tnose,sx+face[m].dx,sy+face[m].tnose,0);     /*tnose*/
line(sx+face[m].leye,sy,sx+face[m].leye,sy+face[m].dy,0);         /*leye*/
line(sx+face[m].center,sy,sx+face[m].center,sy+face[m].dy,0); /*center*/
line(sx+face[m].reye,sy,sx+face[m].reye,sy+face[m].dy,0);         /*reye*/
ix = face[m].dx/2;
iy = face[m].dy/2;

printf("\n calculating fft for window 1.");
clear_cray();                                 /* left half: whole head */
for (y=sy; y<sy+face[m].dy; y+=2)
```

```
      for (x=270; x<270+face[m].dx/2; x+=2) {
         crayr[(x-269)/2][(y-29)/2] = (double) brpixel(x,y)/255;
      }
   ft2(crayr,1+(x-269)/2,1+(y-29)/2);
   save_fft(0);

   printf("\n calculating fft for window 2.");
   clear_cray();                                  /* right half: whole head */
   for (y=sy; y<sy+face[m].dy; y+=2)
      for (x=400; x<400+face[m].dx/2; x+=2) {
         crayr[(x-399)/2][(y-29)/2] = (double) brpixel(x,y)/255;
      }
   ft2(crayr,1+(x-399)/2,1+(y-29)/2);
   save_fft(1);

   printf("\n calculating fft for window 3.");
   clear_cray();                                  /* top half: top to tnose */
   for (y=sy+256; y<sy+256+face[m].tnose; y+=2)
      for (x=15; x<15+face[m].dx; x+=2) {
         crayr[(x-14)/2][(y-285)/2] = (double) brpixel(x,y)/255;
      }
   ft2(crayr,1+(x-14)/2,1+(y-285)/2);
   save_fft(2);

   printf("\n calculating fft for window 4.");
   clear_cray();                                        /* internal features */
   for (y=sy+256; y<sy+256+face[m].cmouth; y+=2)
      for (x=140; x<140+face[m].reye; x+=2) {
         crayr[(x-139)/2][(y-285)/2] = (double) brpixel(x,y)/255;
      }
   ft2(crayr,1+(x-139)/2,1+(y-285)/2);
   save_fft(3);

   printf("\n calculating fft for window 5.");
   clear_cray();                                  /* left internal features */
   for (y=sy+256; y<sy+256+face[m].cmouth; y+=2)
      for (x=270; x<270+face[m].dx/2; x+=2) {
         crayr[(x-269)/2][(y-285)/2] = (double) brpixel(x,y)/255;
      }
   ft2(crayr,1+(x-269)/2,1+(y-285)/2);
   save_fft(4);

   printf("\n calculating fft for window 6.");
   clear_cray();                              /* bottom half: tnose to chin */
   for (y=sy+256; y<sy+256+face[m].dy; y+=2)
      for (x=400; x<400+face[m].dx; x+=2) {
         crayr[(x-399)/2][(y-285)/2] = (double) brpixel(x,y)/255;
      }
   ft2(crayr,1+(x-399)/2,1+(y-285)/2);
   save_fft(5);

   return;
   }
```

```
/***********************************************************************/
 save_fft(index)
 int index;

{
 int x,y;

 ilist[0].feature[index][0][0] = sinaray[0][0];
 ilist[0].feature[index][0][1] = sinaray[0][1];
 ilist[0].feature[index][0][2] = sinaray[0][2];
 ilist[0].feature[index][0][3] = cosaray[0][1];
 ilist[0].feature[index][0][4] = cosaray[0][2];
 ilist[0].feature[index][1][0] = sinaray[1][0];
 ilist[0].feature[index][1][1] = sinaray[1][1];
 ilist[0].feature[index][1][2] = sinaray[1][2];
 ilist[0].feature[index][1][3] = cosaray[1][1];
 ilist[0].feature[index][1][4] = cosaray[1][2];
 ilist[0].feature[index][2][0] = sinaray[2][0];
 ilist[0].feature[index][2][1] = sinaray[2][1];
 ilist[0].feature[index][2][2] = cosaray[2][0];
 ilist[0].feature[index][2][3] = cosaray[2][1];
 ilist[0].feature[index][2][4] = cosaray[2][2];
 ilist[0].feature[index][3][0] = sinaray[3][0];
 ilist[0].feature[index][3][1] = sinaray[3][1];
 ilist[0].feature[index][3][2] = cosaray[3][0];
 ilist[0].feature[index][3][3] = cosaray[3][1];
 ilist[0].feature[index][3][4] = cosaray[3][2];
 ilist[0].feature[index][4][0] = sinaray[4][0];
 ilist[0].feature[index][4][1] = sinaray[4][1];
 ilist[0].feature[index][4][2] = cosaray[4][0];
 ilist[0].feature[index][4][3] = cosaray[4][1];
 ilist[0].feature[index][4][4] = cosaray[4][2];

return;
}

/***********************************************************************/
clear_cray()
{
     int x,y;

for (y=0; y<200; y++)
  for (x=0; x<200; x++)
    crayr[x][y] = 0.0;
return;
}
```

```
/***************************************************************************/
ft2(raray,ftx,fty)

double raray[200][200];

{
int i,j,k,l,x,y;

for (k=0; k<5; k++) {
   for (l=0; l<3; l++) {
      cosaray[k][l] = 0.0;
      sinaray[k][l] = 0.0;
      for (i=0; i<ftx; i++) {
         for (j=0; j<fty; j++) {
cosaray[k][l] += raray[i][j] * cos(-i*(k-2)*2*pi/ftx-j*(l-2)*2*pi/fty);
sinaray[k][l] += raray[i][j] * sin(-i*(k-2)*2*pi/ftx-j*(l-2)*2*pi/fty);

         }
      }
   }
}

return;
}

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

Bibliography

Bush, Larry F. The Design of an Optimum Alphanumeric Symbol Set for Cockpit Displays, MS Thesis AFIT/GE/ENG/77-11. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1977 (AD-A053447).

Darnell, Peter A. and Phillip E. Margolis. Software Engineering in C. New York: Springer-Verlag, 1988.

Fretheim, Erik J. Computer Program. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, Summer 1988.

Fretheim, Erik J. Personal Interview. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, May 1989.

Gonzalez, Rafael C. and Paul Wintz. Digital Signal Processing. Reading MA: Addison-Wesley Publishing Company, 1977.

Kabrisky, Matthew, Director Signal Processing Laboratory. Personal Interview. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, July 1989.

Kernighan, Brian W. and P. J. Plauger. The Elements of Programming Style (Second Edition). New York: McGraw Hill Book Company, 1978.

Lambert, Larry C. Evaluation and Enhancement of the AFIT Autonomous Face Recognition Machine, MS Thesis AFIT/GE/ENG/87D-35. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A188819).

O'Hair, Mark A. Whole Word Recognition Based on Low Freq Fourier Complex and Amplitude Spectra, MS Thesis AFIT/GEO/ENG/84D-4. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.

O'Hair, Mark A. Computer Program. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, August 1989.

Pressman, Roger S. Software Engineering: A Practitioner's Approach (Second Edition). New York: McGraw-Hill Book Company, 1982.

Routh, Richard L. Cortical Thought Theory: A Working Model of the Human Gestalt Mechanism, Ph.D. Dissertation AFIT/DS/EE/85-1. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, July 1985 (AD-A163215).

Russel, Robert L. Jr. Performance of a Working Face Recognition Machine Using Cortical Thought Theory, MS Thesis AFIT/GE/ENG/85D-37. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985 (AD-A167781).

Sander, David D. Enhanced Autonomous Face Recognition Machine, MS Thesis AFIT/GCS/ENG/88D-19. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988 (AD-B128376L).

Smith, Edward J. Development of an Autonomous Face Recognition Machine, MS Thesis AFIT/GE/ENG/86D-36. School of ENgineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986 (AD-A178852).

Wahl, Friedrich M. Digital Image Signal Processing. Boston: Artech House, 1987.

Vita

Captain Barbara C. Robb

She attended the Illinois Institute of Technology and received the degree of Bachelor of Science in Computer Science in May 1975. She worked as a civilian in the computer field for seven years. She was commissioned in June 1983 after attending Officer Training School. She then attended the Air Force Institute of Technology as part of the conversion program, receiving the degree of Bachelor of Science in Electrical Engineering in March 1985. Next, she served as a Project Officer for the Secretary of the Air Force Special Projects at Los Angeles AFB California until entering the School of Engineering, Air Force Institute of Technology in May 1988.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION: School of Engineering

6b. OFFICE SYMBOL *(If applicable)*: AFIT/EN

6c. ADDRESS *(City, State, and ZIP Code)*: Air Force Institute of Technology, Wright-Patterson AFB OH 45433-6583

7a. NAME OF MONITORING ORGANIZATION

7b. ADDRESS *(City, State, and ZIP Code)*

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL *(If applicable)*

8c. ADDRESS *(City, State, and ZIP Code)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*: AUTONOMOUS FACE RECOGNITION MACHINE USING A FOURIER FEATURE SET

12. PERSONAL AUTHOR(S): Barbara C. Robb, B.S., Capt, USAF

13a. TYPE OF REPORT: MS Thesis

13b. TIME COVERED: FROM ______ TO ______

14. DATE OF REPORT *(Year, Month, Day)*: 1989 December

15. PAGE COUNT: 148

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
12	09,01	
06	04	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*: Pattern Recognition; Discrete Fourier Transforms; Face; Face Recognition

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Thesis Advisor: Matthew Kabrisky, PhD
Professor of Electrical Engineering

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL: Matthew Kabrisky, PhD

22b. TELEPHONE *(Include Area Code)*: (513) 255-9267

22c. OFFICE SYMBOL: AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

UNCLASSIFIED

This thesis demonstrates ourier coefficients as a reliable feature set for face recognition, using the Autonomous Face Recognition Machine developed at AFIT over the past several years (Routh, 1985; Russel, 1985; Smith, 1986; Lambert, 1987; Sander, 1988).

The Fourier transform portion of the system was examined and improved. The code was made more efficient. Two Fourier

Fourier transform) were tested and compared. A voting scheme was incorporated for examining multiple looks at test faces. To further demonstrate performance, the number of faces in the data base was doubled.

Recognition scores of up to 87% were achieved, compared to 63% for Sander's process with Fourier coefficients as a feature set and 67% for Lambert's process with a center-of-mass feature set. (Sander, 1988:32).

This thesis includes complete system documentation, to assist those doing further research in this area.

UNCLASSIFIED