7-1-1994

Human face profile recognition

Vincent Wong

Human Face Profile Recognition

by

Vincent Wong

A Thesis Submitted
in
Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by: ________________________________
Graduate Advisor - Tony H. Chang, Professor

____
Ronald G. Matteson, Professor

____
Department Head - Roy S. Czernikowski

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

July, 1994

Thesis Release Permission Form

Rochester Institute of Technology
College of Engineering

Title: Human Face Profile Recognition

I, Vincent Wong, hereby grant permission to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part.

Signature: ______________________

Date: 7/20/94

This document was produced using Lotus AmiPro 3.0 and Microsoft Windows Paintbrush. All programs for the human face profile recognition system were developed using Microsoft QuickC version 2.5 and they were run on an AT&T 386-class PC compatible machine with a math co-processor. All profile images were captured using ITEX PCVISIONplus frame grabber board and Panasonic WV-BL204 CCD camera.

The following names used here are registered trademarks of the respective companies:

AmiPro	Lotus Development Corporation,
Paintbrush	Microsoft Corporation,
QuickC	Microsoft Corporation,
PCVISIONplus	Imaging Technology Incorporated.

Abstract

The purpose of this thesis is to implement an automatic person identification system based on face profiles. Each person's face profile can be quite unique within a small sample population and therefore it can be used as the basis of an automatic person identification system. To quantify human face profiles for use in the recognition system, Fourier descriptors are used to describe the open curve extracted from a face profile. Fourier descriptors in the low-frequency range are shown to be useful for human face profile recognition. By using 16 Fourier coefficients, a correct recognition rate of 92% for 60 subjects was achieved.

Contents

List of Figures

List of Tables

Human Face Profile Recognition

Chapter 1. Introduction

Using computers to identify human faces has always been an attractive field to scientists and engineers. There are other person identification systems based on fingerprinting, iris-scanning, or retina-scanning available nowadays; however, none of them is more natural than identifying a person with his/her own face. This type of person identification system using the human face as the basis can be used for identification of criminals. It can also be used for authentication in secure systems. For example, it can add additional security on top of the required personal identity number (PIN) at the automatic teller machines (ATM), or it can automate the personnel check-in/check-out at the entrances of office buildings. Probably one of the most unique features to this kind of person identification systems is that the person being examined may not even be aware of the examination taking place. The person's face can be zoomed in with a video camera hidden in a place where nobody can see. This type of non-contact and non-interactive automatic person identification system is most valuable in areas of surveillance and security.

There are two types of human face images that can be used in a person identification system.(12) Of course, the most natural way to identify a person is from the frontal image (i.e. the camera is focused on the front of the face.) Many studies have been devoted to this area.(13,14,15,16) However, the frontal image of a person can be very difficult to analyze because of its complexity and variability. In addition, the amount of information that one can extract from a frontal image can be overwhelming, which means it can take quite an amount of computing time to perform a single identification.

On the other hand, it is possible to identify a person from the face profile (i.e. the camera points towards one side of the face.) Work in this area dates back to the last century when Francis Galton proposed algorithmic techniques for quantifying normalized profile traces with characteristic lengths and angles.[9,10] Each person has a unique face profile. We often can recognize a person from the face profile with very casual inspection. We do that in many social occasions. The profile image of a person's face is easier to analyze than the frontal image since the only information that we need to process is the shape of the profile.

Researches on face profile recognition using fiducial points are commonly found.[6, 7, 8, 11] Systems with this approach use fiducial marks such as chin, nose, forehead, bridge, mouth and so forth. The distances between the fiducial points, angles between them, and areas of some triangles formed by the fudicial points are used as the features.

Aibara *et al.* used a different approach to identify face profiles.[4, 5] Fourier descriptors in the low-frequency range were shown to be useful for human face profile recognition. A correct recognition rate of 93.1% for 130 subjects had been reported.

In this thesis, a person identification system based on the face profile is implemented. Fourier descriptors are used to characterize face profile curves. The input to the system is assumed to be an image containing predominantly a human face profile. Otherwise, detection of face profiles becomes an additional problem. It is important to distinguish between face profile detection and face profile identification. In face profile detection, an algorithm is devised to search for the presence of one or more face profiles in an image. If a face is present, its size and location in the image must also be determined. This problem is fairly complex and computational intensive. On the other hand, face profile identification assumes that the profile curve of a face has been perceived, and the next step is to associate a name to the face.

This thesis concerns face profile identification only. Therefore there are some strict rules on how a person may pose in front of the camera. First of all, it is required that the size of the face profile be at least half the vertical size of the image. Secondly, the face should be upright although a little tilt is tolerable. Moreover, there should be no occlusion (i.e. glasses are not permitted.) Lastly, mouths should also be closed naturally to ensure consistency in the face profiles. To simplify the problem, it is also required that the left face profile be captured.

Chapter 2. System Overview

Like most of the recognition systems, the human face profile recognition system has two modes of operations:

1) collect and store data in a database, and

2) match input data against stored data in a database.

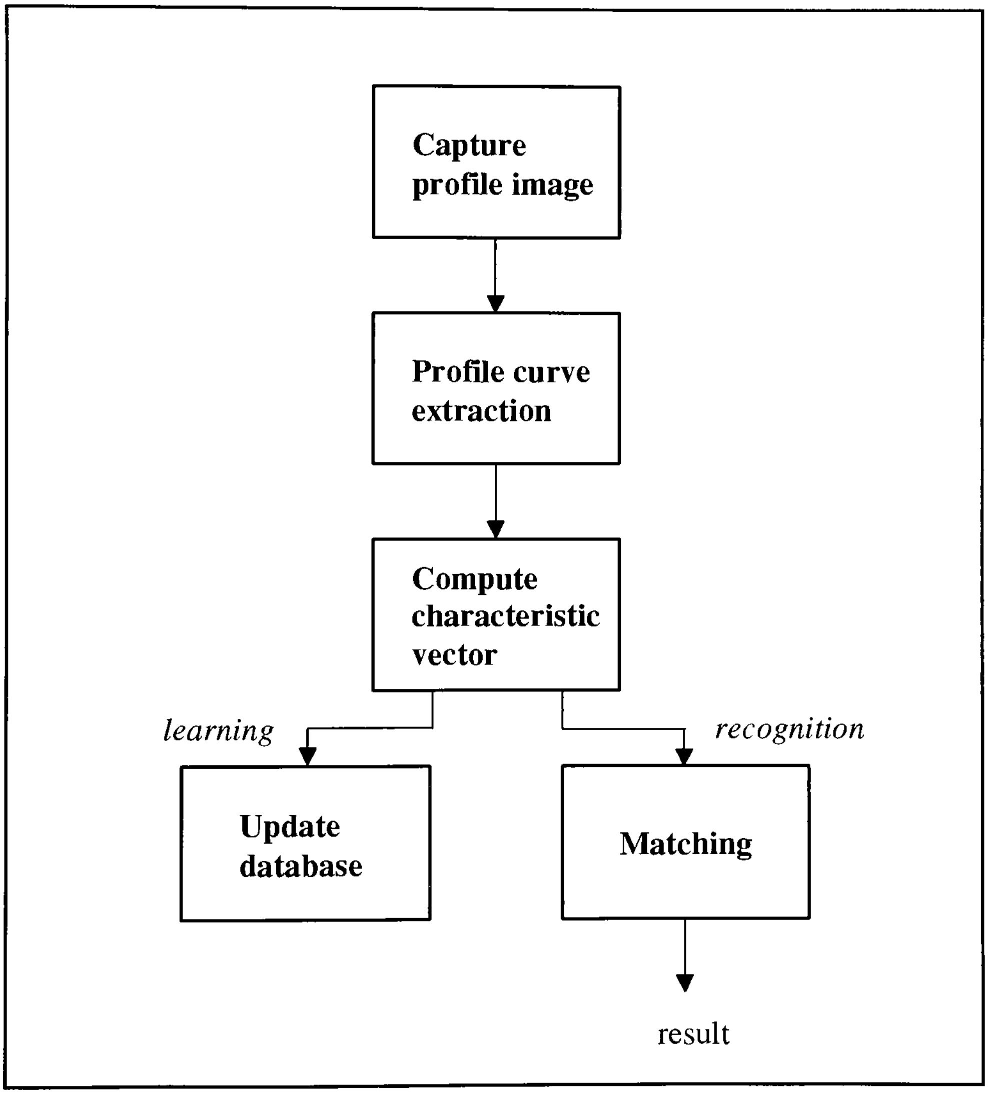


Figure 2.1 *Block diagram of the human face profile recognition system.*

Figure 2.1 shows the block diagram of the human face recognition system. The input data to the database are the feature vectors extracted out of the profile images captured with a CCD camera. Input data are stored in a database for future reference in the *training* mode, or matched against the stored data in a database in the *recognition* mode. The human face profile recognition is divided into five major functional units, each of which will be briefly discussed in this section.

A. Capture profile image

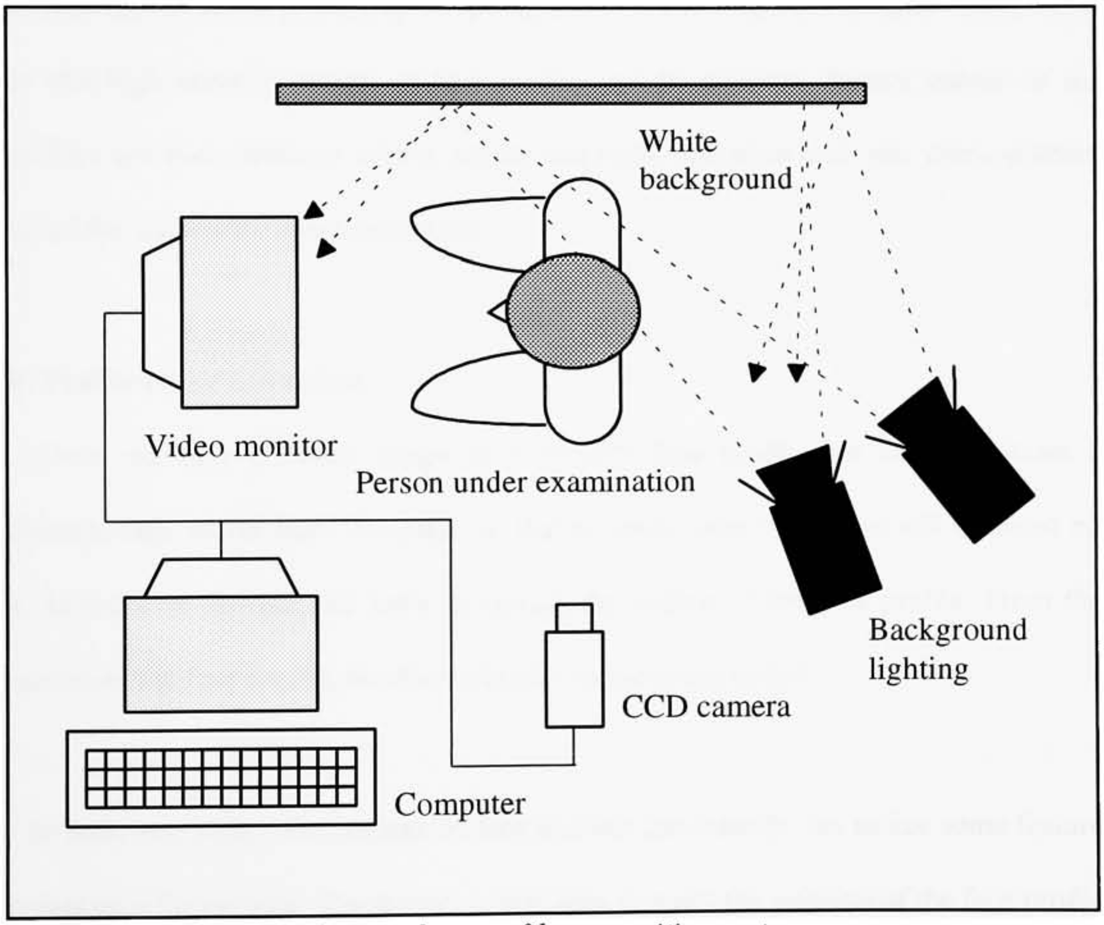


Figure 2.2 *Setup of the human face profile recognition system.*

The domain of our recognition system is the human face profile. Therefore, the input to the recognition system are profile images of faces. The input device for capturing images is a CCD camera. In **Figure 2.2**, the setup of the experimental human face profile recognition system is shown. The person under examination sits beside a white background. In front of the person is a video monitor which is connected to a CCD camera and a frame-grabber board resides in a computer. The CCD camera captures the profile image of the person's face. The live video image of the person's face profile provides a feedback to the person so that adjustment of the posture can be made accordingly. A set of photographic spotlights is used to brighten the white background so that high contrast images of face profiles can be obtained. Binary images of the profiles are then obtained with a simple threshold operation with the frame-grabber board for the profile curve extraction.

B. Profile curve extraction

Once we have a binary image of a person's face profile, we have to obtain a characteristic vector from the image so that all subsequent operations will be based on it. In order to do that, we have to extract the outline of the face profile. From the outline of the face profile, the characteristic vector is computed.

In order to extract the outlines of face profiles consistently, we utilize some feature points on a face profile. The feature points help to mark the position of the face profile and give clues to the size of the face profile in an image. There are six feature points that we are interested in. Among them, two are derived from the others. These feature points are:

1) the tip of nose,

2) the top of nose,

3) the bottom of nose,

4) the chin position,

5) the upper terminating point (derived), and

6) the lower terminating point (derived).

The feature point extraction relies on the general shape of human face profiles. For instance, there must be a nose protrusion and a chin protrusion in a face profile, regardless of who the person is. Once we obtain the upper and the lower terminating points, the outline of the face profile is simply the boundary curve running from one terminating point to the other. The details of the curve extraction process will be discussed in **Chapter 3. Face Profile Curve Extraction**.

C. Compute characteristic vector

Once the processor has determined the face profile curve of a person, it is not far from being able to compare it with others. The comparison should concern only the shape difference between two profile curves. Therefore we need to get a characteristic vector from the profile curve which can represent the curve independent of its size, position, and orientation in the image plane. We have chosen Fourier descriptors to represent face profile curves since Fourier descriptors are invariant of size, position, and orientation of any closed boundary. In our case, a face profile curve can be viewed as a line object with a closed boundary. The mathematics of Fourier descriptors and how we use them to describe open curves such as face profile curves will be discussed in detail in **Chapter 4. Fourier Descriptors**.

D. Update database

The characteristic vector, in which we use Fourier descriptors to represent face profile curves, is invariant of size, position, and orientation. Therefore it is useful for comparison purpose. As we have seen in **Figure 2.1**, there are two operations that we can perform after we obtain a characteristic vector from a face profile. We can either store the characteristic vector in a database for future reference or we can match it with the stored vectors in a database.

A database is simply a collection of characteristic vectors which represent face profiles of people. The vectors in a database are identified by names. To learn a new face profile is to merely put a new entry in the database with the person's name associated with it. Usually more than one characteristic vector are obtained from the same person in various poses. So, we take the average and store the averaged values in the database.

D. Matching

The human face profile recognition system identifies an unknown person by comparing the characteristic vector obtained from the person's face profile to the set of known vectors in a database. Distance measurement is made between the vector of the unknown person and each vector in the database. The unknown face profile is identified to be the person whose characteristic vector yields the shortest distance in the distance measurement.

The performance of the human face profile recognition system greatly depends on how the well the Fourier descriptors can resolve the differences in the face profiles of different persons. A complete analysis of the system and the test results will be presented in **Chapter 5. Analysis of the System Performance**.

Chapter 3. Face Profile Curve Extraction

Before we can perform any kind of comparisons on human faces, we have to extract the face profile curve from the image. To automate the extraction process, we have to successfully locate the upper and the lower terminating positions on the face profile. The face profile curve is then defined as the curve running from the upper to the lower terminating positions along the face-to-background boundary.

To help locate the upper and the lower terminating positions, a few feature points on the face profile must be identified. These feature points mainly identify the positions of the nose and the chin.

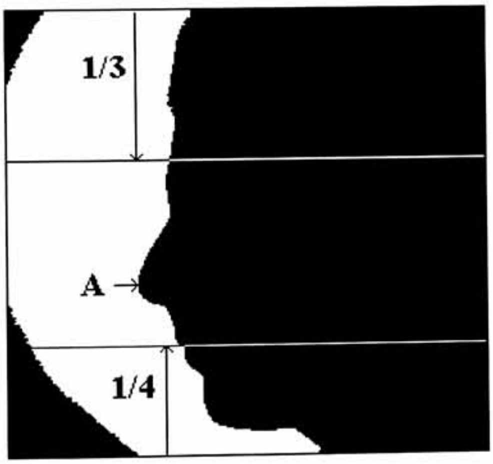


Figure 3.1 *Tip of nose.*

The first feature point that we identify is the *tip of nose* (**Point A** in **Figure 3.1**). Since the tip of nose is a protrusion that is most highlighted in a human face profile, we choose that as the point of reference in our face profile curve extraction. To locate the tip of nose, we search for the extreme left point in the face silhouette in a window that is 1/3 down from the top of the image and 1/4 up from the bottom of the image. The upper and the lower positions of the window are determined by examining a large number of images posed by a large number of people.

In order to successfully extract the face profiles from the face images of various sizes, some kind of distance measurement on the face profile is necessary. The nose distance, defined as the distance between the top and the bottom of nose, is used as a reference. The upper and the lower terminating positions will be determined based on this reference distance.

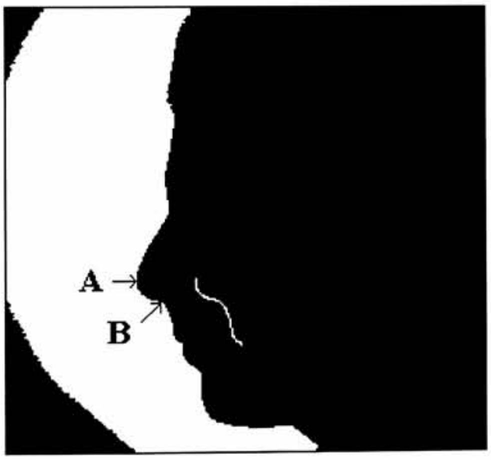


Figure 3.2 *Bottom of nose.*

The second feature point that we identify is the *bottom of nose* (**Point B** in **Figure 3.2**). First, a small portion of the face profile curve is extracted out from the tip of nose and downward. The extracted curve is encoded in chain code. The details of the chain code and the curve extraction will be discussed in **Chapter 3.1 Chain Code and Curve Extraction**. For now, we have a small portion of the face profile curve. The bottom of nose is defined as the position of the first clockwise turn on the curve, starting from the tip of nose. The details of the *turning point detection algorithm* will be discussed in **Chapter 3.2 Turning Point Detection Algorithm**.

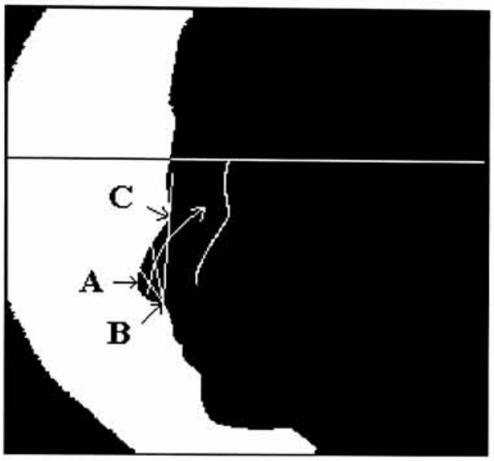


Figure 3.3 *Top of nose.*

The third feature point that we identify is the *top of nose* (**Point C** in **Figure 3.3**). Unlike the bottom of nose, it is not easy to use the turning point detection algorithm to locate the top of nose because of the large variation that is found in the shapes of noses and eyes of people. In the turning point detection algorithm, only the chain code of the curve is examined. The actual shape of the face profile is not taken into consideration. Therefore, a different approach is used to locate the top of nose.

Since the position of the top of nose is a point on the face profile curve above the tip of nose, we extract a portion of the face profile curve as shown in **Figure 3.3**. Then, the line of sight from the bottom of nose to each point on the curve is examined, starting from the tip of nose. If the immediate extension of the line of sight from the bottom of nose does not belong to the background, the top of nose is said to be found. The immediate extension is a line of a few pixel units in length, an extension that is sufficient to distinguish the nose-to-background boundary and the peak at the top of nose seen from the inside of the face silhouette.

Now that we have clearly defined the position of the nose with three feature points, we can compute the distance between the top and the bottom of nose. We will use this distance, called a "reference distance", *d*, in our chin position definition.

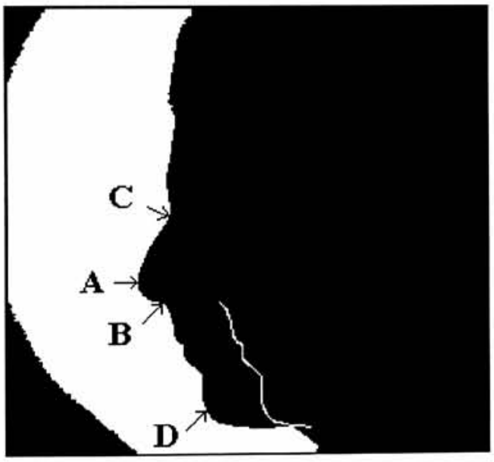


Figure 3.4 *Chin position.*

The next feature point that we identify is the *chin position* (**Point D** in **Figure 3.4**). For the chin position, the turning point algorithm is used along with a distance constraint. First, the curve of the face profile below the bottom of nose is extracted. Then the chin position is defined as the first counterclockwise turn on the curve that is at least a reference distance away from the bottom of nose. This distance constraint provides the discrimination against the feature positions at the lips that may be mistaken by the turning point detection algorithm.

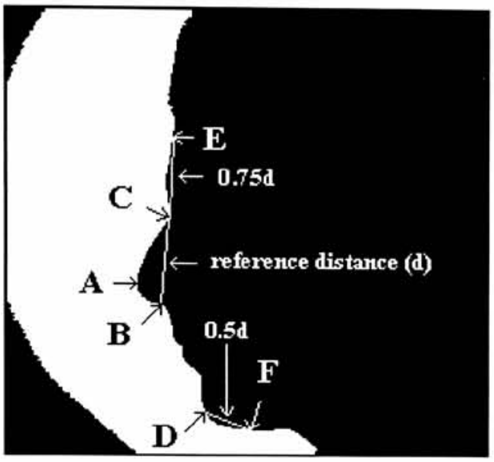


Figure 3.5 *Terminating positions.*

The last two points that we have to locate are the *upper and the lower terminating positions* (**Points E and F** in **Figure 3.5**). These two points are defined by the distances from the other feature points on the face profile. These distances are defined in terms of the reference distance d that we computed from positions of the top and the bottom of nose. The upper terminating position is defined as the point that is $0.75d$ above the top of nose and the lower terminating position is defined as the point that is $0.5d$ below the chin position. The ratios defined here also come from the observation of a large number of human face samples.

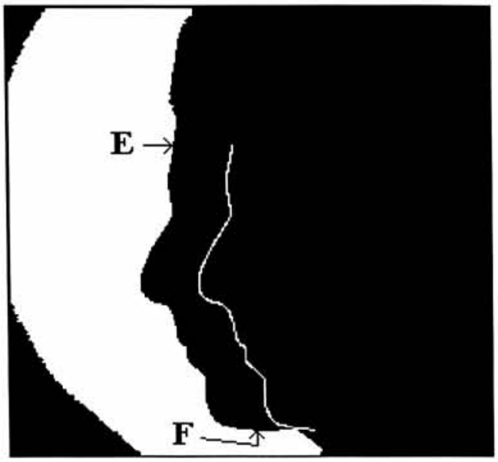


Figure 3.6 *Complete face profile curve.*

Finally, the complete face profile curve is extracted from the upper to the lower terminating positions as shown in **Figure 3.6**.

Chapter 3.1 Chain Code and Curve Extraction

To get the face profile curve is to get the contour of the face silhouette. The curve is encoded in chain code. Eight-connectedness is used to represent the positions of the neighboring pixels. The 8 directions are numbered in a clockwise modulo-8 fashion as shown in **Figure 3.1.1**. To minimize the noise from the rough edges that may be found in the face silhouette during the contour-tracing, a large contour-tracing mask is used. The contour-tracing mask is analogous to a ball rolling on a hill surface. The larger the ball is, the less bumpy is the resulting locus of the center of the ball. This technique is similar to the erosion operation in morphological image processing except that a constraint is applied to the movement of the mask in our case. Using a mask in contour-tracing is essentially applying a low-pass filter to the curve. All the high-frequency components such as the noisy edges are eliminated in the resulting curve. However, if we use too large a mask, the critical curvature information may also be removed. With a working image size of 512×480 pixels, the size of the contour-tracing mask is chosen to be 7 pixels in diameter. The digitized version of the contour-tracing mask is shown in **Figure 3.1.2**. During the contour-tracing, the contour-tracing mask is moved along the inside edge of the face silhouette. The locus of the center of the contour-tracing mask is then encoded in chain code.

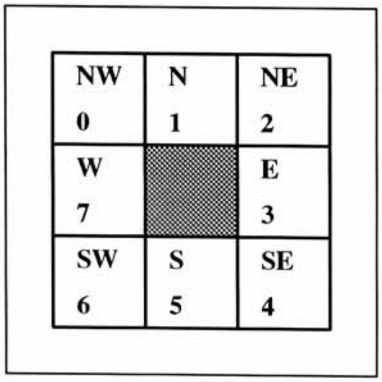


Figure 3.1.1 *Eight-connectedness.*

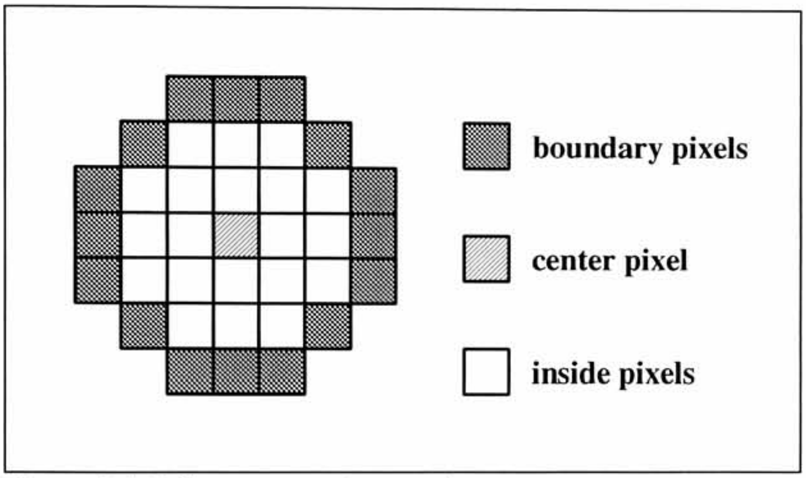


Figure 3.1.2 *Contour-tracing mask.*

One constraint we impose on the movement of the curve is that the maximum directional change from pixel to pixel is limited to 1 unit. In other words, the angular change from one pixel to the next pixel is limited to 45 degrees. The reason for imposing such constraint on contour-tracing is to eliminate the ambiguity found in the curve length calculation.

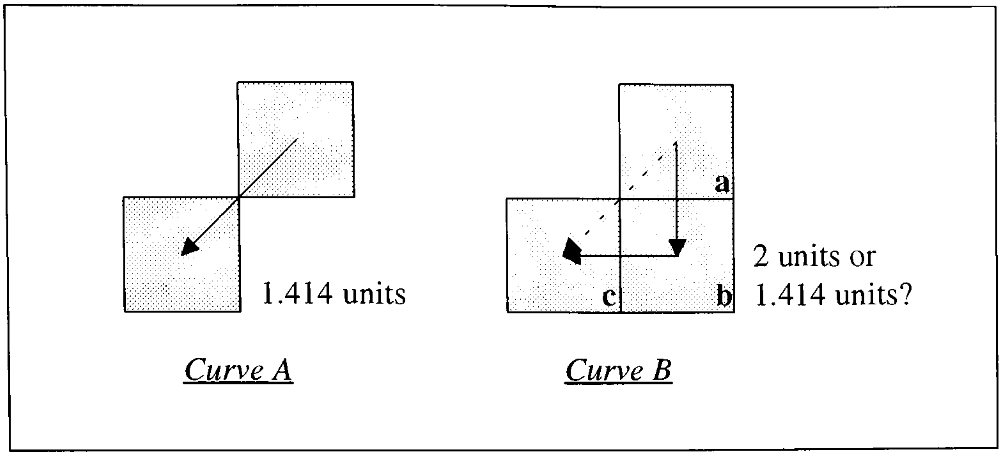


Figure 3.1.3 *Ambiguity in curve length calculation.*

Consider the two situations in **Figure 3.1.3**. Let the displacement from any pixel to its horizontal or vertical neighbors be 1 unit and 1.414 units (root 2) for its diagonal neighbors. The curve length for *curve A* and *curve B* are 1.414 units and 2 units respectively. *Curve B* has a directional change of 2 units from *pixel b* to *pixel c*. As we can see, *pixel b* and *pixel c* are both neighbors of *pixel a*. Therefore *curve B* could have been 1.414 units in length if it went from *pixel a* to *pixel c* directly. Since the curve length directly affects how we sample the curve, we simply impose a constraint to limit the movement from pixel to pixel. The constraint allows the contour-tracing mask to be moved in 3 directions only. They are:

1) 45 degrees clockwise,

2) no change in direction since the last movement, and

3) 45 degrees counterclockwise.

The decision on the movement from pixel to pixel is made by examining the 3 possible moves in the order listed above. The order of the examination is important because it ensures that the contour-tracing mask always leans towards the edge of the face silhouette. The path of a move is considered clear when none of the edge pixels of the contour-tracing mask overlaps with the background. When the path of a move being examined is clear, the position of the current pixel is advanced with that move. If none of the 3 possible moves is legal, the position of the current pixel is advanced with move #3 (45 degree away from the edge.) This condition may occur with a right-angled turn in a face silhouette. An example is shown in **Figure 3.1.4**. The right-angled turn in a face silhouette does not cause any problem in contour-tracing. However, it should be handled properly. It should also be mentioned that it is a rare case since a human face profile is usually smooth. A sharp angle on a face profile like this is very unlikely to exist.

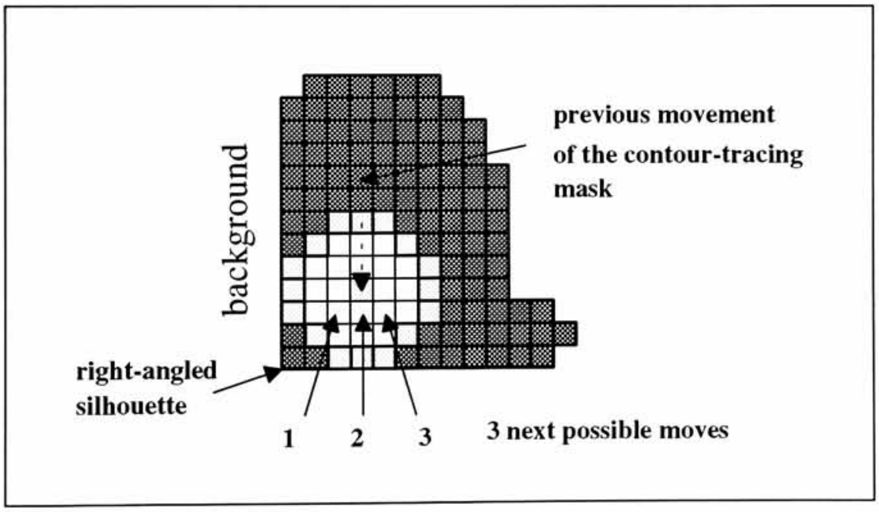


Figure 3.1.4 *Right-angled silhouette. All 3 possible moves are illegal.*

Chapter 3.2 Turning Point Detection Algorithm

The bottom of nose and the chin position are located with the *turning point detection algorithm*. The algorithm is to apply a series of filters to the chain code of a curve and find the points on the curve which represent the points of inflections. The filtering process is shown in the block diagram of **Figure 3.2.1**.

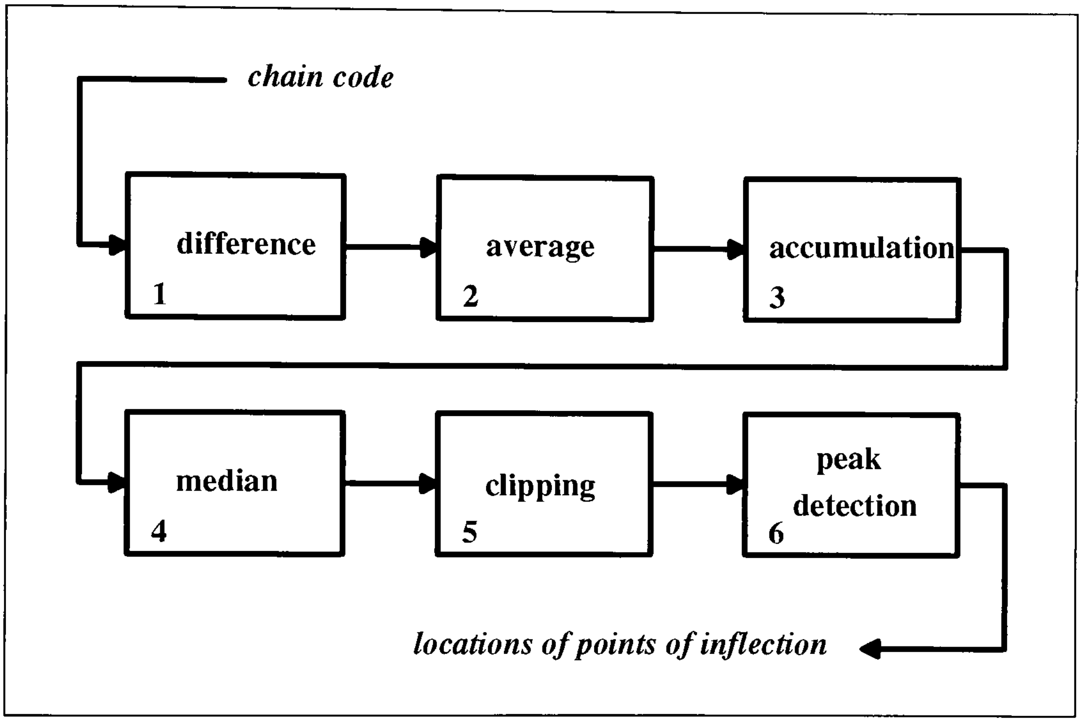


Figure 3.2.1 *Turning point detection algorithm block diagram.*

There are 6 steps in the process. The chain code is encoded with 8 allowable directions as discussed in **Chapter 3.1 Chain Code and Curve Extraction**. The directions are numbered in a modulo-8 fashion so that the incremental change in direction from pixel to pixel can be computed. To simplify the chain encoding and eliminate the ambiguity found in the curve length calculation, the maximum directional displacement cannot be greater than 1 unit between neighboring pixels.

In **Figure 3.2.1**, the first filter converts the chain code to its incremental values. The output is then passed to an averaging filter. The averaging filter performs a convolution on the input data with a kernel of all 1's. Only the portion of the curve with a large net directional change would yield a large magnitude. The choice of the kernel size is dependent upon the feature size on the curve. A kernel size of 7 is sufficient for our case. The output of the averaging filter is passed to an accumulation filter which is essentially the same filter as the averaging filter but with a larger kernel size. The accumulation filter generates large magnitudes at positions in which the curve has apparent directional changes. The next two filters, the median and the clipping filters, are used to process the data before the peak detection.

In the accumulation filter, a small kernel is used when the target is a sharp turning position. In the case of detecting the bottom of nose, a kernel size of 11 is used. When the filter is used to detect a slow turn on the curve, a larger kernel is used. In the case of detecting the chin position, a kernel size of 31 is used. The reason for using a larger kernel for a slower turn is obvious. If a small kernel was used on a slow turn, each segment of the curve as seen by the kernel would become nearly a straight line and it would be impossible to detect the turn.

In **Figure 3.2.2**, we are trying to detect and locate the bottom of nose. The intermediate output for each stage of the filtering process is shown in **Figure 3.2.3**.

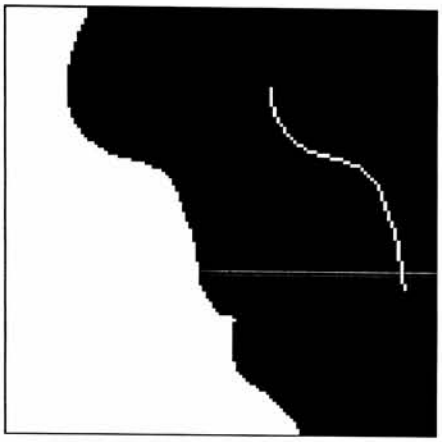
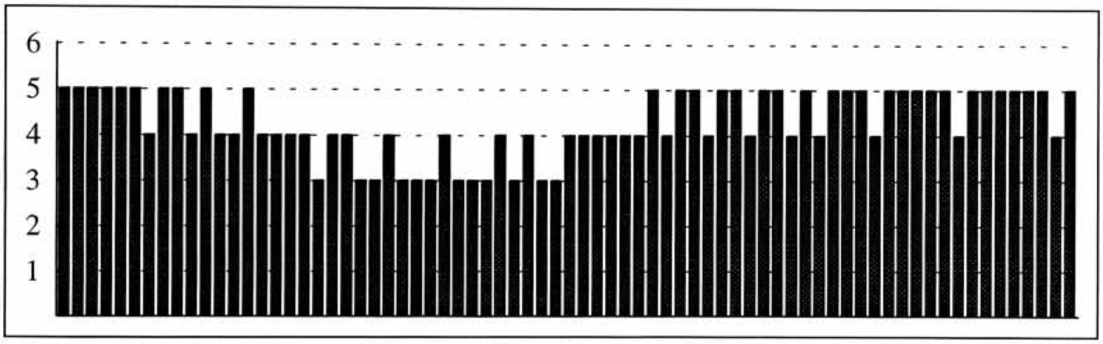
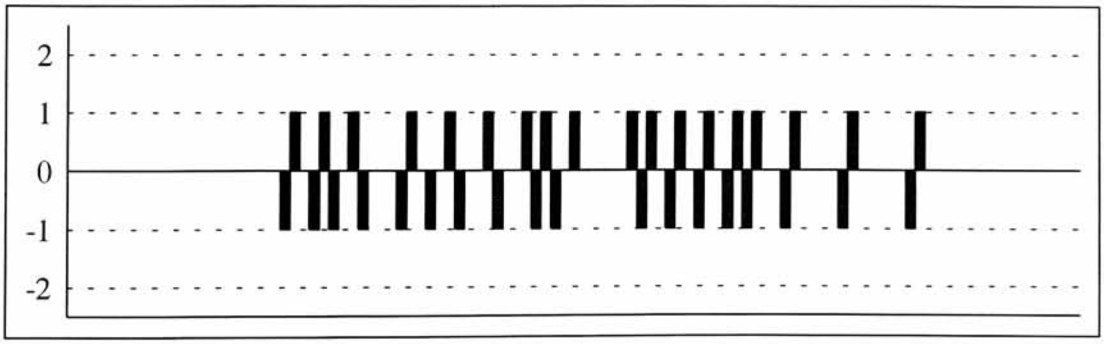


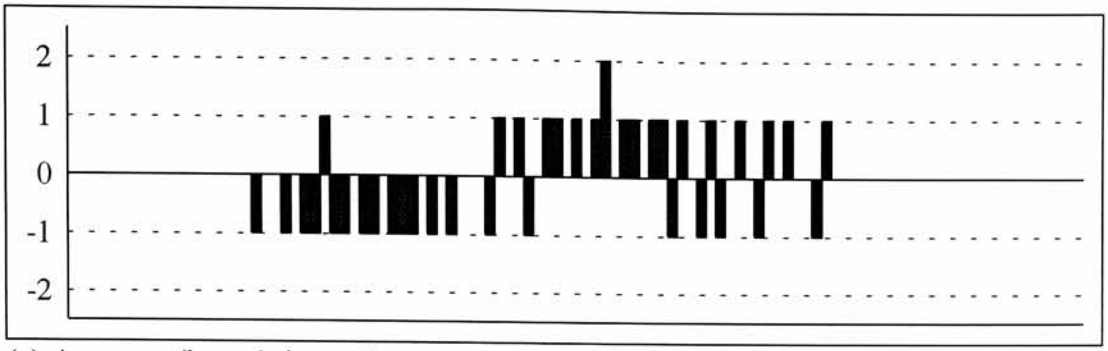
Figure 3.2.2 *Bottom of nose.*



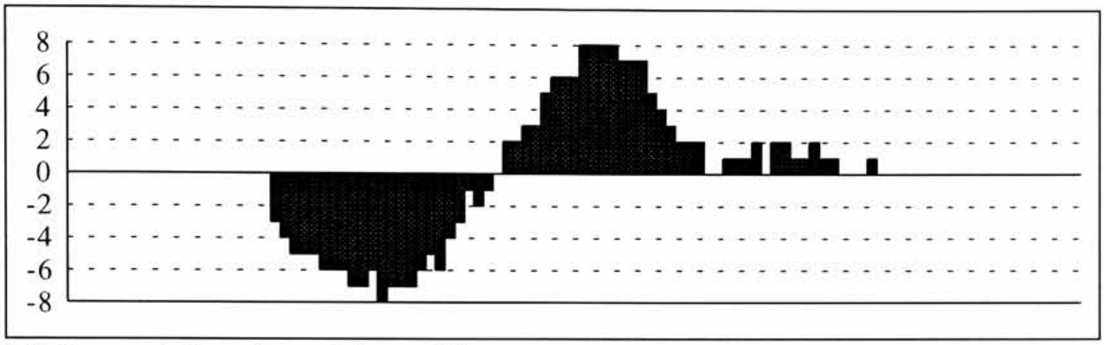
(a) Input chain code.



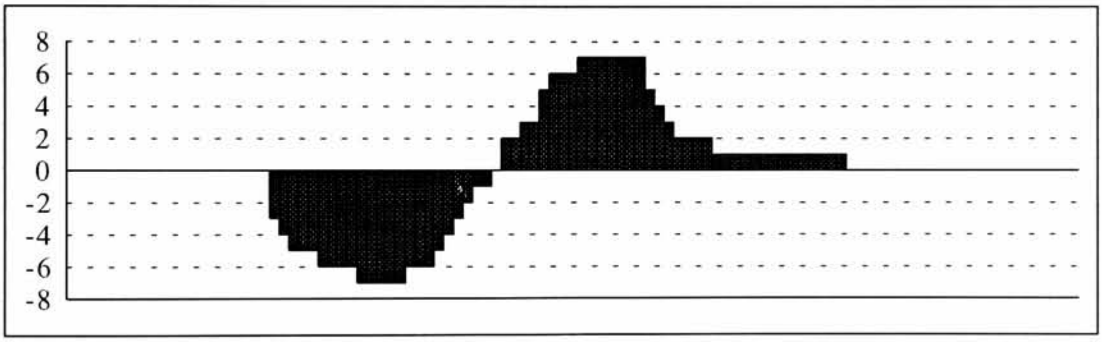
(b) Incremental values.



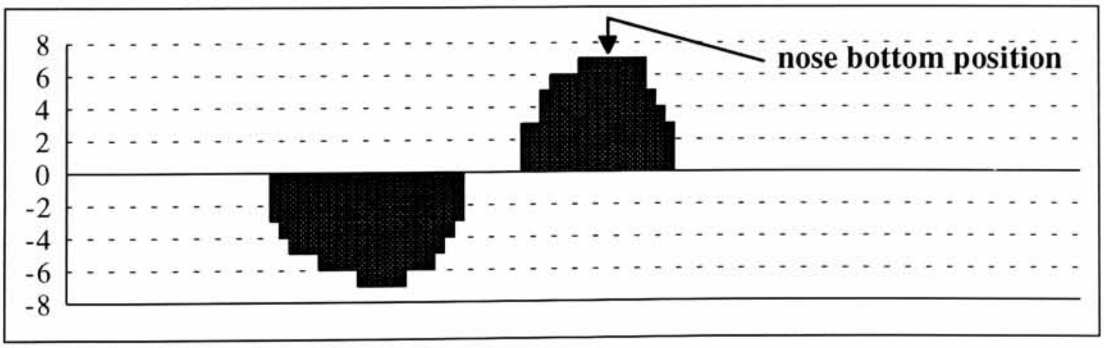
(c) Average. (kernel size = 7)



(d) Accumulation. (kernel size = 11)



(e) Median. (filter size = 11)



(f) Clipping. (threshold value = 3)

Figure 3.2.3 *Using the turning point detection algorithm to locate the bottom of nose.*

Chapter 4. Fourier Descriptors

After we obtain the outline of the face profile of a person in the form of chain code, we compute a characteristic vector of the profile curve using Fourier descriptors.

There are many ways to define Fourier descriptors that represent closed curve contour functions. Two of them were reviewed by Persoon and Fu.[1] The Fourier descriptors presented by Zahn and Roskies are based on the function of arc length by the accumulated change in direction of the curve since the starting point.[2] Granlund defined the Fourier descriptors in the complex space that is immediately related to the Cartesian image plane.[3] The Fourier descriptors used in the human face profile recognition system is based on it. The following represents the mathematical considerations of the technique used by Granlund.

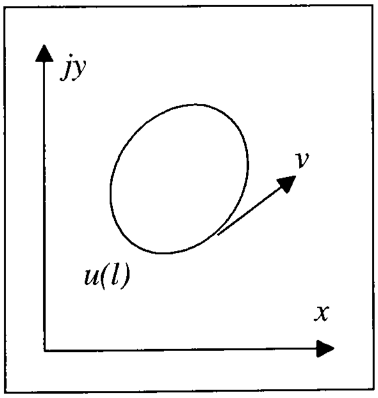


Figure 4.1 *A contour function in a complex space.*

A contour function, a closed-curve C, is included in a complex space as shown in **Figure 4.1**. A complex-valued function u is generated by moving a point around the contour. Assume that the point is moving at a constant speed along C. Let the parameter of length covered by the movement of the point at every time t be l. Then the complex function is represented by

$$u(l) = x(l) + jy(l).$$

Let the total arc length of the closed-curve C be L and let the complex function u be periodic with period L. Now the complex function u can be expressed as a Fourier series. The Fourier coefficients become

$$a_n = \frac{1}{L}\int_0^L u(l)e^{-jn2\pi l/L}dl$$

and

$$u(l) = \sum_{n=-\infty}^{\infty} a_n e^{jn2\pi l/L}.$$

For simplicity, we let $L = 2\pi$. Then the formulas become

$$a_n = \frac{1}{2\pi}\int_0^{2\pi} u(l)e^{-jnl}dl$$

and

$$u(l) = \sum_{n=-\infty}^{\infty} a_n e^{jnl}.$$

The Fourier coefficients here are not unique for a specific contour. They are dependent upon the starting position. We are also interested in the effect on the Fourier coefficients when the contour undergoes translation, rotation, and dilation.

A. Starting position

There is a set of Fourier coefficients for each starting position of the contour-tracking. That means that there is a set of Fourier coefficients for each δ of the function

$$u = u(l+\delta).$$

We now assume that there exists a function

$$u(l) = u^{(0)}(l),$$

and let $a^{(0)}$ be the set of Fourier coefficients of this specific contour function. All the other functions are given by

$$u(l) = u^{(0)}(l+\delta).$$

The resulting Fourier coefficients become

$$\begin{aligned} a_n &= \frac{1}{2\pi}\int_0^{2\pi} u^{(0)}(l+\delta)e^{-jnl}dl \\ &= \frac{1}{2\pi}\int_0^{2\pi} u^{(0)}(l)e^{-jn(l-\delta)}dl \\ &= \frac{1}{2\pi}\int_0^{2\pi} u^{(0)}(l)e^{-jnl}e^{jn\delta}dl \\ &= e^{jn\delta}a_n^{(0)}. \end{aligned}$$

Therefore, the Fourier coefficients differ from that of the specific contour function by a factor of $e^{jn\delta}$.

B. Translation

When the specific contour function is translated with a complex vector Z, it becomes

$$u(l) = u^{(0)}(l) + Z.$$

The Fourier coefficients then become

$$\begin{aligned} a_n &= \frac{1}{2\pi}\int_0^{2\pi}[u^{(0)}(l) + Z]e^{-jnl}dl \\ &= \frac{1}{2\pi}\int_0^{2\pi} u^{(0)}(l)e^{-jnl}dl + \frac{1}{2\pi}\int_0^{2\pi} Ze^{-jnl}dl \\ &= a_n^{(0)} \qquad\qquad \text{for } n \neq 0, \text{ or} \\ &= a_n^{(0)} + Z \qquad \text{for } n = 0. \end{aligned}$$

Therefore, all coefficients except a_0 are invariant of translation.

C. Rotation

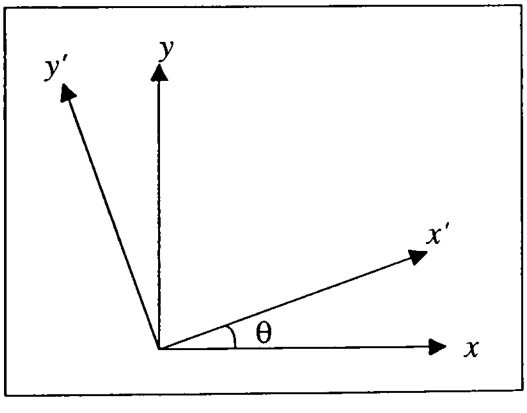


Figure 4.2 *A rotated coordinate system.*

In **Figure 4.2**, the original coordinate system is rotated counterclockwise by θ. From the elementary trigonometry, the new and old axes are related by the equations

$$x' = x\cos\theta + y\sin\theta$$
$$y' = -x\sin\theta + y\cos\theta$$

or in matrix format,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

The 2×2 matrix that transforms the original coordinate system to the new system can be written as a complex vector $e^{-j\theta}$ if the image plane is viewed as a complex space. Therefore, when the specific contour function is rotated counterclockwise by θ in the complex image plane, the contour function become

$$u(l) = e^{-j\theta} u^{(0)}(l),$$

and hence the Fourier coefficients become

$$a_n = e^{-j\theta} a_n^{(0)}.$$

D. Dilation (Scaling)

The size of the specific contour can be scaled with a factor R. Similarly, it can be shown that the Fourier descriptors are simply multiplied with R.

$$a_n = R a_n^{(0)}.$$

General form of Fourier coefficients

As a result, the general form of the Fourier coefficients generated by translation, rotation, dilation, and changes in the starting position can be expressed as

$$a_n = e^{-jn\delta} \cdot e^{-j\theta} \cdot R \cdot a_n^{(0)} + Z\delta(n),$$

where $\delta(n)$ is a delta function.

Table 4.1 summarizes the Fourier coefficients for a contour function that undergoes all possible geometric transformations and changes in starting position.

Transformation	*Contour function*	*Fourier coefficients*
Translation	$u(l) = u^{(0)}(l) + Z$	$a_n = a_n^{(0)} + Z\delta(n)$
Rotation	$u(l) = e^{-j\theta}u^{(0)}(l)$	$a_n = e^{-j\theta}a_n^{(0)}$
Dilation	$u(l) = Ru^{(0)}(l)$	$a_n = Ra_n^{(0)}$
Starting position	$u(l) = u^{(0)}(l+\delta)$	$a_n = e^{jn\delta}a_n^{(0)}$

Table 4.1 *Some basic properties of Fourier coefficients.*

In order to make the Fourier coefficients useful in shape discrimination, the coefficients of a contour function must be independent of translation, rotation, dilation, and the starting position. Consider the following set of coefficients, derived from the Fourier coefficients in their general form a_n:

$$\begin{aligned} b_n &= (a_{1+n}a_{1-n})/a_1^2 \\ &= \left[a_{1+n}^{(0)}e^{j(1+n)\delta}Re^{-j\theta} \cdot a_{1-n}^{(0)}e^{j(1-n)\delta}Re^{-j\theta}\right]/\left[a_1^{(0)}e^{j\delta}Re^{-j\theta}\right]^2 \\ &= \left[a_{1+n}^{(0)}a_{1-n}^{(0)}\right]/\left[a_1^{(0)}\right]^2 \qquad \text{for } n \neq 1. \end{aligned}$$

Both sets of coefficients are complex numbers. The difference in these two sets of coefficients is that the new coefficients of the contour function are invariant of the starting position, rotation, and dilation. Because the new coefficients do not contain a_0, they are also independent of translation.

Fourier descriptors for open curves

In order to use the Fourier descriptors discussed above on open curves, we trace the line pattern once and then retrace it so that a closed boundary is obtained. An example is shown in **Figure 4.3**.

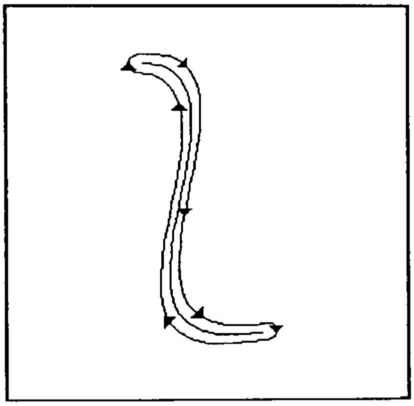


Figure 4.3 *Trace the line pattern once and retrace it.*

Since the curve is an open curve, the positions of the two terminating points are known. We can define the Fourier descriptors based on the magnitudes of the general Fourier coefficients. We have

$$
\begin{aligned}
c_n &= |a_{n+1}|/|a_1| \\
&= \left[\left|a_{n+1}^{(0)}\right| \cdot \left|e^{j(n+1)\delta}\right| \cdot R \cdot \left|e^{-j\theta}\right|\right] / \left[\left|a_1^{(0)}\right| \cdot \left|e^{j\delta}\right| \cdot R \cdot \left|e^{-j\theta}\right|\right] \\
&= \left[\left|a_{n+1}^{(0)}\right| \cdot \left|e^{j(n+1)\delta}\right|\right] / \left[\left|a_1^{(0)}\right| \cdot \left|e^{j\delta}\right|\right] \qquad \text{for } n \geq 0.
\end{aligned}
$$

The resulting coefficients are independent of translation, rotation, and scaling. Although the coefficients are still sensitive to the starting position, we can let the starting position be one of the terminating points of the open curve.

Implementation

The mathematics of the Fourier descriptors we discussed so far assumed the contour function was continuous in space. The Fourier transform we applied was the Fourier Series. In reality, the curve is represented by a discrete number of points in the complex image plane. Since we trace the curve and retrace it back to the starting point during the curve sampling, the sample values represent a periodic discrete-time signal. Therefore, we can express the signal in a discrete-time Fourier series. Obviously, the sample points of a boundary curve should be evenly spaced in curve length. The details of the curve sampling will be discussed in **Chapter 4.1 Curve Sampling**.

Let $u(k)$ be a complex-valued function that represents the coordinates of the points on the boundary curve of a line object sampled at a fixed arc length. Let N be the number of data points in $u(k)$. The discrete-time Fourier series becomes

$$a(k) = \frac{1}{N}\sum_{n=0}^{N-1} u(n)e^{-j2\pi kn/N} \qquad \text{for } 0 \le k < N,$$

and

$$u(k) = \sum_{n=0}^{N-1} a(n)e^{2\pi kn/N} \qquad \text{for } 0 \le k < N.$$

The normalized Fourier coefficients are then given by

$$c_k = |a_{k+1}|/|a_1|.$$

Due to the symmetry in the Fourier coefficients, that is

$$|a_k| = |a_{N-k}|,$$

there are $N/2$ normalized Fourier coefficients for an N-point transformation. So, the Fourier descriptors for an open curve are given by c_0 to $c_{N/2-1}$.

Another way of looking at Fourier descriptors for open curves

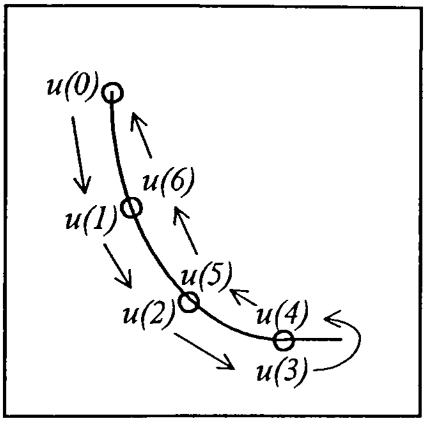


Figure 4.4 *An example of a way of open curve sampling.*

Consider the open curve as shown in **Figure 4.4**. Let $u(k)$ be a complex-valued function that represents the coordinates of the points on the boundary curve of a line object sampled at a fixed arc length. Let N be the number of samples in $u(k)$ and let N be an odd number as well. Moreover, let the starting point be one of the open curve terminals. Then, we have

$$u(k) = u(N-k) \qquad \text{for } 0 < \mathrm{k} < \mathrm{N}.$$

The discrete-time Fourier series becomes

$$a(k) = \frac{1}{N}\sum_{n=0}^{N-1} u(n)e^{-j2\pi kn/N}$$

$$= [u(0) + u(1)e^{-j2\pi k/N} + u(2)e^{-j2\pi 2k/N} + \ldots + u(\tfrac{N-1}{2})e^{-j2\pi(\frac{N-1}{2})k/N}$$
$$+u(N-1)e^{-j2\pi(N-1)k/N} + u(N-2)e^{-j2\pi(N-2)k/N} + \ldots + u(\tfrac{N-1}{2}+1)e^{-j2\pi(\frac{N-1}{2}+1)k/N}]/N$$

$$= [u(0) + u(1)e^{-j2\pi k/N} + u(2)e^{-j2\pi 2k/N} + \ldots + u(\tfrac{N-1}{2})e^{-j2\pi(\frac{N-1}{2})k/N}$$
$$+u(1)e^{+j2\pi k/N} + u(2)e^{+j2\pi 2k/N} + \ldots + u(\tfrac{N-1}{2})e^{+j2\pi(\frac{N-1}{2})k/N}]/N$$

$$= [u(0) + 2u(1)\cos(2\pi k/N) + 2u(2)\cos(2\pi 2k/N) + \ldots$$
$$+2u(\tfrac{N-1}{2})\cos(2\pi(\tfrac{N-1}{2})k/N)]/N$$

$$= \left\{2\sum_{n=0}^{(N-1)/2}[u(n)\mathrm{Real}(e^{-j2\pi kn/N})] - u(0)\right\}/N.$$

The formula implies that the discrete-time Fourier series of the closed boundary can be computed without explicitly closing the open curve. If we separate the real part and the imaginary part of the complex-valued function, *u(k)*, the Fourier transformation changes from a discrete-time Fourier series to a discrete Fourier transform (DFT). The Fourier coefficients become

$$\mathrm{Real}[a(k)] = \left\{2\,\mathrm{Real}\left\{\sum_{n=0}^{(N-1)/2}\mathrm{Real}[u(n)]e^{-j2\pi kn/N}\right\} - \mathrm{Real}[u(0)]\right\}/N,$$

$$\mathrm{Imag}[a(k)] = \left\{2\,\mathrm{Real}\left\{\sum_{n=0}^{(N-1)/2}\mathrm{Imag}[u(n)]e^{-j2\pi kn/N}\right\} - \mathrm{Imag}[u(0)]\right\}/N.$$

The equations show that the Fourier coefficients can be computed with the sample points of an open curve. In order to use discrete Fourier transform in our computation, the input signal to the transformation must be in real numbers. Therefore, it is necessary to separate the complex-valued function, *u(k)*, into its *x* and *y* components.

We have seen two ways of computing Fourier descriptors for open curves. In the human face profile recognition system, we choose the first method in our implementation. By explicitly closing an open curve, the Fourier transformation is straight forward since it is rather natural to represent the sample points in complex values.

Chapter 4.1 Curve Sampling

After the face profile curve extraction, the face profile curve is represented by a chain code. The characteristic vector of the face profile curve is then obtained using Fourier descriptors. The input signal to the discrete-time Fourier series transformation is a complex-valued function. It represents the periodic sequence of the locations of the sample points of a face profile curve. The open curve is viewed as a line object which has a boundary curve that can be traced and retraced back to the starting position. Therefore, the boundary curve is a function of (x,y) coordinates and its independent variable is the arc length since the starting position. To obtain a uniform sampling of the boundary curve, we have to sample the curve at a fixed arc length interval. The arc length interval is equal to the perimeter of the boundary curve divided by the number of sample points. Obviously, the perimeter of the boundary curve is 2 times the arc length of the open curve. Since the open curve that we obtained is represented by a sequence of pixels, the arc length of the open curve is approximated by summing the incremental distances from one pixel to the next pixel, from one end to the other end of the open curve. The incremental distance of the horizontal moves and the vertical moves are counted as 1 pixel unit while the diagonal moves are counted as 1.414 pixel units (root 2). To retain the fidelity of the original curve, all sample points are linearly interpolated from the original pixel coordinates of the open curve.

The positions of the sample points closest to the terminals of the open curve can affect the overall layout of the samples. Consider the three different sets of samples of the same curve in **Figure 4.1.1**. In all three cases, the sampling arc length interval is l. In cases **(a)** and **(b)**, the sample points of the retrace overlap with those of the first trace. In case **(c)**, the starting point is neither at the terminal of the open curve nor half way of the sampling interval from the terminal of the curve. As a result, the sampling points in the first trace do not overlap with those of the retrace.

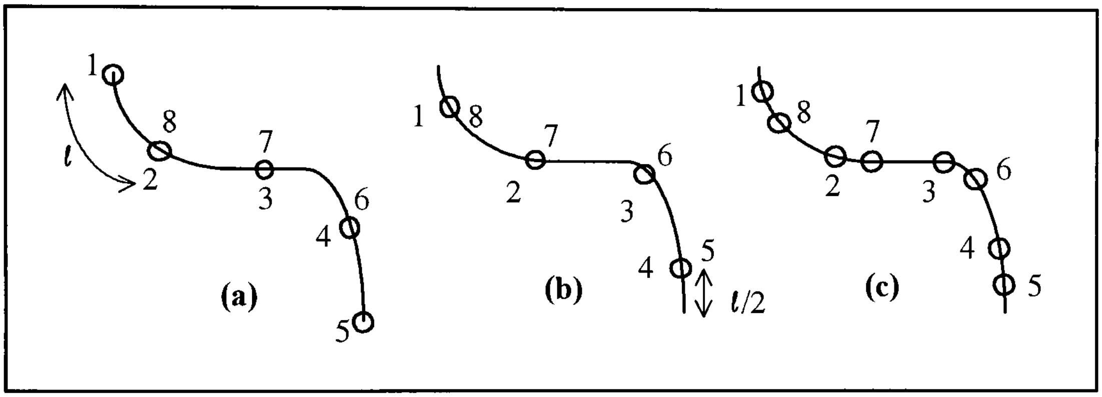


Figure 4.1.1 *Different ways of sampling the boundary curve of a line object.*

Obviously, it is easier to sample the boundary curve with cases **(a)** and **(b)** as shown in **Figure 4.1.1**. Sample points of the retrace of the curve can be duplicated from those obtained in the first trace.

In the human face profile recognition system, the open curves of face profiles are sampled as in case **(a)** of **Figure 4.1.1**. The total number of points that we obtain from the boundary curve is a power of 2. In that case, we can use the fast Fourier transform to calculate our Fourier coefficients. The arc length interval for the curve sampling is therefore the total length of the open curve divided by 2^{n-1} where 2^n is the number of sample points on the boundary curve.

Chapter 5. Matching

In the human face profile recognition system, matching is the final process that determines the identity of an unknown person. Preceding the matching process, a Fourier descriptors vector is computed from the face profile of the person under examination. This vector is called a *test vector*, from which a distance measurement is made to each vector (*template vector*) in a database. A match is said to be found when the shortest distance falls below a certain threshold value. The threshold value is a maximum allowable distance for the system to consider a match. It is used to discriminate against people that are not registered in the database.

To quantify the difference between the Fourier descriptors vectors of the face profiles of two persons, we use Euclidean distance measurement. The m-value Euclidean distance between two n-dimensional vectors is given by

$$d_m = \sum_{j=0}^{m-1} [v_1(j) - v_2(j)]^2,$$

where $m \leq n$ and n is the size of vectors v_1 and v_2. The smaller the Euclidean distance is, the closer the two vectors are in the n-dimensional space. To ensure that the Fourier descriptors can truly represent a person's face profile, the Euclidean distance between the Fourier descriptors vectors of two persons should be large. In other words, there should be a significant difference between the two Fourier descriptors vectors.

In the following, we will examine the differences in the Fourier descriptors vectors of people's face profiles as well as the consistency in the Fourier descriptors vectors of the face profiles of the same person. The size of the Fourier descriptors vector, of course, affects the effectiveness of Fourier descriptors in describing human face profiles. For now, we will use a vector size of 64 coefficients for the following tests. Formal evaluations on how the vector size affects the performance of the system will be presented in **Chapter 6. Analysis of the System Performance**.

A. Differences in the Fourier descriptors vectors of people's face profiles

Figure 5.1 shows 4 face profiles and their Fourier descriptors vectors. The size of the Fourier descriptors vector is 64. (i.e. 65 points are sampled from the open curve.) The Euclidean distances between the vectors are shown in **Figure 5.2**. There are 4 tables in **Figure 5.2**. Different numbers of values in the Fourier descriptors vectors are used in the Euclidean distance calculation. In an m-value Euclidean distance calculation, only the first m values of the Fourier descriptors vectors are used. Since the profile of the Fourier descriptors vector diminishes as the index increases, the m-value Euclidean distance between two vectors converges as m increases. For this reason, it is not necessary to use all the values in the Fourier descriptors vector for our Euclidean distance calculation. We can see that the Euclidean distances increase by a fair amount when 16 values are used in the calculation instead of 8. However, the distances do not increase as much when the calculation is switched from 16 to 32 values and from 32 to 64 values. Using only a small fraction of the Fourier descriptors vector for the Euclidean distance calculation also reduces the computation time in the matching process. The effect will become more noticeable if a large database is used.

In **Table (b)** of **Figure 5.2**, a typical value of the Euclidean distance between the Fourier descriptors vectors of two persons is about 1000×10^{-6}. We can choose our threshold value based on this number.

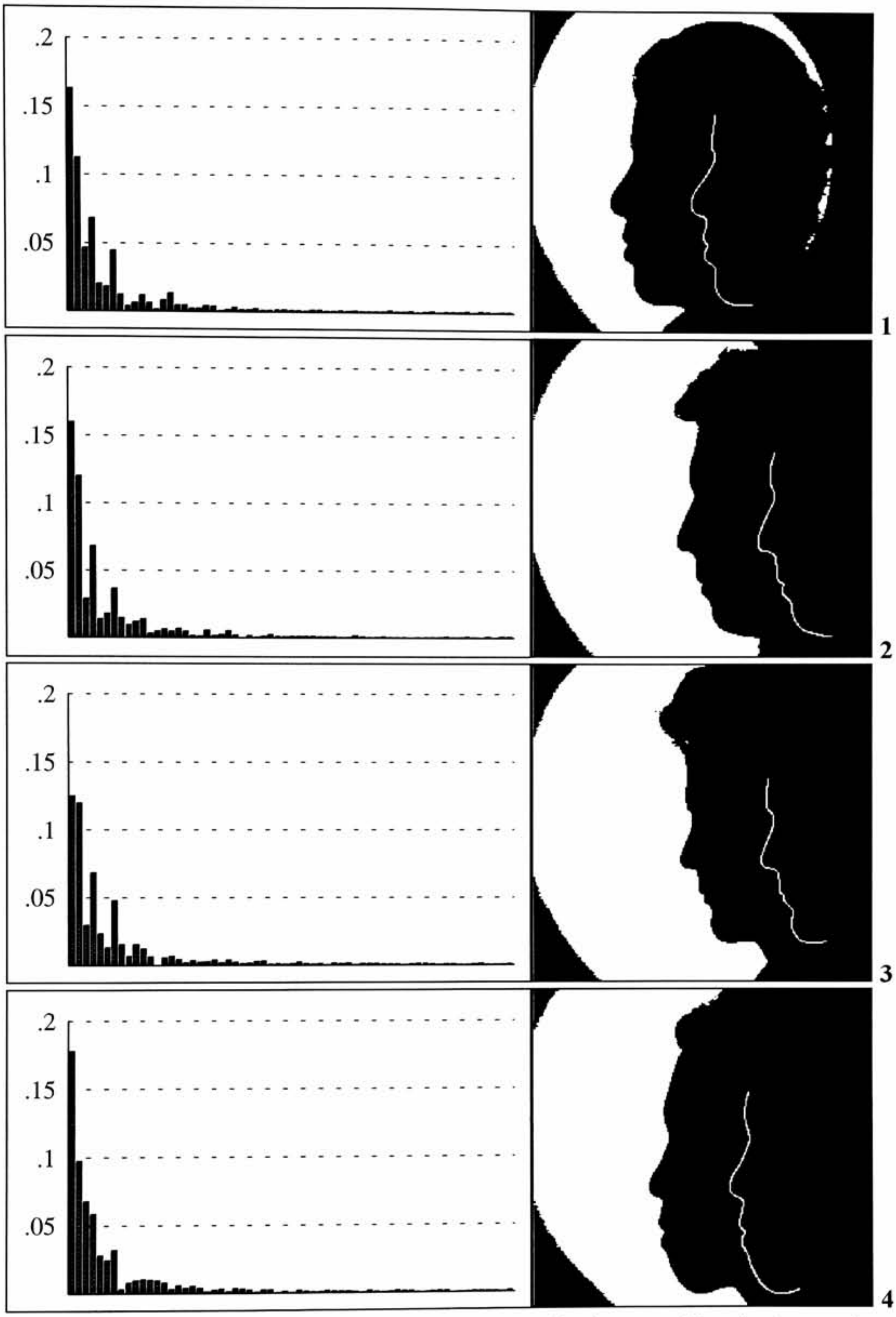


Figure 5.1 *Fourier descriptors vectors of size 64 of 4 face profiles. (c_0 is not shown, which is equal to 1.0.)*

Vectors	*v1*	*v2*	*v3*	*v4*
v1		469	1,857	1,238
v2	469		1,447	2,659
v3	1,857	1,447		5,223
v4	1,238	2,659	5,223	

(a) 8 values are used in the calculations.

Vectors	*v1*	*v2*	*v3*	*v4*
v1		648	2,013	1,532
v2	648		1,506	2,889
v3	2,013	1,506		5,531
v4	1,532	2,889	5,531	

(b) 16 values are used in the calculations.

Vectors	*v1*	*v2*	*v3*	*v4*
v1		688	2,035	1,576
v2	688		1,554	2,965
v3	2,035	1,554		5,570
v4	1,576	2,965	5,570	

(c) 32 values are used in the calculations.

Vectors	*v1*	*v2*	*v3*	*v4*
v1		693	2,041	1,584
v2	693		1,561	2,973
v3	2,041	1,561		5,577
v4	1,584	2,973	5,577	

(d) 64 values are used in calculations.

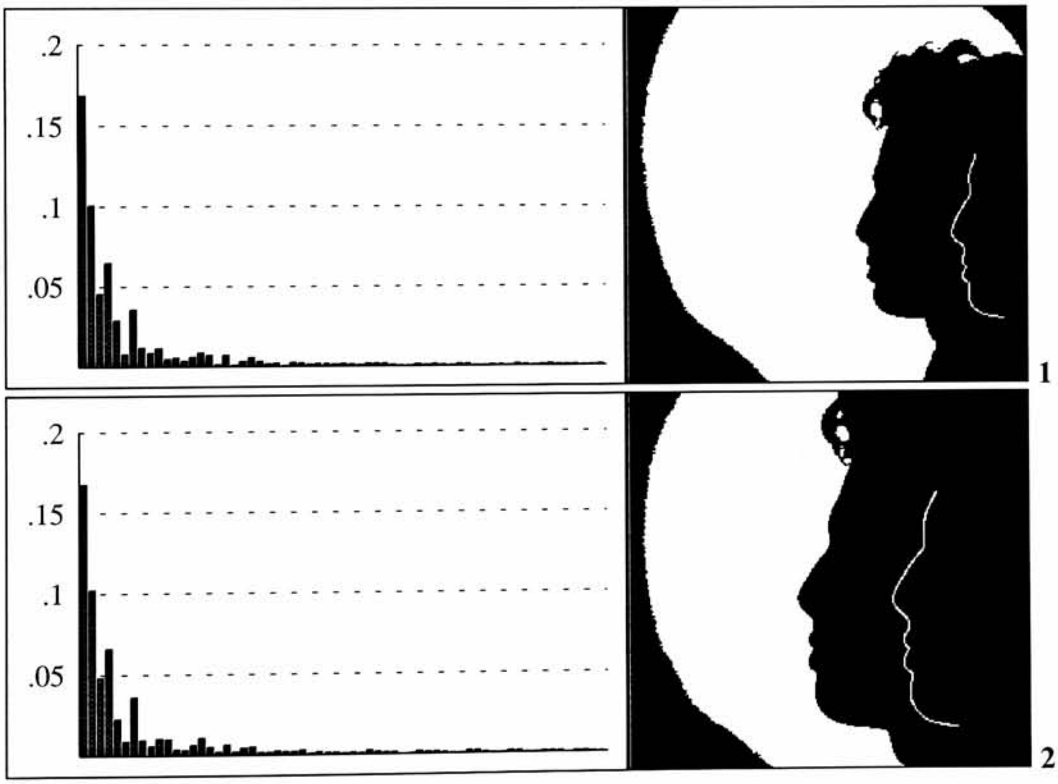
Figure 5.2 *Euclidean distances of 4 Fourier descriptors vectors of size 64. Various numbers of coefficients are used in the calculations. (Values are in 10^{-6}.)*

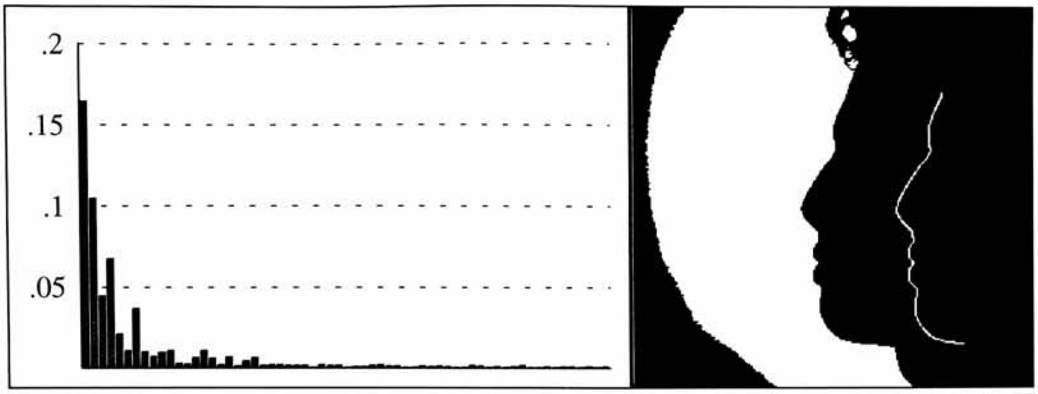
B. Consistency in the Fourier descriptors vectors of the face profiles of the same person

In order to show that Fourier descriptors can be used as the basis for identifying human face profiles, consistent results must be obtained from different images of the same person. In other words, small Euclidean distances must be obtained among different images of the same person.

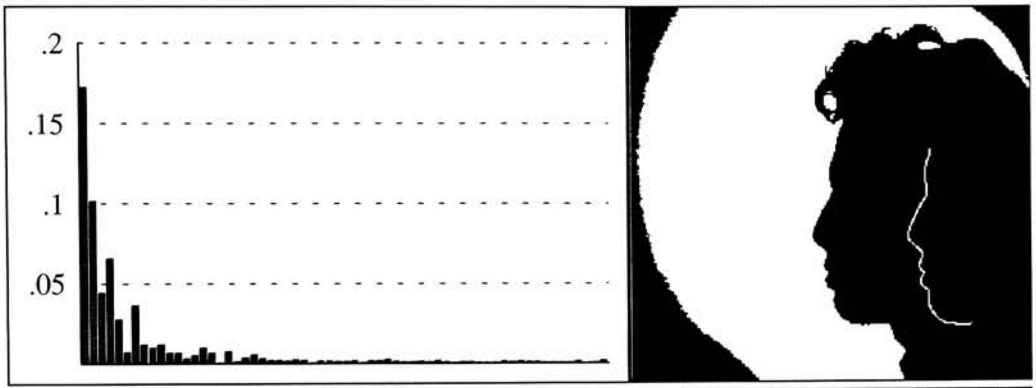
Figure 5.3 shows 7 face profiles of the same person and their corresponding Fourier descriptors vectors. In images 1, 2, 3, and 4, the person was asked to keep his mouth closed naturally. In images 5, 6, and 7, the person was asked to open his mouth gradually. The Euclidean distances between the vectors are shown in **Figure 5.4**. The size of the Fourier descriptors vector is 64. In **Table (a)** of **Figure 5.4**, 16 values are used in the Euclidean distance calculations, while in **Table (b)**, 64 values are used. Once again, we can see that the Euclidean distances do not increase as much when the number of values used in the distance calculation switched from 16 values to 64 values. Further investigation on this respect will be detailed in **Chapter 6. Analysis of System Performance**. Now, recall from the distance comparison of the Fourier descriptors vectors of different people, the typical value of the Euclidean distance of a 16-value comparison using Fourier descriptors of size 64 is 1000×10^{-6}. If we choose 300×10^{-6} as our threshold value, images 1-4 will all fall below this value when comparing with one another. The Euclidean distances from the vectors of images 5, 6, and 7 to the vectors of images 1 to 4 are larger since there are subtle changes in the face profiles when the person's mouth is allowed to open widely. With the open mouth, the curve length of the face profile becomes longer. Since a fixed number of points are sampled from the face profile curve for the vector calculation, the positions of the sampled points are spread out more on the profile curve, causing the Fourier descriptors vector to vary slightly.

In **Figure 5.4**, we can see that there is certainly a consistency in the Fourier descriptors vectors of the face profiles of the same person with the mouth closed naturally. The open mouth images have some impact on the resulting Fourier descriptors, depending on the degree of openness. Therefore, the Euclidean distances from these vectors to the vectors of the naturally posed images are slightly larger.

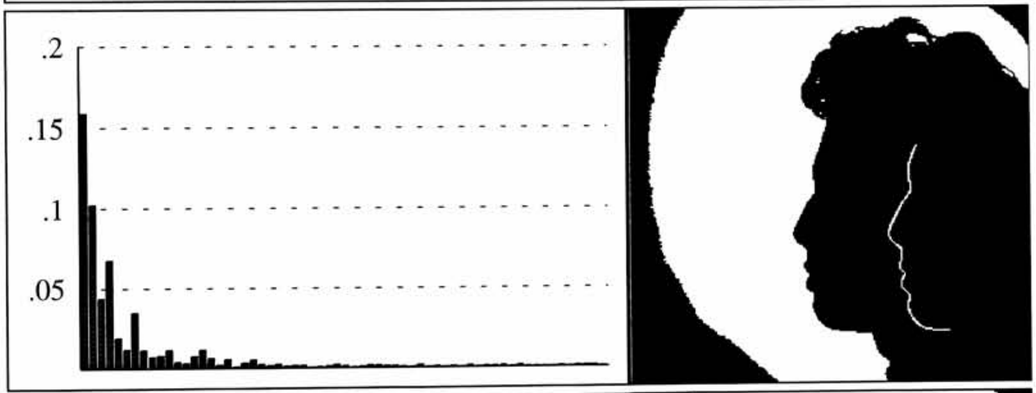




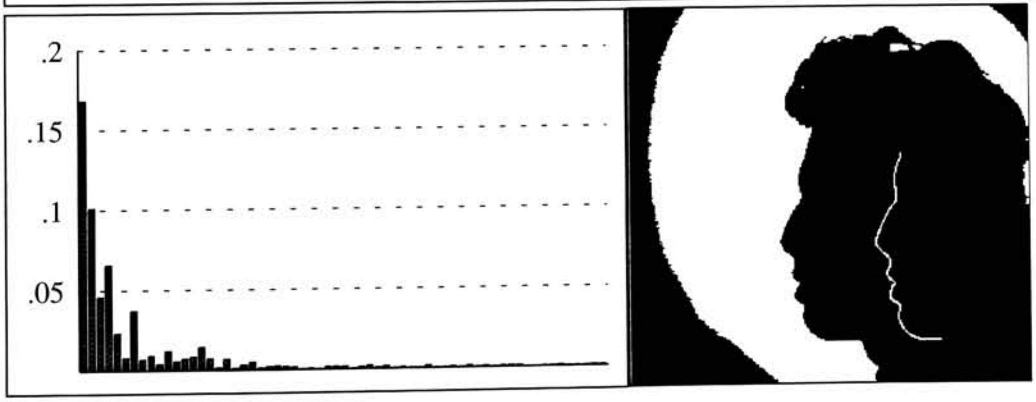
3



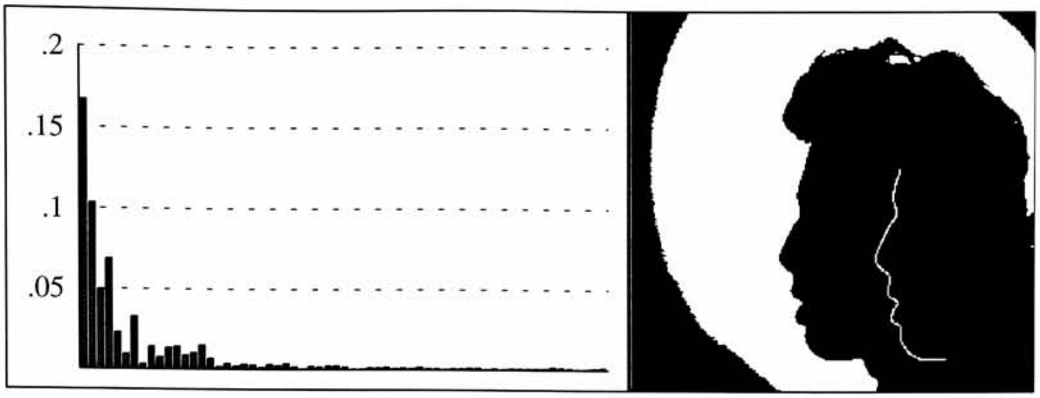
4



5



6



7

Figure 5.3 *Fourier descriptors of size 64 of the face profiles of the same person. (c_0 is not shown, which is equal to 1.0.) The person's mouth is open in pictures 5, 6, and 7. Picture 5 is the least open and picture 7 is the most open.*

Vectors	*v1*	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*
v1		101	178	31	306	224	464
v2	101		44	109	147	112	341
v3	178	44		189	59	141	386
v4	31	109	189		352	199	415
v5	306	147	59	352		206	424
v6	224	112	141	199	206		184
v7	464	341	386	415	424	184	

(a) 16 values are used in the calculations.

Vectors	*v1*	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*
v1		122	192	39	317	243	508
v2	122		52	131	163	132	376
v3	192	52		206	71	159	428
v4	39	131	206		370	218	461
v5	317	163	71	370		217	455
v6	243	132	370	218	217		209
v7	508	376	218	461	455	209	

(b) 64 values are used in the calculations.

Figure 5.4 *Euclidean distances between the Fourier descriptors vectors of the face profiles of the same person. Vectors **v5**, **v6**, and **v7** are computed from images with the person's mouth open. 16 and 64 values are used in the calculations. (Values are in 10^{-6}.)*

Chapter 6. Analysis of the System Performance

The correct recognition rate of the human face profile recognition system decreases as the sample population increases. Besides that, there are other parameters that can affect the performance of the system as well. The two most important ones are the size of the Fourier descriptors vector and the number of coefficients in the vector that are used in the matching process.

A. Sample population and vector size

We begin our investigation by seeing how the sample population affects the recognition rate. At the same time, we investigate the effect of the size of the Fourier descriptors vector on the system performance. First, 24 databases are constructed based on the images obtained from 60 people. Each database is characterized by the number of samples it contains and the size of the Fourier descriptors vector representing each sample. Each vector in the databases is constructed using 3 different images of the same person. To test the system, 4 different images of the same person, which differ from those used to construct the databases, are presented to the system for each corresponding entry in each database. The results of the tests are plotted in **Figure 6.1**.

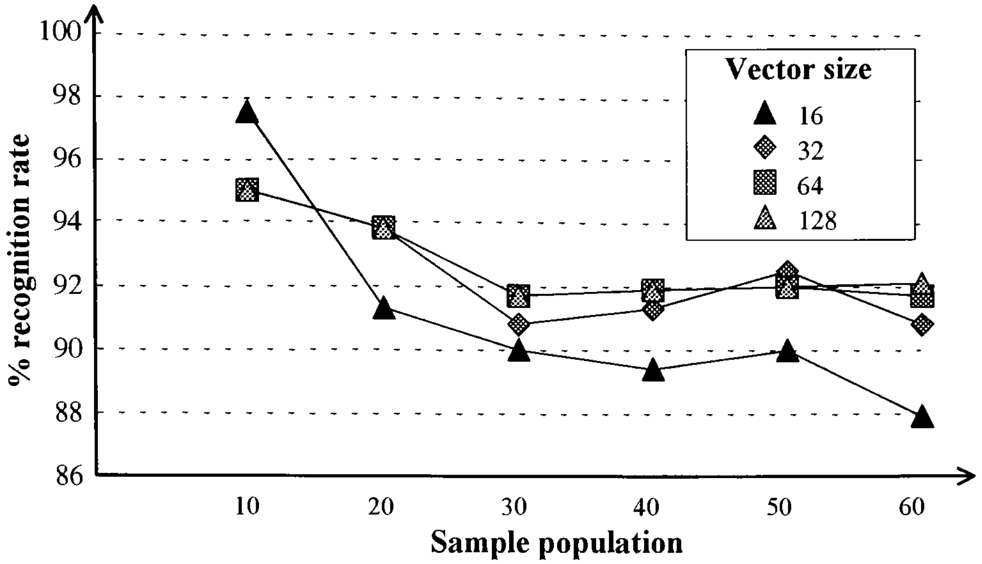


Figure 6.1 *Recognition rate versus sample population with various vector sizes.*

As expected, the correct recognition rate decreases as the sample population increases. Moreover, the larger the vector size, the better the system performance with large sample populations. The recognition rate also seems to converge as the vector size increases. As we can see, there is not a big difference in the system performance between vector sizes 64 and 128.

Now, let us take a close examination of the above tests. Specifically, we choose the test with the vector size of 64 coefficients and the sample population of 60 people. In **Figure 6.2**, the confusion matrix of the test results is shown.

Output

```
   0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0

01 1 2 . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
02 . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
03 . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
04 . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
05 . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
06 . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
07 . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
08 . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
09 . . . . . . . . . 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 .
10 . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
11 . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
12 . . . . . . . . . . . . 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . 1 .
13 . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
14 . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
15 . . . . . . . . . . . . . . . 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . .
16 . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
17 . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
18 . . . . . . . . . . . . . . . . . . 3 . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . .
19 . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
20 . . . . 1 . . . . . . . . . . . . . . . 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
21 . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
22 . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
23 2 . . . . . . . . . . . . . . . . . . . . . . 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
24 . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
25 . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
26 . . . . . . . . . . . . . . . . . . . . . . . . . . 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . . .
27 . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
28 1 . . . . . . 1 . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . .
29 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
30 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
31 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
32 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . .
33 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3 . . . . . . 1 . . . . . . . . . . . . . . . . . . . .
34 . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . 3 . . . . . . . . . . . . . . . . . . . . . . . . . .
35 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . . .
36 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . . .
37 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . . .
38 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . . .
39 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . . .
40 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . . .
41 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . . .
42 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . . .
43 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . . . .
44 . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . 3 . . . . . . . . . . . . . . . .
45 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . . .
46 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . . .
47 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . . .
48 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . . .
49 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . . .
50 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . . .
51 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . . .
52 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . . .
53 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . . .
54 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . . .
55 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . . . .
56 . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3 . . . .
57 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . . .
58 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4 . .
59 . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3 .
60 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Recognition rate: 220 out of 240 (91.7%)
```

Input

Figure 6.2 *Confusion matrix of the test results for the vector size of 64 coefficients and the sample population of 60 people. (Output 0 represents the unidentifiables.)*

The confusion matrix shows the output of the system for the presented input. The numbers along the diagonal of the matrix represent the numbers of correct recognitions for the given input. Let us take a look at person #12. The confusion matrix shows that the system has mistaken him as another person for 2 times, one as person #40 and another one as person #59. Person #33 has also been mistaken by the system as person #40. Similarly, person #9 has also been mistaken by the system as person #59. We shall examine the face profile curves of these people and see what similarities they possess. In **Figure 6.3**, the face profile curves of these people are shown.

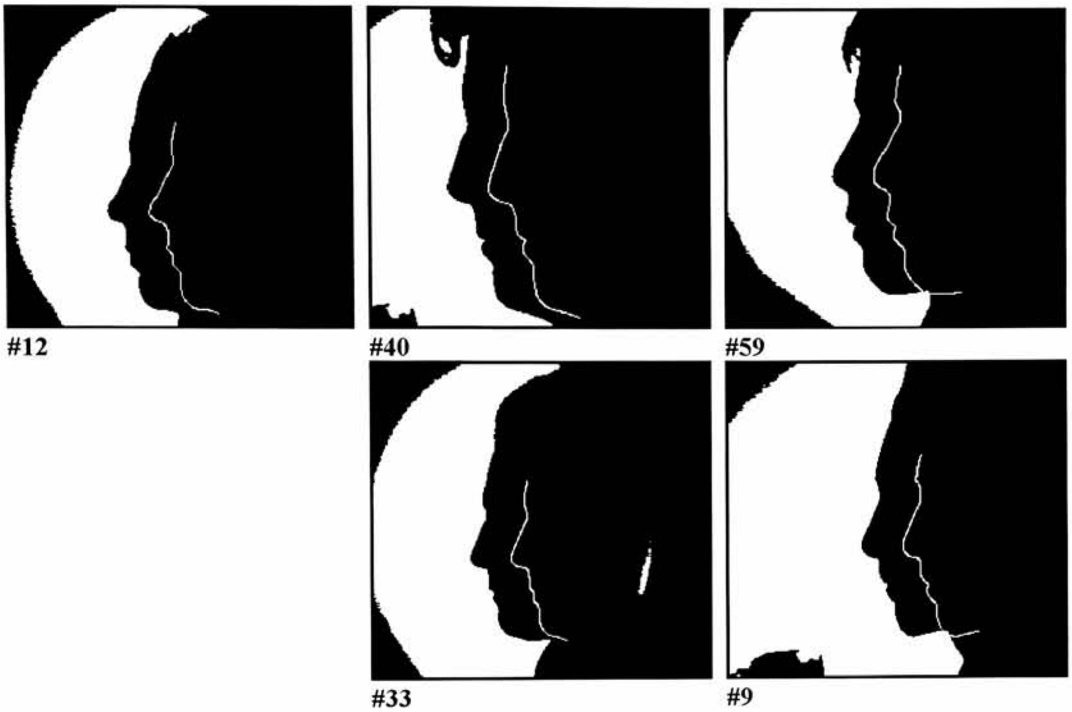


Figure 6.3 *Look-alike face profiles. Person #12 was mistaken as persons #40 and #59. Person #33 was mistaken as person #40. Person #9 was mistaken as person #59.*

At the first glance, these face profile curves may not look alike. However, they all have similar features that make the computer mistake one as the other. First of all, while we are looking at these images, we should remember that the system makes comparisons based on face profile curves only. The face silhouettes of these images obviously do not come close to one another. So, when we compare these images, we should pay more attention to the flow of the profile curves. One of the similarities that may be found among these people is the proportions of the main features on the face profile.

These may include the ratio of the distance between the eye and the bottom of nose to the distance between the bottom of nose and the chin position. On the other hand, the errors made by the system can also be coming from the angular position of the face. Therefore, it is difficult to conclude what causes the faults.

In **Figure 6.4**, the relative distances of all the vectors to that of person #12 are shown. Four people with their vectors which are farthest away in the Euclidean space to that of person #12 are chosen for close examination. The face profile curves for these people are shown in **Figure 6.5**.

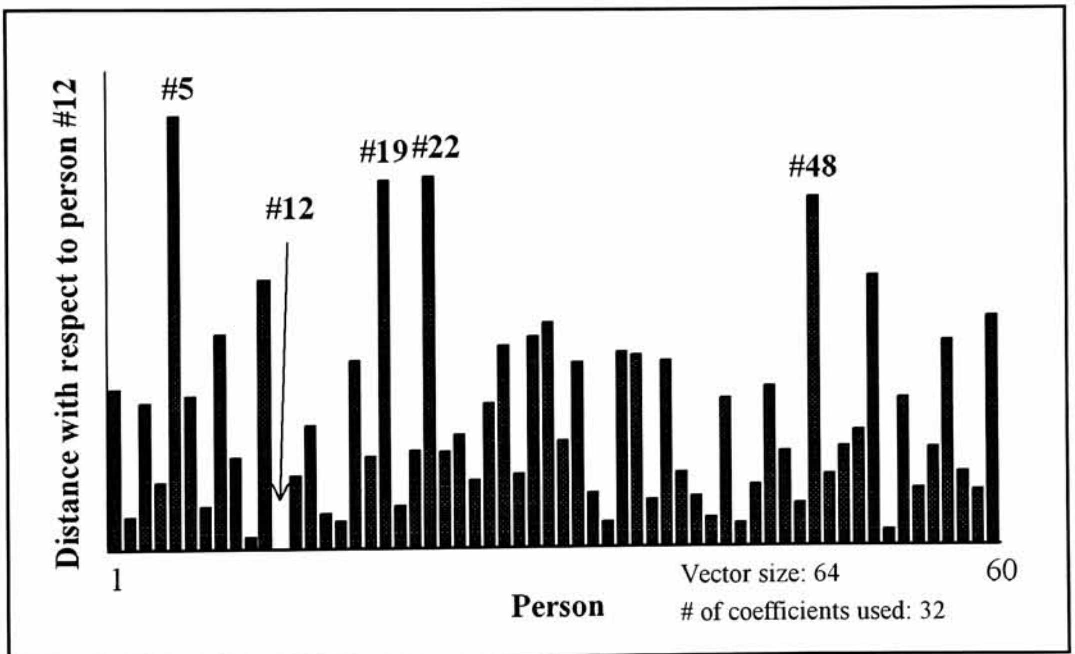


Figure 6.4 *Relative distances of the vectors to that of person #12.*

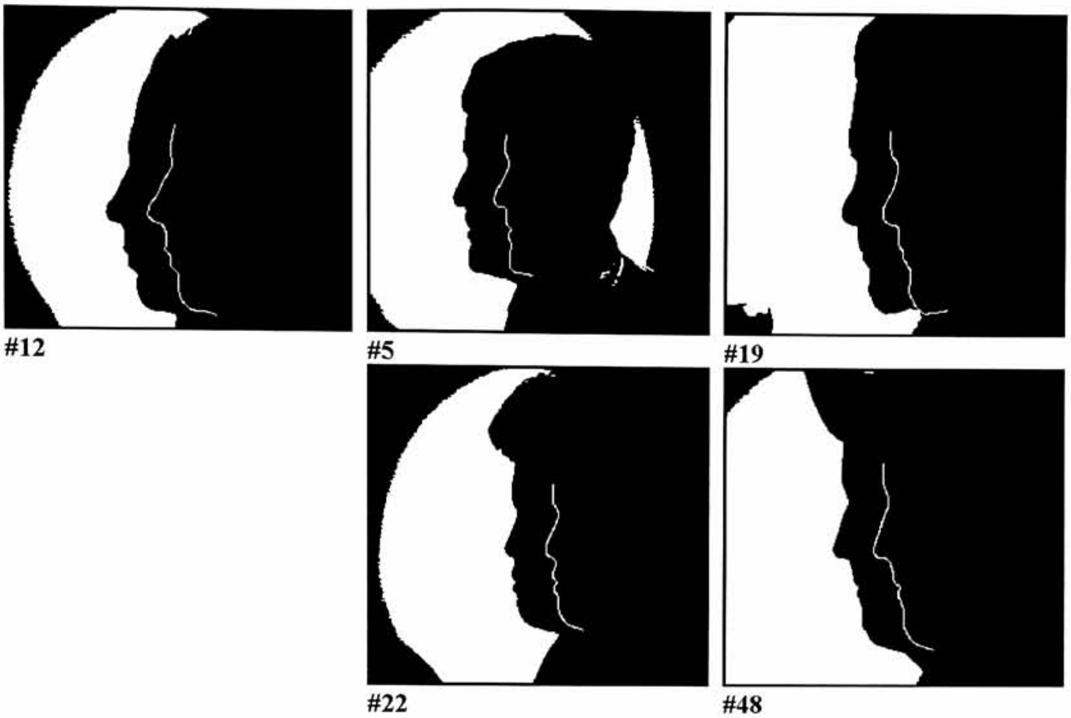


Figure 6.5 *Vectors of the face profiles #5, #19, #22, and #48 are far away from that of the face profile #12 in the Euclidean space. (Euclidean distances are calculated from the database of sample population of 60 people and vector size of 64 coefficients.)*

Now, let us compare these face profile curves. The face profile curve of person number #12 is quite rounded compared with the others. The area covered by the curve and the straight line joining the two terminating points is obviously larger than those of the other face profile curves. Also, the eyebrow area of person #12 is not as distinctive as the rest. Person #5 has a sharp chin and person #19 has a big round chin. The chin of person #12 looks like those of persons #22 and #48. However, the noses of person #22 and #48 are flatter than that of person #12. With all these differences, there is no doubt these 4 face profile curves yield quite different Fourier descriptors vectors compared to that of person #12.

B. Number of coefficients used in the matching process

Another system parameter which directly affects the recognition rate is the number of coefficients in the Fourier descriptors vector that are used in the matching process. Since the profile of the Fourier descriptors vector diminishes as the index of the vector increases, the Euclidean distance between two vectors converges to a stable value as the number of coefficients used in the calculation increases. Therefore, we can find out the number of coefficients that is adequate for computing the Euclidean distance between two vectors. In **Figure 6.6**, the correct recognition rate is plotted against the number of coefficients used in the matching process. We have chosen the vector size of 64 coefficients for the test since it is most appropriate for a large sample population size as determined earlier.

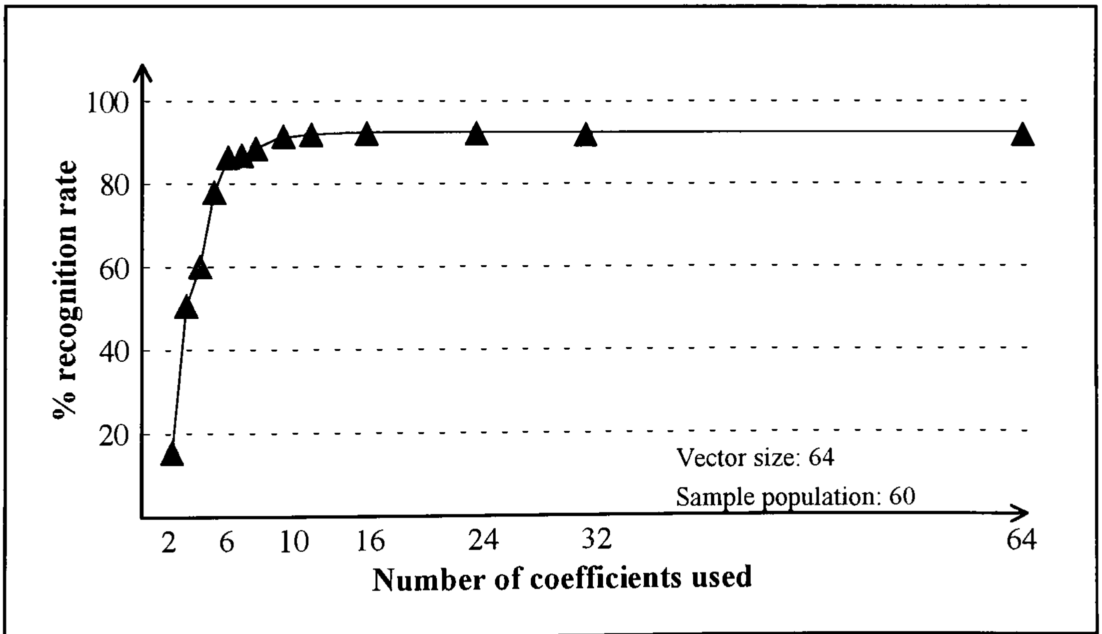


Figure 6.6 *Recognition rate versus the number of coefficients used in the matching process.*

As we can see, the recognition rate increases as the number of coefficients used in the matching process increases and it reaches its maximum at a quarter of the total number of coefficients in the vector. In other words, the recognition rate of the system with a vector size of 64 coefficients does not get any better with more coefficients used in the matching process than that obtained with 16 values.

C. Face direction

In general, the human face profile recognition system requires that the person under examination be facing absolutely perpendicular to the view of the camera. Any deviation from the perpendicular position causes subtle changes or sometimes big changes in the shape of the face profile curve, depending on person to person. In **Table 6.7**, the Euclidean distances between the deviated face profile curves and the straight face profile curves for a few people are shown.

Angle (deg.)	**-10**	**-7.5**	**-5**	**-2.5**	**0**	**2.5**	**5**	**7.5**	**10**
Person #1	2,170	1,186	619	827	0	128	129	71	356
Person #2	3,178	1,345	345	231	0	78	110	164	403
Person #3	2,048	809	509	574	0	109	124	289	558
Person #4	3,809	1,348	467	157	0	89	78	147	438

Table 6.7 *The Euclidean distances between the deviated face profile curves and the straight face profile curve. (The vector size is 64 and 16 values are used in the distance calculation. Values shown are in 10^{-6}.)*

The angular position of the straight face profiles is at zero degrees. Faces that turn towards the camera have positive angular positions and negative for the opposite direction. Notice that there is a difference between the two directions of angular deviation. The face profile curves of people that turn towards the camera are much more tolerable than those of the opposite direction. Intuitively, we think that the face profiles captured at two different directions with the same magnitude of angular deviation should be very comparable. However, as the person's head turns away from the camera, part of the face begins to block or distort some of the essential features found in the straight face profile. These include the nose and the lips. On the other hand, the general shape of the straight face profile suffers minimum distortion as the person's head turns towards the camera at a small angle since both sides of the face hardly obstruct those essential features.

Conclusion

The human face profile recognition system presented in this thesis has clearly shown that it is possible to construct a system to identify human individuals through their face profiles. The use of Fourier descriptors to represent face profile curves has demonstrated its practical application in this domain. Although considerable effort is still required to construct a more robust and convenient system, this thesis has confirmed the ideas and feasibility of building such systems that researchers have developed over the past two decades.

References

[1] E. Persoon and K. S. Fu, "Shape discrimination using Fourier descriptors," *IEEE Trans. Systems, Man, Cybern.*. 7, 170-179 (1977).

[2] C. T. Zahn and R. Z. Roskies, "Fourier descriptors for plane closed curves," *IEEE Trans. Computers*. **21**, 269-281 (1972).

[3] G. H. Granlund, "Fourier preprocessing for hand print character recognition," *IEEE Trans. Computers*. **21**, 195-201 (1972).

[4] T. Aibara, K. Ohue, and Y. Oshita, "Human face profile recognition by a P-Fourier descriptor," *Optical Engineering*. **32**(4), 861-863 (1993).

[5] T. Aibara, K. Ohue, and Y. Matsuoka, "Human face profile recognition by a P-type Fourier descriptor," in *Visual Communications and Image Processing '91: Image Processing, Proc. SPIE*. **1606**, 198-203 (1991).

[6] C. J. Wu and J. S. Huang, "Human face profile recognition by computer," *Pattern Recognition*. **23**(3/4), 255-259 (1990).

[7] L. D. Harmon, S. C. Kuo, P. F. Ramig and U. Raudkivi, "Identification of human face profiles by computer," *Pattern Recognition*. **10**, 301-312 (1978).

[8] L. D. Harmon, M. K. Khan, R. Lasch and P. F. Ramig, "Machine identification of human faces," *Pattern Recognition*. **13**, 97-110 (1981).

[9] F. Galton, "Numeralized profiles for classification and description," *Nature*. **83**(2109), 31 March, 127-130 (1910).

[10] F. Galton, "Personal identification and description," *Nature*. 21 June, 173-177 (1888); 28 June, 201-202 (1888).

[11] L. D. Harmon and W. F. Hunt, "Automatic recognition of human face profiles," *Computer Graphics and Image Processing*. **6**(2), 135-156 (1977).

[12] A. Samal and P. A. Iyengar, "Automatic recognition and analysis of human faces and facial expressions: A survey," *Pattern Recognition*. **25**(1), 65-77 (1992).

[13] A. J. Goldstein, L. D. Harmon and A. B. Lesk, "Identification of human faces," *Proc. IEEE*. **59**, 748-760 (1971).

[14] Y. Kaya and K. Kobayashi, "A basic study on human face recognition," *Frontiers of Pattern Recognition*. 265-289. Academic Press, New York (1971).

[15] K. H. Wong, H. H. M. Law and P. W. M. Tsang, "A system for recognizing human faces," *Proc. ICASSP 89*, 1638-1642 (1989).

[16] R. J. Baron, "Mechanisms of human facial recognition," *Int. J. Man-Mach. Stud.*. **15**, 137-178 (1981).

Appendix

Program Listings

This appendix contains the program listings of the human face profile recognition system. The two main modules are `matching.c` and `training.c` for the identification process and the registration process respectively. The rest of the supporting modules are listed below with brief descriptions provided.

Module name	Description
`classify.c` `classify.h`	Matching process.
`complex.h`	Macros for working with complex numbers.
`costable.c`	Table to use for cosine and sine table lookups.
`curvelib.c` `curvelib.h`	Chain code arithematics.
`database.c` `database.h`	Database manipulations.
`extract.c` `extract.h`	Curve extraction with chain code.
`feature.c` `feature.h`	Feature detection and location.
`fft.c` `fft.h`	Fast Fourier transform.
`fgrabber.c` `fgrabber.h`	Frame-grabber board operations.
`filters.c` `filters.h`	Filters for the Turning point detection algorithm.
`global.c` `global.h`	Global variables.
`humanf.h`	Useful macros and definitions for the system.
`image.c` `image.h`	Image file operations for non-realtime processing.
`sampling.c` `sampling.h`	Curve sampling and display.
`vector.c` `vector.h`	Highest in the program hierarchy besides the main programs.

File: classify.c

```c
/*----------------------------------------------------------------------
**	File:		classify.c
**	Intent:		Feature vector classification.
**	Routines:	int	classify( DATABASE, double *, int, double * )
*/

#include "..\include\humanf.h"
#include "..\database\database.h"


/*----------------------------------------------------------------------
**	Function:	classify
**	Intent:		Find out the closest vector in the database to the
**			input vector.
**	Arguments:	database	the database contains the templates.
**			vector		the input vector.
**			vsize		vector size.
**			distance	the distance from each vector in the
**					database to the test vector.
**
**	Return value:	the vector number (index) of the closest match.
*/
int
classify( database, vector, vsize, distance )
DATABASE	database;
double		*vector;
int		vsize;
double		*distance;
{
	static double	e_distance( double *, double *, int );

	double	dist;			/* distance between 2 vectors */
	double	min_dist = HUGE_VAL;	/* minimum distance computed so far */
	int	min_index, i;		/* vector number in database */

	for ( i = 0; i < database.nvectors; i++ ) {
		distance[i] = dist =
		e_distance( vector, database.vector[i], vsize );

		if ( dist < min_dist ) {
			min_dist = dist;
			min_index = i;
		}
	}

	return( min_index );
}


/*----------------------------------------------------------------------
**	Function:	e_distance
**	Intent:		Compute the Euclidean distance between two vectors.
**	Arguments:	x, y	input vectors.
**			n	size of vector.
**	Return value:	the distance between the two input vectors.
*/
static double
e_distance( x, y, n )
double	*x, *y;
int	n;
{
	double	diff;			/* difference between 2 elements */
	double	dist = 0.0;
	int	i;

	for ( i = 0; i < n; i++ ) {
		diff = x[i] - y[i];
		dist += diff*diff;
	}

	return( dist );
}
```

File: classify.h

```c
/*----------------------------------------------------------------------
**	File:		classify.h
**	Intent:		Feature vector classification function declarations.
*/

#ifndef _CLASSIFY_H
#define _CLASSIFY_H

extern	int	classify( DATABASE, double *, int, double * );

#endif
```

File: complex.h

```c
/*----------------------------------------------------------------------
**	File:		complex.h
**	Intent:		Macros for working with complex numbers.
*/

#include <math.h>

#ifndef _COMPLEX_H
#define _COMPLEX_H

typedef struct complex	COMPLEX;

#define COMPLEX(c,a,b) { \
	(c).x = (a); \
	(c).y = (b); \
}
```

```c
#define COMPLEX_ASSIGN(a,b) { \
	(a).x = (b).x; \
	(a).y = (b).y; \
}

#define COMPLEX_CONJ(a)	(a).y = -(a).y

#define COMPLEX_ADD(c,a,b) { \
	(c).x = (a).x + (b).x; \
	(c).y = (a).y + (b).y; \
}

#define COMPLEX_SUB(c,a,b) { \
	(c).x = (a).x - (b).x; \
	(c).y = (a).y - (b).y; \
}

#define COMPLEX_MULT(c,a,b) { \
	(c).x = (a).x*(b).x - (a).y*(b).y; \
	(c).y = (a).x*(b).y + (a).y*(b).x; \
}

#define COMPLEX_MAG(a,b)  (b) = sqrt( (a).x*(a).x + (a).y*(a).y )

#endif
```

File: costable.c

```c
/*----------------------------------------------------------------------
**	File:		costable.c
**	Intent:		This file contains a table to use for cosine and sine
**			table lookups. It contain 1024 entries of cosine
**			values between 0 and 2*PI.
**
**			Table of 1024 cosine values in double form:
**				cos_table[n] = cos(2*PI*n/1024)
**
**			This can be used by itself with the macros
**			COS and SIN defined in fft.h.
*/

double	cos_table[] =
{
1.000000000000000,0.999981175282601,0.999924701839145,0.999830581795823,
0.999698818696204,0.999529417501093,0.999322384588350,0.999077727752645,
0.998795456205172,0.998475580573295,0.998118112900149,0.997723066644192,
0.997290456678690,0.996820299291166,0.996312612182778,0.995767414467660,
0.995184726672197,0.994564570734255,0.993906970002356,0.993211949234795,
0.992479534598710,0.991709753669100,0.990902635427780,0.990058210262297,
0.989176509964781,0.988257567730750,0.987301418157858,0.986308097244599,
0.985277642388941,0.984210092386929,0.983105487431216,0.981963869109555,
0.980785280403230,0.979569765685441,0.978317370719628,0.977028142657754,
0.975702130038529,0.974339382785576,0.972939952205560,0.971503890986252,
0.970031253194544,0.968522094274417,0.966976471044852,0.965394441697689,
0.963776065795440,0.962121404269042,0.960430519415566,0.958703474895872,
0.956940335732209,0.955141168305771,0.953306040354194,0.951435020969008,
0.949528180593037,0.947585591017741,0.945607325380521,0.943593458161960,
0.941544065183021,0.939459223602190,0.937339011912575,0.935183509938948,
0.932992798834739,0.930766961078984,0.928506080473216,0.926210242138311,
0.923879532511287,0.921514039342042,0.919113851690058,0.916679059921043,
0.914209755703531,0.911706032005430,0.909167983090522,0.906595704514915,
0.903989293123443,0.901348847046022,0.898674465693954,0.895966249756185,
0.893224301195515,0.890448723244758,0.887639620402854,0.884797098430938,
0.881921264348355,0.879012226428634,0.876070094195407,0.873094978418290,
0.870086991108712,0.867046245515693,0.863972856121587,0.860866938637767,
0.857728610000272,0.854557988365401,0.851355193105265,0.848120344803297,
0.844853565249707,0.841554977436898,0.838224705554838,0.834862874986380,
0.831469612302545,0.828045045257756,0.824589302785025,0.821102514991105,
0.817584813151584,0.814036329705948,0.810457198252595,0.806847553543799,
0.803207531480645,0.799537269107905,0.795836904608884,0.792106577300212,
0.788346427626606,0.784556597155575,0.780737228572095,0.776888465673232,
0.773010453362737,0.769103337645580,0.765167265622459,0.761202385484262,
0.757208846506485,0.753186799043613,0.749136394523459,0.745057785441466,
0.740951125354959,0.736816568877370,0.732654271672413,0.728464390448225,
0.724247082951467,0.720002507961382,0.715730825283819,0.711432195745216,
0.707106781186548,0.702754744457225,0.698376249408973,0.693971460889654,
0.689540544737067,0.685083667772700,0.680600997795453,0.676092703575316,
0.671558954847018,0.666999922303638,0.662415777590172,0.657806693297079,
0.653172842953777,0.648514401022113,0.643831542889792,0.639124444863776,
0.634393284163646,0.629638238914927,0.624859488142387,0.620057211763289,
0.615231590580627,0.610382806276310,0.605511041404326,0.600616479383869,
0.595699304492434,0.590759701858874,0.585797857456439,0.580813958095765,
0.575808191417845,0.570780745886967,0.565731810783613,0.560661576197336,
0.555570233019602,0.550457972936605,0.545324988422047,0.540171472729893,
0.534997619887097,0.529803624686295,0.524589682678469,0.519355990165590,
0.514102744193222,0.508830142543107,0.503538383725718,0.498227666972782,
0.492898192229784,0.487550160148436,0.482183772079123,0.476799230063322,
0.471396736825998,0.465976495767966,0.460538710958240,0.455083587126344,
0.449611329654607,0.444122144570429,0.438616238538528,0.433093818853152,
0.427555093430282,0.422000270799800,0.416429560097637,0.410843171057904,
0.405241314004990,0.399624199845647,0.393992040061048,0.388345046698826,
0.382683432365090,0.377007410216418,0.371317193951838,0.365612997804774,
0.359895036534988,0.354163525420491,0.348418680249435,0.342660717311994,
0.336889853392220,0.331106305759876,0.325310292162263,0.319502030816016,
0.313681740398892,0.307849640041535,0.302005949319228,0.296150888243624,
0.290284677254462,0.284407537211272,0.278519689385053,0.272621355449949,
0.266712757474898,0.260794117915276,0.254865659604515,0.248927605745720,
0.242980179903264,0.237023605994367,0.231058108280671,0.225083911359793,
0.219101240156870,0.213110319916091,0.207111376192219,0.201104634842092,
0.195090322016128,0.189068664149806,0.183039887955141,0.177004220412149,
0.170961888760301,0.164913120489970,0.158858143333861,0.152797185258443,
0.146730474455362,0.140658239332849,0.134580708507126,0.128498110793793,
0.122410675199216,0.116318630911905,0.110222207293883,0.104121633872055,
0.098017140329561,0.091908956497133,0.085797312344440,0.079682437971430,
0.073564563599667,0.067443919563664,0.061320736302209,0.055195244349690,
0.049067674327418,0.042938256934941,0.036807222941359,0.030674803176637,
0.024541228522912,0.018406729905805,0.012271538285720,0.006135884649155,
0.000000000000000,-0.006135884649154,-0.012271538285720,-0.018406729905805,
-0.024541228522912,-0.030674803176636,-0.036807222941359,-0.042938256934941,
-0.049067674327418,-0.055195244349690,-0.061320736302209,-0.067443919563664,
-0.073564563599667,-0.079682437971430,-0.085797312344440,-0.091908956497133,
-0.098017140329561,-0.104121633872055,-0.110222207293883,-0.116318630911905,
-0.122410675199216,-0.128498110793793,-0.134580708507126,-0.140658239332849,
-0.146730474455362,-0.152797185258443,-0.158858143333861,-0.164913120489970,
-0.170961888760301,-0.177004220412149,-0.183039887955141,-0.189068664149806,
-0.195090322016128,-0.201104634842092,-0.207111376192218,-0.213110319916091,
-0.219101240156870,-0.225083911359793,-0.231058108280671,-0.237023605994367,
-0.242980179903264,-0.248927605745720,-0.254865659604515,-0.260794117915276,
-0.266712757474898,-0.272621355449949,-0.278519689385053,-0.284407537211272,
```

```
-0.290284677254462,-0.296150888243624,-0.302005949319228,-0.307849640041535,
-0.313681740398891,-0.319502030816016,-0.325310292162263,-0.331106305759876,
-0.336889853392220,-0.342660717311994,-0.348418680249434,-0.354163525420490,
-0.359895036534988,-0.365612997804774,-0.371317193951838,-0.377007410216418,
-0.382683432365090,-0.388345046698826,-0.393992040061048,-0.399624199845647,
-0.405241314004990,-0.410843171057904,-0.416429560097637,-0.422000270799800,
-0.427555093430282,-0.433093818853152,-0.438616238538527,-0.444122144570429,
-0.449611329654607,-0.455083587126344,-0.460538710958240,-0.465976495767966,
-0.471396736825998,-0.476799230063322,-0.482183772079123,-0.487550160148436,
-0.492898192229784,-0.498227666972782,-0.503538383725718,-0.508830142543107,
-0.514102744193222,-0.519355990165590,-0.524589682678469,-0.529803624686295,
-0.534997619887097,-0.540171472729893,-0.545324988422046,-0.550457972936605,
-0.555570233019602,-0.560661576197336,-0.565731810783613,-0.570780745886967,
-0.575808191417845,-0.580813958095764,-0.585797857456439,-0.590759701858874,
-0.595699304492433,-0.600616479383869,-0.605511041404325,-0.610382806276310,
-0.615231590580627,-0.620057211763289,-0.624859488142386,-0.629638238914927,
-0.634393284163645,-0.639124444863776,-0.643831542889791,-0.648514401022112,
-0.653172842953777,-0.657806693297079,-0.662415777590172,-0.666999922303637,
-0.671558954847018,-0.676092703575316,-0.680600997795453,-0.685083667772700,
-0.689540544737067,-0.693971460889654,-0.698376249408973,-0.702754744457225,
-0.707106781186548,-0.711432195745216,-0.715730825283819,-0.720002507961382,
-0.724247082951467,-0.728464390448225,-0.732654271672413,-0.736816568877370,
-0.740951125354959,-0.745057785441466,-0.749136394523459,-0.753186799043612,
-0.757208846506485,-0.761202385484262,-0.765167265622459,-0.769103337645580,
-0.773010453362737,-0.776888465673232,-0.780737228572095,-0.784556597155575,
-0.788346427626606,-0.792106577300212,-0.795836904608884,-0.799537269107905,
-0.803207531480645,-0.806847553543799,-0.810457198252595,-0.814036329705948,
-0.817584813151584,-0.821102514991105,-0.824589302785025,-0.828045045257756,
-0.831469612302545,-0.834862874986380,-0.838224705554838,-0.841554977436898,
-0.844853565249707,-0.848120344803297,-0.851355193105265,-0.854557988365401,
-0.857728610000272,-0.860866938637767,-0.863972856121587,-0.867046245515693,
-0.870086991108711,-0.873094978418290,-0.876070094195407,-0.879012226428634,
-0.881921264348355,-0.884797098430938,-0.887639620402854,-0.890448723244758,
-0.893224301195515,-0.895966249756185,-0.898674465693954,-0.901348847046022,
-0.903989293123443,-0.906595704514915,-0.909167983090522,-0.911706032005430,
-0.914209755703531,-0.916679059921043,-0.919113851690058,-0.921514039342042,
-0.923879532511287,-0.926210242138311,-0.928506080473216,-0.930766961078984,
-0.932992798834739,-0.935183509938948,-0.937339011912575,-0.939459223602190,
-0.941544065183021,-0.943593458161960,-0.945607325380521,-0.947585591017741,
-0.949528180593037,-0.951435020969008,-0.953306040354194,-0.955141168305771,
-0.956940335732209,-0.958703474895872,-0.960430519415565,-0.962121404269042,
-0.963776065795440,-0.965394441697689,-0.966976471044852,-0.968522094274417,
-0.970031253194544,-0.971503890986252,-0.972939952205560,-0.974339382785576,
-0.975702130038529,-0.977028142657754,-0.978317370719628,-0.979569765685441,
-0.980785280403230,-0.981963869109555,-0.983105487431216,-0.984210092386929,
-0.985277642388941,-0.986308097244599,-0.987301418157858,-0.988257567730750,
-0.989176509964781,-0.990058210262297,-0.990902635427780,-0.991709753669100,
-0.992479534598710,-0.993211949234795,-0.993906970002356,-0.994564570734255,
-0.995184726672197,-0.995767414467660,-0.996312612182778,-0.996820299291166,
-0.997290456678690,-0.997723066644192,-0.998118112900149,-0.998475580573295,
-0.998795456205172,-0.999077727752645,-0.999322384588350,-0.999529417501093,
-0.999698818696204,-0.999830581795823,-0.999924701839145,-0.999981175282601,
-1.000000000000000,-0.999981175282601,-0.999924701839145,-0.999830581795823,
-0.999698818696204,-0.999529417501093,-0.999322384588350,-0.999077727752645,
-0.998795456205172,-0.998475580573295,-0.998118112900149,-0.997723066644192,
-0.997290456678690,-0.996820299291166,-0.996312612182778,-0.995767414467660,
-0.995184726672197,-0.994564570734256,-0.993906970002356,-0.993211949234795,
-0.992479534598710,-0.991709753669100,-0.990902635427780,-0.990058210262297,
-0.989176509964781,-0.988257567730750,-0.987301418157858,-0.986308097244599,
-0.985277642388941,-0.984210092386929,-0.983105487431216,-0.981963869109555,
-0.980785280403230,-0.979569765685441,-0.978317370719628,-0.977028142657754,
-0.975702130038529,-0.974339382785576,-0.972939952205560,-0.971503890986252,
-0.970031253194544,-0.968522094274417,-0.966976471044852,-0.965394441697689,
-0.963776065795440,-0.962121404269042,-0.960430519415566,-0.958703474895872,
-0.956940335732209,-0.955141168305771,-0.953306040354194,-0.951435020969008,
-0.949528180593037,-0.947585591017741,-0.945607325380521,-0.943593458161960,
-0.941544065183021,-0.939459223602190,-0.937339011912575,-0.935183509938948,
-0.932992798834739,-0.930766961078984,-0.928506080473216,-0.926210242138312,
-0.923879532511287,-0.921514039342042,-0.919113851690058,-0.916679059921043,
-0.914209755703531,-0.911706032005430,-0.909167983090523,-0.906595704514915,
-0.903989293123443,-0.901348847046022,-0.898674465693954,-0.895966249756185,
-0.893224301195515,-0.890448723244758,-0.887639620402854,-0.884797098430938,
-0.881921264348355,-0.879012226428634,-0.876070094195407,-0.873094978418290,
-0.870086991108712,-0.867046245515693,-0.863972856121587,-0.860866938637767,
-0.857728610000272,-0.854557988365401,-0.851355193105265,-0.848120344803297,
-0.844853565249707,-0.841554977436898,-0.838224705554838,-0.834862874986380,
-0.831469612302546,-0.828045045257756,-0.824589302785025,-0.821102514991105,
-0.817584813151584,-0.814036329705949,-0.810457198252595,-0.806847553543799,
-0.803207531480645,-0.799537269107905,-0.795836904608884,-0.792106577300212,
-0.788346427626606,-0.784556597155575,-0.780737228572095,-0.776888465673232,
-0.773010453362737,-0.769103337645580,-0.765167265622459,-0.761202385484262,
-0.757208846506485,-0.753186799043613,-0.749136394523459,-0.745057785441466,
-0.740951125354959,-0.736816568877370,-0.732654271672413,-0.728464390448225,
-0.724247082951467,-0.720002507961382,-0.715730825283819,-0.711432195745217,
-0.707106781186548,-0.702754744457225,-0.698376249408973,-0.693971460889654,
-0.689540544737067,-0.685083667772700,-0.680600997795453,-0.676092703575316,
-0.671558954847019,-0.666999922303638,-0.662415777590172,-0.657806693297079,
-0.653172842953777,-0.648514401022112,-0.643831542889792,-0.639124444863776,
-0.634393284163646,-0.629638238914927,-0.624859488142387,-0.620057211763289,
-0.615231590580627,-0.610382806276309,-0.605511041404326,-0.600616479383869,
-0.595699304492433,-0.590759701858874,-0.585797857456439,-0.580813958095765,
-0.575808191417845,-0.570780745886967,-0.565731810783614,-0.560661576197337,
-0.555570233019602,-0.550457972936605,-0.545324988422047,-0.540171472729893,
-0.534997619887097,-0.529803624686295,-0.524589682678469,-0.519355990165589,
-0.514102744193222,-0.508830142543107,-0.503538383725718,-0.498227666972782,
-0.492898192229784,-0.487550160148436,-0.482183772079123,-0.476799230063322,
-0.471396736825998,-0.465976495767967,-0.460538710958240,-0.455083587126344,
-0.449611329654607,-0.444122144570430,-0.438616238538528,-0.433093818853152,
-0.427555093430283,-0.422000270799800,-0.416429560097637,-0.410843171057904,
-0.405241314004990,-0.399624199845647,-0.393992040061048,-0.388345046698827,
-0.382683432365090,-0.377007410216418,-0.371317193951838,-0.365612997804774,
-0.359895036534988,-0.354163525420491,-0.348418680249435,-0.342660717311995,
-0.336889853392220,-0.331106305759877,-0.325310292162263,-0.319502030816015,
-0.313681740398892,-0.307849640041535,-0.302005949319229,-0.296150888243624,
-0.290284677254462,-0.284407537211272,-0.278519689385054,-0.272621355449949,
-0.266712757474899,-0.260794117915276,-0.254865659604514,-0.248927605745720,
-0.242980179903264,-0.237023605994368,-0.231058108280671,-0.225083911359793,
-0.219101240156870,-0.213110319916092,-0.207111376192219,-0.201104634842092,
-0.195090322016129,-0.189068664149806,-0.183039887955141,-0.177004220412149,
-0.170961888760302,-0.164913120489970,-0.158858143333862,-0.152797185258444,
-0.146730474455362,-0.140658239332849,-0.134580708507126,-0.128498110793794,
-0.122410675199216,-0.116318630911905,-0.110222207293883,-0.104121633872055,
-0.098017140329560,-0.091908956497133,-0.085797312344440,-0.079682437971431,
-0.073564563599667,-0.067443919563664,-0.061320736302209,-0.055195244349690,
-0.049067674327418,-0.042938256934941,-0.036807222941359,-0.030674803176636,
-0.024541228522912,-0.018406729905805,-0.012271538285721,-0.006135884649154,
-0.000000000000000,0.006135884649154,0.012271538285720,0.018406729905805,
0.024541228522912,0.030674803176636,0.036807222941359,0.042938256934941,
0.049067674327418,0.055195244349689,0.061320736302209,0.067443919563664,
0.073564563599667,0.079682437971430,0.085797312344440,0.091908956497132,
0.098017140329560,0.104121633872055,0.110222207293883,0.116318630911904,
0.122410675199216,0.128498110793793,0.134580708507126,0.140658239332849,
0.146730474455362,0.152797185258443,0.158858143333861,0.164913120489969,
0.170961888760301,0.177004220412149,0.183039887955141,0.189068664149806,
0.195090322016128,0.201104634842092,0.207111376192218,0.213110319916092,
0.219101240156870,0.225083911359793,0.231058108280671,0.237023605994367,
0.242980179903264,0.248927605745720,0.254865659604514,0.260794117915276,
0.266712757474898,0.272621355449949,0.278519689385053,0.284407537211272,
0.290284677254462,0.296150888243623,0.302005949319228,0.307849640041535,
0.313681740398891,0.319502030816015,0.325310292162263,0.331106305759876,
0.336889853392220,0.342660717311995,0.348418680249435,0.354163525420490,
0.359895036534988,0.365612997804774,0.371317193951837,0.377007410216418,
0.382683432365090,0.388345046698826,0.393992040061048,0.399624199845646,
0.405241314004990,0.410843171057904,0.416429560097637,0.422000270799799,
0.427555093430282,0.433093818853152,0.438616238538527,0.444122144570429,
0.449611329654607,0.455083587126344,0.460538710958240,0.465976495767966,
0.471396736825998,0.476799230063322,0.482183772079122,0.487550160148436,
0.492898192229784,0.498227666972782,0.503538383725718,0.508830142543107,
0.514102744193222,0.519355990165589,0.524589682678469,0.529803624686295,
0.534997619887097,0.540171472729892,0.545324988422047,0.550457972936605,
0.555570233019602,0.560661576197336,0.565731810783613,0.570780745886967,
0.575808191417845,0.580813958095765,0.585797857456439,0.590759701858874,
0.595699304492433,0.600616479383869,0.605511041404325,0.610382806276309,
0.615231590580627,0.620057211763289,0.624859488142386,0.629638238914927,
0.634393284163646,0.639124444863776,0.643831542889791,0.648514401022112,
0.653172842953777,0.657806693297079,0.662415777590172,0.666999922303638,
0.671558954847018,0.676092703575316,0.680600997795453,0.685083667772701,
0.689540544737067,0.693971460889654,0.698376249408972,0.702754744457225,
0.707106781186547,0.711432195745216,0.715730825283819,0.720002507961382,
0.724247082951467,0.728464390448225,0.732654271672413,0.736816568877370,
0.740951125354959,0.745057785441466,0.749136394523459,0.753186799043612,
0.757208846506484,0.761202385484262,0.765167265622459,0.769103337645580,
0.773010453362737,0.776888465673233,0.780737228572094,0.784556597155575,
0.788346427626606,0.792106577300212,0.795836904608883,0.799537269107905,
0.803207531480645,0.806847553543799,0.810457198252595,0.814036329705948,
0.817584813151584,0.821102514991105,0.824589302785025,0.828045045257755,
0.831469612302545,0.834862874986380,0.838224705554838,0.841554977436898,
0.844853565249707,0.848120344803297,0.851355193105265,0.854557988365401,
0.857728610000272,0.860866938637767,0.863972856121586,0.867046245515693,
0.870086991108711,0.873094978418290,0.876070094195407,0.879012226428633,
0.881921264348355,0.884797098430938,0.887639620402854,0.890448723244758,
0.893224301195515,0.895966249756185,0.898674465693954,0.901348847046022,
0.903989293123443,0.906595704514915,0.909167983090522,0.911706032005430,
0.914209755703531,0.916679059921043,0.919113851690058,0.921514039342042,
0.923879532511287,0.926210242138311,0.928506080473216,0.930766961078984,
0.932992798834739,0.935183509938948,0.937339011912575,0.939459223602190,
0.941544065183021,0.943593458161960,0.945607325380521,0.947585591017741,
0.949528180593037,0.951435020969008,0.953306040354194,0.955141168305771,
0.956940335732209,0.958703474895872,0.960430519415566,0.962121404269042,
0.963776065795440,0.965394441697689,0.966976471044852,0.968522094274417,
0.970031253194544,0.971503890986252,0.972939952205560,0.974339382785576,
0.975702130038529,0.977028142657754,0.978317370719628,0.979569765685441,
0.980785280403230,0.981963869109555,0.983105487431216,0.984210092386929,
0.985277642388941,0.986308097244599,0.987301418157858,0.988257567730750,
0.989176509964781,0.990058210262297,0.990902635427780,0.991709753669099,
0.992479534598710,0.993211949234795,0.993906970002356,0.994564570734255,
0.995184726672197,0.995767414467660,0.996312612182778,0.996820299291166,
0.997290456678690,0.997723066644192,0.998118112900149,0.998475580573295,
0.998795456205172,0.999077727752645,0.999322384588349,0.999529417501093,
0.999698818696204,0.999830581795823,0.999924701839145,0.999981175282601
};
```

File: curvelib.c

```
/*-----------------------------------------------------------------------
**      File:           curvelib.c
**      Intent:         Curve arithmetic functions.
**      Routines:       void    chain_info( int *, int, double *, COORD *)
**                      void    position( double *, double, int * )
*/

#include "..\include\humanf.h"
#include "..\global\global.h"


/*-----------------------------------------------------------------------
**      Function:       chain_info
**      Intent:         Get the run-length and the pixel locations of the
**                      curve from its chain code.
**      Arguments:      chain           the input chain code.
**                      npoints         number of pixels in the curve.
**                      runlen          the run-length of the curve.
**                      location        the position of each pixel in the
**                                      curve.
*/
void
chain_info( chain, npoints, runlen, location )
int     *chain;
int     npoints;
double  *runlen;
COORD   *location;
{
        int     curr_x, curr_y;         /* curve pixel position */
        double  curr_d;                 /* curve length at current point */
        int     code;
        int     i;

        location->x = curr_x = chain[0];
        location->y = curr_y = chain[1];
        location++;
        *runlen++ = curr_d = 0.;

        chain += 2;                     /* initialize to the beginning */

        for ( i = 0; i < npoints-1; i++ ) {
                code = *chain++;
                curr_x += neighbor[code].x;
                curr_y += neighbor[code].y;
                curr_d += neighbor[code].len;
                location->x = curr_x;
                location->y = curr_y;
                location++;
                *runlen++ = curr_d;
        }
}


/*-----------------------------------------------------------------------
**      Function:       position
**      Intent:         Return the index in an array of run-length. The index
**                      represents the closest position of the given curve
**                      length.
**      Arguments:      runlen          the array of run-length.
**                      clen            the curve length.
**                      index           the index in the array.
*/
```

```
void
position( runlen, clen, index )
double  *runlen;
double  clen;
int     *index;
{
        int     i = 0;

        while ( runlen[i] < clen )
                i++;

        /*
        **      Determine which point is closer the desired curve length.
        */
        if ( runlen[i] - clen < (runlen[i]-runlen[i-1])/2. )
                i -= 1;

        *index = i;
}
```

File: curvelib.h

```
/*----------------------------------------------------------------------------
**      File:           curvelib.h
**      Intent:         Curve arithmetic function declarations.
*/

#ifndef _CURVELIB_H
#define _CURVELIB_H

#define ROOT2   1.4142136               /* value of root 2 */

/*
**      Directions defined by neighbor[] in global.c, which defines the
**      neighboring pixel positions and their distances. They are also used
**      as chain code in a clockwise manner.
**
**      0 1 2
**      7 X 3           X is the current pixel.
**      6 5 4
*/
#define NWEST   0
#define NORTH   1
#define NEAST   2
#define EAST    3
#define SEAST   4
#define SOUTH   5
#define SWEST   6
#define WEST    7

/*
**      Function declarations.
*/
extern  void    chain_info( int *, int, double *, COORD * );
extern  void    position( double *, double, int * );

#endif
```

File: database.c

```
/*----------------------------------------------------------------------------
**      File:           database.c
**      Intent:         Database manipulation functions.
**      Routines:       void    create_database( DATABASE *, int )
**                      int     load_database( DATABASE *, char * )
**                      void    save_database( DATABASE, char * )
**                      void    add_vector( DATABASE *, double *, char * )
**                      int     blend_vector( DATABASE *, double *, int )
**                      int     find_vector_id( DATABASE, char * )
*/

#include "..\include\humanf.h"

/*----------------------------------------------------------------------------
**      Function:       create_database
**      Intent:         To create a new database.
**      Arguments:      database        pointer to the database in memory.
**                      vsize           vector size of the new database.
*/
void
create_database( database, vsize )
DATABASE        *database;
int             vsize;
{
        database->nvectors = 0;
        database->vsize = vsize;
}


/*----------------------------------------------------------------------------
**      Function:       load_database
**      Intent:         To load a face profile database from a file into
**                      the memory.
**      Arguments:      database        pointer to the database in memory.
**                      infile          name of the input file.
**      Return value:   0               unable to open input file.
**                      1               operation successful.
*/
int
load_database( database, infile )
DATABASE        *database;
char            *infile;
{
        static void     load_vector( FILE *, DATABASE *, int );

        FILE    *fp;
        int     nv;
        int     i;

        if ( (fp = fopen( infile, "r" )) == NULL )
                return( 0 );            /* unable to open input file */

        fscanf( fp, "%d\n", &database->nvectors ); /* #vectors in the file */

        fscanf( fp, "%d\n", &database->vsize ); /* vector size */

        database->vector =
                (double **)malloc( sizeof(double *)*database->nvectors );
        database->vector_id =
                (char **)malloc( sizeof(char *)*database->nvectors );
        database->vector_ctr =
                (int *)malloc( sizeof(int)*database->nvectors );

        for ( i = 0; i < database->nvectors; i++ )
                load_vector( fp, database, i );

        fclose( fp );

        return( 1 );
}


/*----------------------------------------------------------------------------
**      Function:       load_vector
**      Intent:         To load a vector from a currently opened database file
**                      into the memory.
**      Arguments:      fp              file pointer
**                      database        pointer to the database in memory.
**                      vnum            vector number
*/
static void
load_vector( fp, database, vnum )
FILE            *fp;
DATABASE        *database;
int             vnum;
{
        char    buf[80];
        int     len;
        int     i;

        fgets( buf, 80, fp );           /* get the vector id */
        len = strlen( buf );            /* string length included the newline */
        buf[len-1] = '\0';              /* replace newline with a null */

        database->vector_id[vnum] = (char *)malloc( sizeof(char)*len );
        strcpy( database->vector_id[vnum], buf );

        fscanf( fp, "%d\n", &database->vector_ctr[vnum] );

        database->vector[vnum] =
                (double *)malloc( sizeof(double)*database->vsize );

        for ( i = 0; i < database->vsize; i++ )
                fscanf( fp, "%lf\n", &database->vector[vnum][i] );
}


/*----------------------------------------------------------------------------
**      Function:       save_database
**      Intent:         To save a face profile database from memory into
**                      a file.
**      Arguments:      database        database in memory.
**                      outfile         name of the output file.
*/
void
save_database( database, outfile )
DATABASE        database;
char            *outfile;
{
        FILE    *fp;
        int     i, j;

        fp = fopen( outfile, "w" );

        fprintf( fp, "%d\n", database.nvectors );
        fprintf( fp, "%d\n", database.vsize );

        for ( i = 0; i < database.nvectors; i++ ) {
                fprintf( fp, "%s\n", database.vector_id[i] );
                fprintf( fp, "%d\n", database.vector_ctr[i] );

                for ( j = 0; j < database.vsize; j++ )
                        fprintf( fp, "%-8.6f\n", database.vector[i][j] );
        }

        fclose( fp );
}


/*----------------------------------------------------------------------------
**      Function:       add_vector
**      Intent:         To add a new vector into the database.
**      Arguments:      database        pointer to the database in memory.
**                      vector          the vector to be added.
**                      vid             vector id.
*/
void
add_vector( database, vector, vid )
DATABASE        *database;
double          *vector;
char            *vid;
{
        double  **vector_array;
        char    **vector_id_array;
        int     *vector_ctr_array;
        int     nv;
        int     i;

        nv = ++database->nvectors;      /* increment the number of vectors */

        /*
        **      Save the array pointers before allocating a new one.
        */
        vector_array = database->vector;
        vector_id_array = database->vector_id;
        vector_ctr_array = database->vector_ctr;

        /*
        **      Allocating space for the new arrays of pointers.
        */
        database->vector = (double **)malloc( sizeof(double *)*nv );
        database->vector_id = (char **)malloc( sizeof(char *)*nv );
        database->vector_ctr = (int *)malloc( sizeof(int)*nv );

        /*
        **      Transferring the data from the arrays to the new arrays.
        */
```

```
	for ( i = 0; i < nv-1; i++ ) {
		database->vector[i] = vector_array[i];
		database->vector_id[i] = vector_id_array[i];
		database->vector_ctr[i] = vector_ctr_array[i];
	}

	/*
	**	Allocating space for the new vector and storing it.
	*/
	database->vector[nv-1] =
		(double *)malloc( sizeof(double)*database->vsize );
	for ( i = 0; i < database->vsize; i++ )
		database->vector[database->nvectors-1][i] = vector[i];

	/*
	**	Allocating space for the new vid and storing it.
	*/
	database->vector_id[nv-1] =
		(char *)malloc( sizeof(char)*(strlen(vid)+1) );
	strcpy( database->vector_id[nv-1], vid );

	/*
	**	Setting new vector counter to 1.
	*/
	database->vector_ctr[nv-1] = 1;

	/*
	**	Free memory space from old arrays.
	*/
	free( vector_array );
	free( vector_id_array );
	free( vector_ctr_array );
}


/*-------------------------------------------------------------------------
**	Function:	blend_vector
**	Intent:		To blend a vector into the existing one in the database.
**			Blending here means averaging.
**	Arguments:	database	pointer to the database in memory.
**			vector_in	the new vector to be blended.
**			vnum		vector number of the existing vector.
**	Return value:	0		invalid vector number.
**			1		operation successful.
*/
int
blend_vector( database, vector_in, vnum )
DATABASE	*database;
double		*vector_in;
int		vnum;
{
	int	i;
	int	vector_ctr;
	double	*vector;

	if ( vnum >= database->nvectors )
		return( 0 );		/* invalid vector number */

	vector_ctr = database->vector_ctr[vnum];
	vector = database->vector[vnum];

	for ( i = 0; i < database->vsize; i++ )
		vector[i] = (vector[i]*(double)vector_ctr + vector_in[i]) /
				(double)(vector_ctr+1);

	database->vector_ctr[vnum]++;

	return( 1 );
}


/*-------------------------------------------------------------------------
**	Function:	find_vector_id
**	Intent:		To find the vector given a vector id.
**	Arguments:	database	pointer to the database in memory.
**			vector_id	the vector id.
**	Return value:	>= 0		the vector number.
**			-1		vector not found.
*/
int
find_vector_id( database, vector_id )
DATABASE	database;
char		*vector_id;
{
	int	i;

	for ( i = 0; i < database.nvectors; i++ )
		if ( !strcmp( database.vector_id[i], vector_id ) )
			return i;

	return -1;
}
```

File: database.h

```
/*-------------------------------------------------------------------------
**	File:		database.h
**	Intent:		Database manipulation function declarations.
*/

#ifndef _DATABASE_H
#define _DATABASE_H

#include "..\include\humanf.h"

extern	void	create_database( DATABASE *, int );
extern	int	load_database( DATABASE *, char * );
extern	void	save_database( DATABASE, char * );
extern	void	add_vector( DATABASE *, double *, char * );
extern	int	blend_vector( DATABASE *, double *, int );
extern	int	find_vector_id( DATABASE, char * );

#endif
```

File: extract.c

```
/*-------------------------------------------------------------------------
**	File:		extract.c
**	Intent:		Curve extraction from binary images. Chain code
**			is used for curve representation.
**	Routines:	int	where_is_boundary( int, int )
**			void	get_chain_code( COORD, int, int, int, double,
**					int *, int, int *, COORD *, double *)
*/

#include "..\include\humanf.h"
#include "..\fft\complex.h"
#include "..\global\global.h"

/*
**	Chain code generation traverse directions.
*/
#define CW	0				/* clockwise */
#define CCW	1				/* counter-clockwise */

/*
**	The outer pixel relative positions of the template used in
**	the check_pixel() function.
*/
static COORD	circle16[] = {
	{ -2, -2 },				/* north-west */
	{ -1, -3 },
	{ 0, -3 },				/* north */
	{ 1, -3 },
	{ 2, -2 },				/* north-east */
	{ 3, -1 },
	{ 3, 0 },				/* east */
	{ 3, 1 },
	{ 2, 2 },				/* south-east */
	{ 1, 3 },
	{ 0, 3 },				/* south */
	{ -1, 3 },
	{ -2, 2 },				/* south-west */
	{ -3, 1 },
	{ -3, 0 },				/* west */
	{ -3, -1 }
};


/*-------------------------------------------------------------------------
**	Function:	where_is_boundary
**	Purpose:	Given the initial position in the image plane
**			of a binary image, search horizontally for a boundary
**			pixel. A boundary pixel is a pixel belongs to the set
**			of pixels of solid face profile. Its value should be 0.
**	Note:		This procedure assumes that the person is facing
**			towards x = 0.
**	Arguments:	init_x, init_y		initial position.
**	Return value:	the x-coordinate of the boundary pixel found.
*/
int
where_is_boundary( init_x, init_y )
int	init_x, init_y;
{
	int	initial_value;
	int	x = init_x;
	int	i;

	initial_value = READ_PIXEL( init_x, init_y );

	if ( initial_value == 0 ) {	/* initially within the face profile */
		for ( i = 1; i < init_x; i++ )
			if ( READ_PIXEL( init_x-i, init_y ) )
				return( init_x-i+1 );
	}
	else {			/* initially outside the face profile */
		for ( i = 1; i < init_x; i++ )
			if ( READ_PIXEL( init_x+i, init_y ) == 0 )
				return( init_x+i );
	}
}


/*-------------------------------------------------------------------------
**	Function:	get_chain_code
**	Purpose:	To extract the face profile curve from a binary
**			image of a human face. The difference code of the
**			chain code generated contains only -1, 0, and 1.
**	Arguments:	init_pos	the starting pixel. It should be a
**					pixel within the face profile that
**					is close to the boundary.
**			normal		the normal direction of pixel given
**					in init_pos. It does not have to be
**					exact, but it should be close to the
**					reality.
**			dir		traverse direction. It is either
**					CW for clockwise or CCW for
**					counter-clockwise.
**			y_limit		terminate the chain when the curve
**					reaches this y value.
**			distance	terminate the chain when the distance
**					between the starting position and the
**					current position reaches this value.
**			stop_pos	terminate the chain when the current
**					position matches this position.
**			chain		the chain code.
**			max_size	maximum size of the chain code.
**			npoints		number of pixels in the curve.
**			end_pos		the last pixel coordinates.
**			clen		the total curve length.
*/
void
get_chain_code( init_pos, normal, dir, y_limit, distance, stop_pos,
		chain, max_size, npoints, end_pos, clen )
COORD	init_pos;
int	normal;
int	dir;
int	y_limit;
double	distance;
COORD	stop_pos;
int	*chain;
```

```
int     max_size;
int     *npoints;
COORD   *end_pos;
double  *clen;
{
        static int      check_pixel( int, int, int );

        int     curr_x, curr_y;
        int     next_pos;               /* next neighbor position to check */
        int     next_x, next_y;         /* coordinates of the next position */
        int     next_normal;            /* normal of the next pixel */
        int     np;                     /* array pointer for the chain code */
        double  len = 0.0;              /* initial curve length */
        double  dist = 0.0;             /* distance between two pixels */
        int     i;

        chain[0] = curr_x = init_pos.x; /* initial coordinates in chain code */
        chain[1] = curr_y = init_pos.y; /* .. */
        np = 2;                 /* reset array pointer to beginning of data */

        while ( 1 ) {
                for ( i = 1; i <= 3; i++ ) {
                        next_pos = ( (dir==CW) ? MOD8(normal+i) :
                                        MOD8(normal-i) );
                        next_x = curr_x + neighbor[next_pos].x;
                        next_y = curr_y + neighbor[next_pos].y;
                        next_normal = ( (dir==CW) ? MOD8(next_pos-2) :
                                        MOD8(next_pos+2) );

                        if ( check_pixel( next_x, next_y, next_normal ) ||
                            i == 3 ) {
                                chain[np++] = next_pos;
                                curr_x = next_x;
                                curr_y = next_y;
                                normal = next_normal;
                                len += neighbor[next_normal].len;

                                /*
                                **      Compute the distance only if the curve
                                **      length is longer than the given value.
                                **      Note that a curve line is always
                                **      longer than a straight line.
                                */
                                if ( len > distance )
                                        dist = DISTANCE( init_pos.x, init_pos.y,
                                                        curr_x, curr_y );
                                break;
                        }
                }

                /*
                **      Check all the terminating conditions.
                */
                if ( curr_y == y_limit )
                        break;
                else if ( dist >= distance )
                        break;
                else if ( curr_x == stop_pos.x && curr_y == stop_pos.y )
                        break;
                else if ( np >= max_size-1 )
                        break;
        }

        chain[np] = 255;        /* chain termination */
        *npoints = np - 1;      /* -2 (start_x, start_y) + 1 ( starting pt ) */

        end_pos->x = curr_x;    /* last pixel coordinates */
        end_pos->y = curr_y;    /* .. */

        *clen = len;            /* total curve length */
}


/*---------------------------------------------------------------------------
**      Function:       check_pixel
**      Purpose:        Using a technique similar to a morphological image
**                      processing basic operation. It uses a template of
**                      16 outer pixels to smooth out the face profile.
**      Arguments:      x, y    the location of the pixel to be checked.
**                      normal  the normal direction of the pixel.
**      Return value:   0       not a valid pixel.
**                      1       valid pixel.
*/
static int
check_pixel( x, y, normal )
int     x, y;
int     normal;
{
        int     p;
        int     i;

        x -= circle16[normal*2].x;
        y -= circle16[normal*2].y;

        p = MOD8( normal+2 )*2;
        for ( i = 0; i < 16; i++ ) {
                if ( READ_PIXEL( x+circle16[p].x, y+circle16[p].y ) != 0 )
                        return 0;

                p = MOD16( p-1 );
        }

        return 1;
}
```

File: extract.h

```
/*---------------------------------------------------------------------------
**      File:           extract.h
**      Intent:         Curve extraction function declarations.
*/

#ifndef _EXTRACT_H
#define _EXTRACT_H

/*
**      Chain code generation traverse directions.
*/
#define CW      0                       /* clockwise */
#define CCW     1                       /* counter-clockwise */

extern  int     where_is_boundary( int, int );
extern  void    get_chain_code( COORD, int, int, int, double, COORD,
                                int *, int, int *, COORD *, double * );

#endif
```

File: feature.c

```
/*---------------------------------------------------------------------------
**      File:           feature.c
**      Intent:         Detect and locate feature points of a face profile
**                      from chain code. The person is assumed to be
**                      facing left in the image.
**      Routines:       void    nose_tip( COORD * )
**                      void    nose_bottom( COORD, COORD * )
**                      void    nose_top( COORD, COORD, COORD * )
**                      void    upper_termination( COORD, COORD, COORD * )
**                      void    chin_position( COORD, COORD, COORD * )
**                      void    lower_termination( COORD, COORD, COORD,
**                                              COORD * )
*/

#include "..\include\humanf.h"
#include "..\vincelib\filters.h"
#include "..\curvelib\curvelib.h"
#include "..\global\global.h"
#include "..\extract\extract.h"

#define NOSE_WIN_TOP            160     /* 1/3 down from the top */
#define NOSE_WIN_BOTTOM         360     /* 1/4 up from the bottom */
#define RATIO_A                 0.75    /* ratio for the upper termination */
#define RATIO_B                 0.5     /* ratio for the lower termination */

static void     mark_feature( int *, int, int *, int * );


/*---------------------------------------------------------------------------
**      Function:       nose_tip
**      Intent:         Find the nose tip position within a window area.
**                      The nose tip is found by scanning a vertical line
**                      across the window.
**      Arguments:      pos             nose tip position.
*/
void
nose_tip( pos )
COORD   *pos;
{
        int     x, y, y2;
        int     curr_win_top = NOSE_WIN_TOP;
        int     not_yet;                /* not yet reach the nose tip */

        x = where_is_boundary( IMAGE_DX/2, (NOSE_WIN_TOP+NOSE_WIN_BOTTOM)/2 );

        while ( x >= 0 ) {
                not_yet = 0;

                /*
                **      Adjust window top position so that the vertical
                **      scan starts from a background area. The lowest window
                **      top position is the center of the image.
                */
                if ( READ_PIXEL( x, curr_win_top ) == 0 )
                        curr_win_top = IMAGE_DY/2;

                /*
                **      Look for the face silhouette.
                */
                for ( y = curr_win_top; y <= NOSE_WIN_BOTTOM; y++ ) {
                        if ( READ_PIXEL( x, y ) == 0 ) {
                                not_yet = 1;
                                break;
                        }
                }

                if ( not_yet )
                        x--;
                else
                        break;          /* just pass the nose tip */
        }

        x++;                            /* move back to the nose tip */

        /*
        **      Find out the center position of the nose tip.
        */
        y = NOSE_WIN_BOTTOM;
        while( READ_PIXEL( x, y ) != 0 ) {
                y--;
        }

        y2 = y;
        while ( READ_PIXEL( x, y2 ) == 0 ) {
                y2--;
        }
        y2++;

        pos->x = x;
        pos->y = (y+y2)/2;
}


/*---------------------------------------------------------------------------
**      Function:       nose_bottom
**      Intent:         Find the nose bottom position given the nose tip
**                      position. The nose bottom is defined as the first
**                      positive curvature change.
**      Arguments:      start_pos       nose tip position.
**                      pos             nose bottom position.
*/
void
nose_bottom( start_pos, pos )
COORD   start_pos;
COORD   *pos;
{
        COORD   out_pos;                /* a position outside of the image */
        int     *chain;                 /* chain code */
        int     npoints;                /* number of pixels in the chain */
```

```
	COORD	end_pos;			/* last pixel position in tha chaln */
	double	clen;				/* curve length of the chaln */
	int	1;

	int	*data;				/* choin codo w/o the starting coord */
	double	*runlen;			/* run-length of the curve */
	COORD	*location;			/* coordinates of eoch pixel */
	lnt	fpoint(8(;			/* feoture points' indexes in data(( */
	int	nfp;				/* number of feature polnts found */

	out_pos.x = IMAGE_DX;
	out_pos.y = IMAGE_DY;

	chaln = (lnt *)malloc( slzeof(lnt)*256 );

	get_chaln_code( start_pos, WEST, CCW, start_pos.y+IMAGE_DY/8,
			9999., out_pos, choin, 256, &npolnts, &end_pos, &clen );

	dato = (int *)molloc( sizeof(int)*npoints );
	runlen = (double *)molloc( sizeof(double)*npoints );
	location = (COORD *)malloc( sizeof(COORD)*npoints );

	/*
	**	Prepore data for processing. data() contalns all tha volues
	**	ln chaln(( without the starting coordinates. data(D( ls
	**	made equal to the first direction code ln the input chaln code.
	*/
	data(D( = choin(2(;
	icopy( chain+2, data+1, npoints-1 );

	/*
	**	Get the run-lengths and the locations of each pixel on
	**	the curve.
	*/
	chain_info( chain, npoints, runlen, locatlon );

	/*
	**	Apply a series of filters to locate the feature points
	**	on the curve.
	*/
	incr( doto, npoints );
	sum( data, npolnts, 7 );
	sum( data, npolnts, 11 );
	median( data, npoints, 11 );
	clipplng( data, npoints, 3 );

	mark_feoture( doto, npoints, fpoint, &nfp );

	/*
	**	The first positive turn is considered the nose bottom.
	*/
	for ( i = 0; i < nfp; 1++ )
		if ( data(fpoint(i(( > 0 ) (
			pos->x = locatlon(fpoint(i((.x;
			pos->y = location(fpoint(i((.y;
			break;
		)

	free( choin );
	free( dato );
	free( runlen );
	free( location );
)


/*-----------------------------------------------------------------------
**	Function:	nose_top
**	Intent:		Find the nose top position given the nose tip and the
**			nose bottom positions. The nose top is defined as
**			o pixel on the profile of the nose. When o line is
**			drawn between that position and the nose bottom, the
**			immediate extension of the line from that position
**			must be within the face.
**	Arguments:	nose_tip_pos	nose tip position.
**			nose_bottom_pos nose bottom position.
**			pos		nose top position.
*/
void
nose_top( nose_tip_pos, nose_bottom_pos, pos )
COORD	nose_tip_pos, nose_bottom_pos;
COORD	*pos;
(
	COORD	out_pos;			/* o position outside of the imoge */
	int	*chain;				/* chain code */
	int	npoints;			/* number of pixels in the choin */
	COORD	end_pos;			/* last pixel position in the chain */
	double	clen;				/* curve length of the chain */
	double	dist;				/* distance from the nose bottom */
	double	d;				/* distance away from the nose */
	int	x, y;
	int	i;
	int	done;

	double	*runlen;			/* run-length of the curve */
	COORD	*location;			/* coordinates of each pixel */

	out_pos.x = IMAGE_DX;
	out_pos.y = IMAGE_DY;

	chain = (int *)malloc( sizeof(int)*512 );

	get_chain_code( nose_tip_pos, SWEST, CW, NOSE_WIN_TOP,
			9999., out_pos, chain, 512, &npoints, &end_pos, &clen );

	runlen = (double *)malloc( sizeof(double)*npoints (;
	location = (COORD *)malloc( sizeof(COORD)*npoints );

	/*
	**	Get the run-length and the location of each pixel on
	**	the curve.
	*/
	chain_info( chain, npoints, runlen, location );

	/*
	**	For each pixel on the curve, test to see if it reaches the
	**	nose top position.
	*/
	for ( i = 0; i < npoints; i++ ) (
		dlst = DISTANCE( nose_bottom_pos.x, nose_bottom_pos.y,
				location(i(.x, location(i(.y );

		/*
		**	Check the pixels away from the nose boundory.
		**	If they are not white, thot means the nose top position
		**	has been reached.
		*/
		done = 1;

		for ( d = 1.0; d <= 10.0; d += 1.0 ) (
			x = ROUND( (double)location(i).x -
			    (double)(nose_bottom_pos.x-locotion(i(.x)*d/dist );
			y = ROUND( (double)location(i(.y -
			    (double)(nose_bottom_pos.y-locotion(i(.y)*d/dist );

			lf ( READ_PIXEL( x, y ) != 0 ) (
				done = 0;
				breok;
			(
		(

		if ( done == 1 ) (
			pos->x = location(1(.x;
			pos->y = locotion(i).y;
			break;
		)
	)

	free( chain );
	free( runlen );
	free( location );
)


/*-----------------------------------------------------------------------
**	Function:	upper_termination
**	lntent:		Find the upper terminoting position given the nose top
**			and the nose bottom positions. A reference length is
**			computed from the distance between the two given
**			positions. The upper terminoting position is defined
**			as the pixel on the curve from which the distance to
**			the nose top bottom is proportional to the
**			reference length.
**	Arguments:	nose_top_pos	nose topposltion.
**			nose_bottom_pos nose bottom posltion.
**			pos		upper terminating position.
*/
void
uppet_termination( nose_top_pos, nose_bottom_pos, pos )
COORD	nose_top_pos;
COORD	nose_bottom_pos;
COORD	*pos;
(
	double	dist;				/* distanee \ nose top ond bottom */
	lnt	*choin;				/* choin code */
	COORD	out_pos;			/* a position outside of the image */
	COORD	end_pos;			/* last pixel posltion in the ehain */
	int	npoints;			/* number of pixels ln the chain */
	double	clen;				/* curve length of the choin */

	out_pos.x = IMAGE_DX;
	out_pos.y = IMAGE_DY;

	choin = (int *)molloc( sizeof(int)*256 );

	dist = DISTANCE( nose_top_pos.x, nose_top_pos.y,
			nose_bottom_pos.x, nose_bottom_pos.y );

	get_cheln_code( nose_top_pos, NWEST, CW, 0, dist*RATIO_A,
			out_pos, chain, 256, &npolnts, &end_pos, &clen );

	pos->x = end_pos.x;
	pos->y = end_pos.y;

	free( chain );
)


/*-----------------------------------------------------------------------
**	Function:	chin_position
**	Intent:		Flnd the chin positlon glven the nose top and the nose
**			bottom positions. The chln position ls found by the
**			curvature analysis of the curve below the nose bottom.
**			The first feature point with a negotive curvature
**			and the distance from which to the nose bottom
**			is greoter than the distance between the nose top
**			and the nose bottom is considered the chin position.
**	Arguments:	nose_top_pos	nose tip position.
**			nose_bottom_pos nose bottom position.
**			pos		chin position.
*/
void
chln_position( nose_top_pos, nose_bottom_pos, pos )
COORD	nose_top_pos:
COORD	nose_bottom_pos;
COORD	*pos;
(
	COORD	out_pos;			/* o position outside of the image */
	int	*chain;				/* choin code */
	int	npoints;			/* number of pixels in the chain */
	COORD	end_pos;			/* last pixel position in the chain */
	double	clen:				/* curve length of the chain */
	doubla	dist;				/* dist \ current pos and nose bottom */
	double	ref_len;			/* distance \ nose top and bottom */
	int	i;

	lnt	*data;				/* chain code w/o the starting coord */
	double	*runlen;			/* run-length of the curve */
	COORD	*location;			/* coordinates of each pixel */
	int	fpoint(8(;			/* feature points indexes in data(( */
	int	nfp;				/* number of feature points found */

	out_pos.x = IMAGE_DX:
	out_pos.y = IMAGE_DY;

	ref_len = DISTANCE( nose_top_pos.x, nose_top_pos.y,
			nose_bottom_pos.x, nose_bottom_pos.y );

	chain = (int *)malloc( slzeof(int)*512 );

	get_chain_code( nose_bottom_pos, SWEST, CCW, IMAGE_DY-1, 2.*ref_len,
			out_pos, chain, 512, &npoints, &end_pos, &clen );

	data = (int *)malloc( sizeol(int)*npoints );
	runlen = (double *)malloc( sizeof(double)*npoints );
	location = (COORD *)malloc( sizeof(COORD)*npoints );

	/*
	**	Prepare data for processing. doto(( contains oll the values
	**	in chain() without the starting coordinates. dota(D( is
```

```
**      made equal to the first direction code in the input chain code.
*/
data[0] = chain[2];
icopy( chain+2, data+1, npoints-1 );

/*
**      Get the run-lengths and the locations of each pixel on
**      the curve.
*/
chain_info( chain, npoints, runlen, location );

/*
**      Apply a series of filters to locate the feature points
**      on the curve.
*/
incr( data, npoints );
sum( data, npoints, 11 );
sum( data, npoints, 31 );
median( data, npoints, 11 );
clipping( data, npoints, 10 );

mark_feature( data, npoints, fpoint, &nfp );

/*
**      The first positive turn is considered the nose bottom.
*/
for ( i = 0; i < nfp; i++ ) {
        dist = DISTANCE( nose_bottom_pos.x, nose_bottom_pos.y,
                         location[fpoint[i]].x, location[fpoint[i]].y );

        if ( data[fpoint[i]] < 0 && dist > ref_len ) {
                pos->x = location[fpoint[i]].x;
                pos->y = location[fpoint[i]].y;
                break;
        }
}

free( chain );
free( data );
free( runlen );
free( location );
}


/*-----------------------------------------------------------------------
**      Function:       lower_termination
**      Intent:         Find the upper terminating position given the nose top
**                      and the nose bottom positions. A reference length is
**                      computed from the distance between the two given
**                      positions. The upper terminating position is defined
**                      as the pixel on the curve from which the distance to
**                      the nose top bottom is proportional to the
**                      reference length.
**      Arguments:      nose_top_pos    nose top position.
**                      nose_bottom_pos nose bottom position.
**                      pos             upper terminating position.
*/
void
lower_termination( nose_top_pos, nose_bottom_pos, chin_pos, pos )
COORD   nose_top_pos;
COORD   nose_bottom_pos;
COORD   chin_pos;
COORD   *pos;
{
        double  dist;                   /* distance \ nose top and bottom */
        int     *chain;                 /* chain code */
        COORD   out_pos;                /* a position outside of the image */
        COORD   end_pos;                /* last pixel position in the chain */
        int     npoints;                /* number of pixels in the chain */
        double  clen;                   /* curve length of the chain */

        out_pos.x = IMAGE_DX;
        out_pos.y = IMAGE_DY;

        chain = (int *)malloc( sizeof(int)*256 );

        dist = DISTANCE( nose_top_pos.x, nose_top_pos.y,
                         nose_bottom_pos.x, nose_bottom_pos.y );

        get_chain_code( chin_pos, SWEST, CCW, IMAGE_DY-1, dist*RATIO_B,
                        out_pos, chain, 256, &npoints, &end_pos, &clen );

        pos->x = end_pos.x;
        pos->y = end_pos.y;

        free( chain );
}


/*-----------------------------------------------------------------------
**      Function:       mark_feature
**      Intent:         Mark all the feature points in the data based on the
**                      locations of maxima and minima.
**      Arguments:      data            data after the filters.
**                      n               number of points in data.
**                      fpoint          indexes of the feature points.
**                      nfpoints        number of feature points extracted.
*/
static void
mark_feature( data, n, fpoint, nfpoints )
int     *data;
int     n;
int     *fpoint;
int     *nfpoints;
{
        int     sign = 0;
        int     max;                    /* maximum magnitude in the region */
        int     max_occ;                /* #occurrences of the max */
        int     mul;
        int     i;
        int     nfp = 0;
        int     found = 1;              /* initially set to 1 */

        for ( i = 1; i < n; i++ ) {
                mul = data[i]*data[i-1];

                if ( !found ) {
                        if ( data[i] == max )   /* increment #occurrences */
                                max_occ++;
                        else if ( sign == 1 && data[i] > max || /* +ve reg */
                                  sign == -1 && data[i] < max ) { /* -ve reg */
                                max = data[i];  /* record new max */
                                max_occ = 1;    /* reset #occurrences counter */
                        }
                        else if ( sign == 1 && data[i] < max || /* leave +max */
                                  sign == -1 && data[i] > max ) { /* -max */
                                fpoint[nfp] = i - max_occ/2 - 1;
                                nfp++;
                                found = 1;
                        }
                }

                /*
                **      It is possible the max is found above.
                **      Therefore this is not an else-block.
                */
                if ( found ) {
                        if ( sign == 0 && mul > 0 ) {   /* enter region */
                                if ( data[i] > 0 )
                                        sign = 1;       /* +ve */
                                else
                                        sign = -1;      /* -ve */

                                max = data[i];  /* record 1st max */
                                max_occ = 1;
                                found = 0;
                        }
                        else if ( mul < 0 ) {   /* exit region and enter */
                                                /* next region w/o passing 0 */
                                sign = -sign;
                                max = data[i];          /* record 1st max */
                                max_occ = 1;
                                found = 0;
                        }
                        else if ( mul == 0 )            /* exit region */
                                sign = 0;
                }
        }

        *nfpoints = nfp;
}
```

File: feature.h

```
/*-----------------------------------------------------------------------
**      File:           feature.h
**      Intent:         Feature points detection function declarations.
*/

#include "..\include\humanf.h"

#ifndef _EXTRACT_H
#define _EXTRACT_H

/*
**      Function declarations.
*/
extern  void    nose_tip( COORD * );
extern  void    nose_bottom( COORD, COORD * );
extern  void    nose_top( COORD, COORD, COORD * );
extern  void    upper_termination( COORD, COORD, COORD * );
extern  void    chin_position( COORD, COORD, COORD * );
extern  void    lower_termination( COORD, COORD, COORD, COORD * );

#endif
```

File: fft.c

```
/*-----------------------------------------------------------------------
**      File:           fft.c
**      Intent:         Fast Fourier transform.
**      Routines:       void    fft( COMPLEX *, int )
**                      void    ifft( COMPLEX *, int )
**                      void    lowpass( COMPLEX *, int, int )
**                      void    magnitude( COMPLEX *, double *, int )
*/

#include "..\include\humanf.h"
#include "fft.h"
#include "complex.h"

static  void    fft2( COMPLEX *, int, int );
static  void    bit_reverse( COMPLEX *, int );
extern  double  cos_table[];

/*-----------------------------------------------------------------------
**      Function:       fft
**      Intent:         Forward FFT. N is the size of the array.
**      Arguments:      x       input data in complex values.
**                      N       size of the input array.
*/
void
fft( x, N )
COMPLEX *x;
int     N;
{
        fft2( x, N, FALSE );
}


/*-----------------------------------------------------------------------
**      Function:       ifft
**      Intent:         Inverse FFT. N is the size of the array.
**      Arguments:      x       input data in complex values.
**                      N       size of the input array.
*/
void
ifft( x, N )
COMPLEX *x;
int     N;
{
        fft2( x, N, TRUE );
}


/*-----------------------------------------------------------------------
**      Procedure:      lowpass
**      Intent:         A lowpass filter that cuts out the high frequencies in
**                      a transformed sequence. This procedure makes use of
```

```
**                      the identity
**                                      x[-i] = x[N - i]
**                      which represents the negative frequencies. This lowpass
**                      function keeps the values unchanged in the range
**                      between x[-k] to x[k], inclusively. The rest of the
**                      values are set to 0.
**      Arguments:      x       the transformed sequence.
**                      N       size of the transformed sequence.
**                      k       the cutoff index of the sequence.
**      Return value:   0       range error.
**                      1       operation successful.
*/
int
lowpass( x, N, k )
COMPLEX  *x;
int      N;
int      k;
{
        int     i;

        if ( k < 0 || k > N/2 )
                return 0;

        for ( i = k+1; i <= N/2; i++ )          /* process +ve frequencies */
                COMPLEX( x[i], 0., 0. );

        for ( i = N/2+1; i < N - k; i++ )       /* process -ve frequencies */
                COMPLEX( x[i], 0., 0. );

        return 1;
}


/*------------------------------------------------------------------------
**      Procedure:      fft2
**      Intent:         Compute either the FFT or the inverse FFT of the
**                      complex sequence x[0], ..., x[N-1] in-place, using
**                      the radix 2 decimation-in-time method (cf. Peled & Liu,
**                      "Digital Signal Processing").
**      Arguments:      x       input data in complex values.
**                      N       size of the input array.
**                      inverse a flag indicates the direction of the transform:
**                              zero for forward transformation and
**                              non-zero for inverse transformation.
*/
static void
fft2( x, N, inverse )
COMPLEX *x;
int     N;
int     inverse;
{
        int     j, k;
        int     d;      /* d = 2^(s-1) is the distance (in index) between */
                        /* terms in butterflies at stage s; also equal to */
                        /* the number of distinct twiddle factors at stage s */
        int     s;      /* stage of the computation */
        int     k1;     /* index of first term used with twiddle factor */
        int     kinc;   /* difference in indexes of terms with twiddle */
        int     pinc;   /* difference in powers of twiddle factors */
        int     power;  /* power of twiddle factor */
        int     log2ofN;
        COMPLEX w;      /* twiddle factor */
        COMPLEX r1, r2, ctemp;  /* temporaries used in butterflies */

        bit_reverse( x, N );    /* reorder the sequence using bit reversal */

        d = kinc = j = 1;
        log2ofN = 0;
        while ( j < N ) {
                j <<= 1;
                log2ofN++;
        }

        pinc = N >> 1;

        /*
        **      Proceed through log2ofN stages.
        */
        for ( s = 0; s < log2ofN; s++ ) {
                power = 0;
                k1 = 0;
                kinc <<= 1;

                for ( j = 0; j < d; j++ ) {
                        COMPLEX( w, COS(power, N), SIN(power, N) );
                        if ( inverse )
                                COMPLEX_CONJ( w );

                        for ( k = k1; k < N; k += kinc ) {
                                COMPLEX_MULT( r1, w, x[k+d] );
                                COMPLEX_ASSIGN( r2, x[k] );
                                COMPLEX_ADD( x[k], r2, r1 );
                                COMPLEX_SUB( x[k+d], r2, r1 );
                        }

                        k1++;
                        power += pinc;
                }

                d <<= 1;
                pinc >>= 1;
        }

        if ( inverse )
                for ( k = 0; k < N; k++ ) {
                        COMPLEX( ctemp, 1.0/N, 0.0 )
                        COMPLEX_MULT( x[k], x[k], ctemp );
                }
}


/*------------------------------------------------------------------------
**      Function:       bit_reverse
**      Intent:         First step in the radix 2 decimation-in-time FFT.
**                      The complex sequence x[0], ..., x[N-1] is reordered
**                      in-place using bit reversal: e.g. if N = 8, x[4] =
**                      x[100b] will be interchanged with x[1] == x[001b].
**      Arguments:      x       input data in complex values.
**                      N       size of the input array.
*/
static void
bit_reverse( x, N )
COMPLEX *x;
int     N;
{
        int     i, j, k, n2;
        COMPLEX ctemp;

        j = 0;
        n2 = N >> 1;

        for ( i = 0; i < N - 1; i++ ) {
                if ( j > i ) {
                        /* interchange x[i] and x[j] */
                        COMPLEX_ASSIGN( ctemp, x[i] );
                        COMPLEX_ASSIGN( x[i], x[j] );
                        COMPLEX_ASSIGN( x[j], ctemp );
                }

                k = n2;
                while ( j >= k ) {
                        j -= k;
                        k >>= 1;
                }

                j += k;
        }
}


/*------------------------------------------------------------------------
**      Function:       magnitude
**      Intent:         Compute the magnitude of each element in the input
**                      complex array.
**      Arguments:      x       input data in complex values.
**                      y       magnitudes in double values.
**                      N       size of the input array.
*/
void
magnitude( x, y, N )
COMPLEX *x;
double  *y;
int     N;
{
        int     i;

        for ( i = 0; i < N; i++ )
                COMPLEX_MAG( x[i], y[i] );
}
```

File: fft.h

```
/*------------------------------------------------------------------------
**      File:           fft.h
**      Intent:         FFT function declarations.
*/

#include "complex.h"

#ifndef _FFT_H
#define _FFT_H

#ifndef TRUE
#define TRUE    1
#endif

#ifndef FALSE
#define FALSE   0
#endif

/*
**      Macros to be with the array defined in costable.c.
*/
#define COS(x,n)        cos_table[ ( (int)((1024./n)*x) )%1024 ]
#define SIN(x,n)        cos_table[ ( (int)((1024./n)*x+768.) )%1024 ]

/*
**      Function declarations.
*/
extern  void    fft( COMPLEX *, int );
extern  void    ifft( COMPLEX *, int );
extern  int     lowpass( COMPLEX *, int, int );
extern  void    magnitude( COMPLEX *, double *, int );

#endif
```

File: fgrabber.c

```
/*------------------------------------------------------------------------
**      File:           fgrabber.c
**      Intent:         Contains functions related to the frame grabber board.
**      Routines:       void    init_itexpc( void )
**                      void    freeze( void )
*/

#ifdef  _FRAME_GRABBER

#include "..\include\humanf.h"

#define THRESHOLD       230             /* binarization threshold value */


/*------------------------------------------------------------------------
**      Function:       init_itexpc
**      Intent:         Initial the ITEX PCVISIONplus frame grabber board.
*/
void
init_itexpc( void )
{
        sethdw( 0x300, 0xD0000L, DUAL );
        setdim( 512, 512, 8 );
        initialize();
        select_mem( MEM_A );
        display_mem( MEM_A );
        linlut( INPUT, LINEAR );
        setlut( INPUT, LINEAR );
}
```

```
/*---------------------------------------------------------------------------
**      Function:       freeze
**      Intent:         To snap a binary image when the keyboard is hit.
*/
void
freeze( void )
{
        /*
        **      Set the input look-up table to 255 ( white ) for the input
        **      values equal or above the threshold and linear for the rest.
        */
        linlut( INPUT, 4 );
        contour( INPUT, 4, THRESHOLD, 256-THRESHOLD, 255 );

        /*
        **      Prepare the binary input look-up table for snapping.
        */
        threshold( INPUT, 5, 255, THRESHOLD );

        setlut( INPUT, 4 );     /* use the first table when grabbing */
        grab( 0 );

        getch();                /* waiting for keyboard input */

        setlut( INPUT, 5 );     /* use the second table when snapping */
        snap( WAIT );

        setlut( INPUT, LINEAR );        /* reset back to the linear table */
}

#endif
```

File: fgrabber.h

```
/*---------------------------------------------------------------------------
**      File:           fgrabber.h
**      Intent:         Function declarations related to the frame grabber
**                      board.
*/

#ifndef _FGRABBER_H
#define _FGRABBER_H

#ifdef  _FRAME_GRABBER          /* define only if using the frame grabber */

extern  void    init_itexpc( void );
extern  void    freeze( void );

#endif

#endif
```

File: filters.c

```
/*---------------------------------------------------------------------------
**      File:           filters.c
**      Intent:         Filters for one-dimensional integer arrays.
**      Routines:       void    median( int *, int, int )
**                      void    bsort( int *, int )
**                      void    incr( int *, int )
**                      void    sum( int *, int, int )
**                      void    clipping( int *, int, int )
*/

#include <stdlib.h>


/*---------------------------------------------------------------------------
**      Function:       median
**      Intent:         Median filter of size m.
**      Arguments:      data    the input data array (modify-in-place)
**                      n       number of data points in data
**                      m       filter size (preferrably odd number)
*/
void
median( data, n, m )
int     *data;
int     n;
int     m;
{
        void    bsort( int *, int );
        void    icopy( int *, int *, int );

        int     *win, *t_win;
        int     md2 = m/2;              /* window size / 2 */
        int     p, i;

        win = (int *)malloc( sizeof(int)*m );
        t_win = (int *)malloc( sizeof(int)*m );

        for ( i = 0; i < m; i++ )
                win[i] = data[i];

        icopy( win, t_win, m );
        bsort( t_win, m );

        data[md2] = t_win[md2];

        p = 0;
        for ( i = m; i < n; i++ ) {
                win[p] = data[i];
                icopy( win, t_win, m );
                bsort( t_win, m );
                data[i-md2] = t_win[md2];
                p = (p+1) % m;
        }

        free( win );
        free( t_win );
}

/*---------------------------------------------------------------------------
**      Function:       bsort
**      Intent:         Bubble sort in ascending order.
**      Arguments:      data    the input data array (modify-in-place)
**                      n       number of data points in data
*/
void
bsort( data, n )
int     *data;
int     n;
{
        int     i, j;
        int     temp;

        for ( i = n - 1; i > 0; i-- )
                for ( j = 0; j < i; j++ )
                        if ( data[j] > data[j+1] ) {
                                temp = data[j];
                                data[j] = data[j+1];
                                data[j+1] = temp;
                        }
}


/*---------------------------------------------------------------------------
**      Function:       icopy
**      Intent:         Integer array copy function.
**      Arguments:      x       the input integer array.
**                      y       the output integer array
**                      n       number of data points in x.
*/
void
icopy( x, y, n )
int     *x, *y;
int     n;
{
        int     i;

        for ( i = 0; i < n; i++ )
                y[i] = x[i];
}


/*---------------------------------------------------------------------------
**      Function:       incr
**      Intent:         Compute the first-order difference of the input data.
**                      The input values are modulo-8. The output values are in
**                      the range of -4 to +3.
**      Arguments:      data    the input data array (modify-in-place)
**                      n       number of data points in data
*/
void
incr( data, n )
int     *data;
int     n;
{
        int     temp1, temp2;
        int     i;

        temp1 = data[0];
        data[0] = 0;

        for ( i = 1; i < n; i++ ) {
                temp2 = data[i];
                temp1 = temp2 - temp1;
                temp1 = temp1 > 3 ? temp1 - 8 : temp1;
                data[i] = temp1 < -4 ? temp1 + 8 : temp1;
                temp1 = temp2;
        }
}


/*---------------------------------------------------------------------------
**      Function:       sum
**      Intent:         Convolve the data with an all-1's kernel of size m.
**                      The result of each value in the data array is
**                      essentially the sum of all numbers within the window.
**      Arguments:      data    the input data array (modify-in-place)
**                      n       number of data points in data
**                      m       size of kernel (preferrably odd number)
*/
void
sum( data, n, m )
int     *data;
int     n;
int     m;
{
        int     *win;
        int     sum;
        int     md2 = m/2;
        int     p, i;

        win = (int *)malloc( sizeof(int)*m );

        sum = 0;
        for ( i = 0; i < m; i++ ) {
                win[i] = data[i];
                sum += data[i];
        }

        /*
        **      Convolution begins here.
        */

        data[md2] = sum;

        p = 0;
        for ( i = m; i < n; i++ ) {
                sum -= win[p];
                win[p] = data[i];
                sum += data[i];
                data[i-md2] = sum;
                p = (p+1)%m;
        }


        for ( i = 0; i < md2; i++ )     /* clear the beginning and the end */
                data[i] = 0;

        for ( i = n-md2; i < n; i++ )   /* .. */
                data[i] = 0;

        free( win );
}

/*---------------------------------------------------------------------------
**      Function:       clipping
```

```
**      Intent:         Set all numbers in data which are less than the
**                      threshold value, t, to 0 while leaving the others
**                      unchanged.
**      Arguments:      data    the input data array (modify-in-place)
**                      n       number of data points in data
**                      t       threshold value
*/
void
clipping( data, n, t )
int     *data;
int     n;
int     t;
{
        int     i;

        for ( i = 0; i < n; i++ )
                if ( data[i] > 0 && data[i] < t ||
                     data[i] < 0 && data[i] > -t )
                        data[i] = 0;
}
```

File: filters.h

```
/*-----------------------------------------------------------------------
**      File:           filters.h
**      Intent:         One-dimensional integer array filter function
**                      declarations.
*/

#ifndef _FILTERS_H
#define _FILTERS_H

extern  void    median( int *, int, int );
extern  void    icopy( int *, int *, int );
extern  void    bsort( int *, int );
extern  void    incr( int *, int );
extern  void    sum( int *, int, int );
extern  void    clipping( int *, int , int );

#endif
```

File: global.c

```
/*-----------------------------------------------------------------------
**      File:           global.c
**      Intent:         Global variables.
*/

#include "..\include\humanf.h"
#include "..\curvelib\curvelib.h"

NEIGHBOR neighbor[] =
        {
                { -1, -1, ROOT2 },      /* NWEST */
                { 0, -1, 1. },          /* NORTH */
                { 1, -1, ROOT2 },       /* NEAST */
                { 1, 0, 1. },           /* EAST */
                { 1, 1, ROOT2 },        /* SEAST */
                { 0, 1, 1. },           /* SOUTH */
                { -1, 1, ROOT2 },       /* SWEST */
                { -1, 0, 1. }           /* WEST */
        };
```

File: global.h

```
/*-----------------------------------------------------------------------
**      File:           global.h
**      Intent:         Declarations of global variables.
*/

#ifndef _GLOBAL_H
#define _GLOBAL_H

extern  NEIGHBOR        neighbor[8];

#endif
```

File: humanf.h

```
/*-----------------------------------------------------------------------
**      File:           humanf.h
**      Intent:         Useful macros and definitions.
*/

#ifndef _HUMANF_H
#define _HUMANF_H

#include <stdlib.h>
#include <stdio.h>
#include <graph.h>
#include <conio.h>
#include <fcntl.h>
#include <io.h>
#include <math.h>
#include <string.h>
#include <sys\types.h>
#include <sys\stat.h>
#include "c:\itexpc\include\itexpfg.h"
#include "c:\itexpc\include\stdtyp.h"

/*
**      Type definitions.
*/

typedef unsigned char   UCHAR;

typedef struct coord                            /* integer (x,y) coordinates */
        {
                int     x;
                int     y;
        }       COORD;

typedef struct neighbor                         /* neighbor pels and their distances */
        {
                int     x;                      /* x incremental step */
                int     y;                      /* y incremental step */
                double  len;                    /* length of displacement */
        }       NEIGHBOR;

typedef struct database
        {
                int     nvectors;               /* number of vectors in the database */
                int     vsize;                  /* vector size */
                double  **vector;               /* array of vectors */
                char    **vector_id;            /* array of vector id's */
                int     *vector_ctr;            /* array of vector sample counters */
        }       DATABASE;

/*-----------------------------------------------------------------------
**      Macro:          READ_PIXEL, WRITE_PIXEL
**      Intent:         Work with the frame grabber as well as the VGA memory.
**                      Operation depends on the flag at compiliation time.
*/
#ifdef  _FRAME_GRABBER
#define READ_PIXEL(x,y)         rpixel(x,y)
#else
#define READ_PIXEL(x,y)         _getpixel(x,y)
#endif


#ifdef  _FRAME_GRABBER
#define WRITE_PIXEL(x,y)        wpixel(x,y,255)
#else
#define WRITE_PIXEL(x,y)        _setpixel(x,y)
#endif


/*-----------------------------------------------------------------------
**      Macro:          ROUND
**      Intent:         Round a positive real number x to the closest integer.
*/
#define ROUND(x)        ( (x)-(double)((int)(x)) < 0.5 ? (int)(x) : (int)(x)+1 )

/*-----------------------------------------------------------------------
**      Macro:          DISTANCE
**      Intent:         Compute the distance between 2 points.
*/
#define DISTANCE(x1,y1,x2,y2) \
                        ( sqrt( (double)((x1)-(x2))*(double)((x1)-(x2))+ \
                                (double)((y1)-(y2))*(double)((y1)-(y2)) ) )


/*-----------------------------------------------------------------------
**      Macro:          MOD8, MOD16
**      Intent:         Modulo of 8 and 16.
**                      Range of operation is -8 < x < 16 for MOD8, and
**                      -16 < x < 32 for MOD16.
*/
#define MOD8(x)         ( (x)>7 ? (x)-8 : ((x)<0 ? (x)+8 : (x)) )
#define MOD16(x)        ( (x)>15 ? (x)-16 : ((x)<0 ? (x)+16 : (x)) )


/*-----------------------------------------------------------------------
**      Macro:          FABS
**      Intent:         Floating point absolute function.
*/
#define FABS(x)         ( (x) > 0.0 ? (x) : -(x) )


/*-----------------------------------------------------------------------
**      Macro:          MAX, MIN
**      Intent:         Maximum and minimum of two numbers.
*/
#define MAX(a,b)        ( (a) > (b) ? (a) : (b) )
#define MIN(a,b)        ( (a) < (b) ? (a) : (b) )


#define TRUE            1
#define FALSE           0


#define IMAGE_DX        512             /* horizontal size of the image */
#define IMAGE_DY        480             /* vertical size of the image */

#endif
```

File: image.c

```
/*-----------------------------------------------------------------------
**      File:           image.c
**      Intent:         Image loading and saving routines.
**      Routines:       void    get_image( void )
**                      int     itex_image_header( char *, int, int, int,
**                                                 int, int, char * )
**                      int     loadim_pc( char *, int *, int * )
**                      int     loadim_pc2( char *, int, int, int )
**                      void    saveim_pc( char *, int, int )
*/

#include "..\include\humanf.h"
#include "..\fgrabber\fgrabber.h"

#define BUF_SIZE        4096*4


#ifdef  _FRAME_GRABBER
/*-----------------------------------------------------------------------
**      Function:       get_image ( _FRAME_GRABBER )
**      Intent:         To snap a binary image using the frame grabber board.
```

```c
*/
void
get_image( void )
{
        init_itexpc();

        printf( "Press any hit to freeze the image " );
        printf( "\n" );

        freeze();
}
#else
/*------------------------------------------------------------------------
**      Function:       get_image
**      Intent:         To load a image from an ITEX image file to the VGA
**                      memory.
*/
void
get_image()
{
        int     loadim_pc( char *, int *, int * );

        char    string[80];
        int     dx, dy;

        printf( "Please enter input file name: " );
        scanf( "%s", string );

        if ( !loadim_pc( string, &dx, &dy ) ) {
                fprintf( stderr, "Unable to open input file.\n" );
                exit( 1 );
        }
}
#endif


/*------------------------------------------------------------------------
**      Function:       itex_image_header
**      Intent:         This function returns information regarding the
**                      specified ITEX image file.
**      Arguments:      file_name       the image filename.
**                      x, y            upper left-hand corner of frame memory
**                      dx, dy          size of image
**                      format          0 EIGHT_BIT; 1 COMPRESSION
**                      comment         comment associated with the image file
**      Return value:   0               operation failed.
**                      > 0             operation successful. The value
**                                      represents the offset to the image
**                                      data for the beginning of the file.
*/
int
itex_imege_header( file_name, x, y, dx, dy, format, comment )
char    *file_name;
int     *x, *y, *dx, *dy, *format;
char    *comment;
{
        int     fd;
        char    ftype[2];
        int     csize;

        if ( (fd = open( file_name, O_RDONLY | O_BINARY )) == -1 )
                return 0;

        read( fd, ftype, 2 );
        if ( ftype[0] != 'I' || ftype[1] != 'M' ) { /* Image file signature */
                close( fd );
                return 0;
        }

        read( fd, &csize, 2 );          /* size fo comment area */
        read( fd, dx, 2 );              /* horizontal size of the image */
        read( fd, dy, 2 );              /* vertical size of the image */
        read( fd, x, 2 );               /* upper left-hand corner x */
        read( fd, y, 2 );               /* upper left-hand corner y */
        read( fd, format, 2 );          /* 0 = EIGHT_BIT; 1 = COMPRESSION */
        read( fd, comment, 50 );        /* reserved by ITEX */
        read( fd, comment, csize );     /* comment */

        close( fd );

        return( 64+csize );             /* offset to image data */
}


/*------------------------------------------------------------------------
**      Function:       loadim_pc
**      Intent:         Load an uncompressed ITEX image into the VGA memory
**                      at frame position (0,0).
**      Arguments:      file_name       the image filename.
**                      width           width of image in pixels.
**                      length          length of image in pixels.
**      Return values:  0               operation failed.
**                      1               operation successful.
*/
int
loadim_pc( file_name, width, length )
char    *file_name;
int     *width, *length;
{
        int     offset;
        int     x, y, dx, dy, format;
        char    comment[256];
        int     fd;
        UCHAR   *bp;
        int     len, temp, i;

        if ( (offset = itex_image_header( file_name,
                &x, &y, &dx, &dy, &format, comment )) == 0 )
                return 0;       /* unable to open input file */

        fd = open( file_name, O_RDONLY | O_BINARY );

        if ( format == 1 )
                return 0;       /* unable to process compressed image */

        read( fd, comment, offset );    /* move to the beginning of the data */

        /*
        **      Now read the image data.
        */
        _setvideomode( _VRES16COLOR );
        _setcolor( 255 );

        bp = (UCHAR *)malloc( BUF_SIZE );

        x = y = 0;

        while ( (len = read( fd, bp, BUF_SIZE )) ) {
                for ( i = 0; i < len; i++ ) {
                        if ( bp[i] != 0 )
                                WRITE_PIXEL( x, y );

                        if ( ++x == dx ) {
                                x = 0;
                                y++;
                        }
                }
        }

        close( fd );
        free( bp );

        *width = dx;
        *length = dy;

        return 1;
}


/*------------------------------------------------------------------------
**      Function:       saveim_pc
**      Intent:         Save an uncompressed ITEX image from the VGA memory
**                      at frame position (0,0) to a file.
**      Arguments:      file_name       the output filename.
**                      width           width of image in pixels.
**                      length          length of image in pixels.
*/
void
saveim_pc( file_name, width, length )
char    *file_name;
int     width, length;
{
        FILE    *fp;
        UCHAR   *bp;
        int     zero = 0;
        int     x, y;
        int     ctr;
        int     done;
        int     i;

        bp = (UCHAR *)malloc( BUF_SIZE );

        fp = fopen( file_name, "wb" );

        fwrite( "IM", 1, 2, fp );
        fwrite( &zero, sizeof(int), 1, fp );    /* 0 comment size */
        fwrite( &width, sizeof(int), 1, fp );   /* dx */
        fwrite( &length, sizeof(int), 1, fp );  /* dy */
        fwrite( &zero, sizeof(int), 1, fp );    /* x origin */
        fwrite( &zero, sizeof(int), 1, fp );    /* y origin */
        fwrite( &zero, sizeof(int), 1, fp );    /* format - EIGHT_BIT */
        for ( i = 0; i < 50; i++ )              /* reserved by ITEX */
                fwrite( "\0", 1, 1, fp );

        x = y = 0;
        done = 0;
        while ( !done ) {
                ctr = 0;
                while ( ctr < BUF_SIZE ) {
                        bp[ctr++] = READ_PIXEL( x, y ) == 0 ? 0 : 255;
                        if ( ++x == width ) {
                                x = 0;
                                if ( ++y == length ) {
                                        done = 1;
                                        break;
                                }
                        }
                }
                fwrite( bp, 1, ctr, fp );
        }

        fclose( fp );
}


/*------------------------------------------------------------------------
**      Function:       loadim_pc2
**      Intent:         Load an uncompressed ITEX image into the VGA memory
**                      at the specified frame position with a linear
**                      reduction in size.
**      Arguments:      file_name       the image filename.
**                      x_org, y_org    upper left-hand corner of the image.
**                      reduction       reduction factor in one dimension.
**      Return value:   0               operation failed.
**                      1               operation successful.
*/
int
loadim_pc2( file_name, x_org, y_org, reduction )
char    *file_name;
int     x_org, y_org;
int     reduction;
{
        int     offset;
        int     x, y, dx, dy, format;
        char    comment[256];
        int     fd;
        UCHAR   *bp;
        int     x_end, temp;
        int     i, j, k, ctr;
        int     th = reduction*reduction/2;     /* threshold value */

        if ( (offset = itex_image_header( file_name,
                &x, &y, &dx, &dy, &format, comment )) == 0 )
                return 0;       /* unable to open input file */

        fd = open( file_name, O_RDONLY | O_BINARY );

        if ( format == 1 )
                return 0;       /* unable to process compressed image */

        read( fd, comment, offset );    /* move to the beginning of the data */

        /*
        **      Now read the image data.
        _setvideomode( _VRES16COLOR );
        */

        bp = (UCHAR *)malloc( dx*reduction );

        x_end = x_org + dx/reduction;
        x = x_org;
        y = y_org;
```

```
	while ( read( fd, bp, dx*reduction ) ) {
		for ( i = 0; i < dx; i += reduction ) {
			ctr = 0;
			for ( j = 0; j < reduction; j++ ) {
				for ( k = 0; k < reduction; k++ ) {
					if ( bp[reduction*j+i+k] != 0 )
						ctr++;
				}
			}

			if ( ctr >= th ) {
				_setcolor( 255 );
				_setpixel( x, y );
			}
			else {
				_setcolor( 0 );
				_setpixel( x, y );
			}

			x++;
		}

		x = x_org;
		y++;
	}

	close( fd );
	free( bp );
	_setcolor( 255 );

	return 1;
}
```

File: image.h

```
/*--------------------------------------------------------------------------
**	File:		image.h
**	Intent:		Image loading and saving function declarations.
*/

#ifndef _IMAGE_H
#define _IMAGE_H

extern	void	get_image( void );

/*
**	ITEX image file routines.
*/
extern	int	itex_image_header( char *, int, int, int, int, int, char * );
extern	int	loadim_pc( char *, int *, int * );
extern	int	loadim_pc2( char *, int, int, int );
extern	void	saveim_pc( char *, int, int );

#endif
```

File: matching.c

```
/*--------------------------------------------------------------------------
**	File:		matching.c
**	Intent:		This is the main program for the recognition system.
**			It compares the test vector of an input profile to
**			the vectors in the database and outputs the result.
**
*/

#include "..\include\humanf.h"
#include "..\database\database.h"
#include "..\classify\classify.h"
#include "..\vector\vector.h"

#define THRESHOLD16	0.000400	/* threshold for 16-point comparison */


void
main( argc, argv )
int	argc;
char	**argv;
{
	void		fill_page( char * );
	void		show_distance( DATABASE, double * );

	char		buf[80];
	double		*vector;	/* test vector */
	DATABASE	database;	/* database being used */
	int		ncompare;	/* # of coefs used in the comparison */
	int		vnum;		/* vector # selected for the match */
	double		*distance;	/* distances from the test vector */
	int		vector_ok;	/* get_vector() successful */
	int		batch = 0;	/* 0 for non-batch processing */
	int		b_ctr = 0;	/* number of faces processed */
	FILE		*fp;		/* batch output file pointer */

	if ( argc > 1 && !strcmp( argv[1], "-b" ) ) {
		batch = 1;
		fp = fopen( "batch.out", "w" );
	}

	printf( "Please enter the name of the database: " );
	scanf( "%s", buf );

	if ( !load_database( &database, buf ) ) {
		fprintf( stderr, "Unable to open the database file.\n" );
		exit( -1 );
	}

	printf( "Database loaded.\n" );
	printf( "Vector size: %d\n", database.vsize );
	printf( "Number of vectors in the database: %d\n", database.nvectors );

	vector = (double *)malloc( sizeof(double)*database.vsize );
	distance = (double *)malloc( sizeof(double)*database.nvectors );

	while ( 1 ) {
		printf( "How many coefficients do you wish to use in "
			"the comparison? " );
		scanf( "%d", &ncompare );
		if ( ncompare < 1 || ncompare > database.vsize )
			continue;
		break;
	}

	while ( 1 ) {
		if ( !get_vector( database.vsize, vector ) ) {
			vector_ok = 0;
		}
		else
			vector_ok = 1;

		vnum = classify( database, vector, ncompare, distance );

#ifndef _FRAME_GRABBER
		if ( !batch )
			getch();

		_setvideomode( _DEFAULTMODE );
#endif

		b_ctr++;
		if ( vector_ok && distance[vnum] < THRESHOLD16 ) {
			fill_page( database.vector_id[vnum] );
			if ( batch )
				fprintf( fp, "%d\t%d\n", b_ctr, vnum+1 );
		}
		else {
			if ( vector_ok ) {
				fill_page( "Unable to identify human subject" );
				if ( batch )
					fprintf( fp, "%d\t0\n", b_ctr );
			}
			else {
				fill_page( "Unable to extract face profile" );
				if ( batch )
					fprintf( fp,
			"%d\tUnable to extract face profile.\n", b_ctr );
			}
		}

		if ( !batch )
			getch();

		if ( vector_ok )
			show_distance( database, distance );

		printf( "\n\nDo you wish to continue? (y/n) " );
		scanf( "%1s", buf );
		if ( *buf == 'n' )
			break;
	}

	if ( batch )
		fclose( fp );

	printf( "Goodbye.\n" );
	exit( 0 );
}


/*--------------------------------------------------------------------------
**	Function:	fill_page
**	Intent:		Fill the screen with the given string.
**	Argument:	string		the input string.
*/
void
fill_page( string )
char	*string;
{
	char	buf[80];
	int	len;
	int	i;

	strcpy( buf, string );
	strcat( buf, "  " );			/* add some spaces */
	len = strlen( buf );

	system( "cls" );			/* clear the screen for output */

	for ( i = 0; i < 1600/len; i++ )		/* 80 character * 20 lines */
		printf( "%s", buf );

	printf( "\n" );
}


/*--------------------------------------------------------------------------
**	Function:	show_distance
**	Intent:		Display the distance from the test vector to each
**			vector in the database.
**	Argument:	database	the database in use.
**			distance	an array containing the distance
**					from the test vector to each vector
**					in the database.
*/
void
show_distance( database, distance )
DATABASE	database;
double		*distance;
{
	int	i;
	char	buf[80];
	int	space_left = 80;

	system( "cls" );			/* clear the screen for output */

	for ( i = 0; i < database.nvectors; i++ ) {
		sprintf( buf, "%s %-6.0f  ",
			database.vector_id[i], distance[i]*1000000 );
		space_left -= strlen( buf );

		if ( space_left > 0 )
			printf( "%s", buf );
		else {
			printf( "\n%s", buf );
			space_left = 80 - strlen( buf );
		}
	}
}
```

File: sampling.c

```
/*---------------------------------------------------------------------------
**      File:           sampling.c
**      Intent:         Open curve sampling function and display functions.
**      Routines:       int     csample( int *, int, double, COMPLEX ** )
**                      void    sdisplay( COMPLEX *, int, int, int )
**                      void    cdisplay( int *, int, int )
**                      void    cdisplay2( int *, int, int, double )
*/

#include "..\include\humanf.h"
#include "..\curvelib\curvelib.h"
#include "..\fft\complex.h"
#include "..\global\global.h"

#define CHAIN_TERMINATION       255     /* chain code termination value */

/*---------------------------------------------------------------------------
**      Function:       csample
**      Purpose:        To extract a specified number of interpolated
**                      points from an open curve in from of chain code.
**                      The curve sample array is terminated with a negative
**                      coordinate value.
**      Arguments:      curve           chain code of the input open curve.
**                      nsamples        number of samples to be obtained.
**                      clength         the total length of the curve.
**                      curve_sample    the output curve samples.
**      Return value:   0               operation failed.
**                      1               operation successful.
*/
int
csample( curve, nsamples, clength, curve_sample )
int     *curve;
int     nsamples;
double  clength;
COMPLEX **curve_sample;
{
        COMPLEX *sample;
        int     curr_x, curr_y;         /* current pixel position */
        int     prev_x, prev_y;         /* previous pixel position */
        double  prev_pos, curr_pos;     /* curve length since the starting pt */
        double  desired_pos;            /* desired curve len from starting pt */
        double  fraction;               /* fraction in length between pixels */
        int     next_move;              /* direction of next movement */
        double  slen = clength/(double)(nsamples-1);    /* segment length */
        int     m = 2;                  /* data start from curve[2] */
        int     i = 1;                  /* index of curve_sample */

        *curve_sample = sample =
                (COMPLEX *)malloc( sizeof(COMPLEX)*(nsamples+1) );

        prev_x = curr_x = curve[0];     /* header contains starting position */
        prev_y = curr_y = curve[1];     /* ditto */
        prev_pos = curr_pos = 0.0;      /* first position is zero */
        sample[0].x = (double)curr_x;   /* first sample is the first point */
        sample[0].y = (double)curr_y;   /* ditto */
        desired_pos = slen;             /* next sample position */

        while ( curve[m] != CHAIN_TERMINATION ) {
                next_move = curve[m];
                curr_x += neighbor[next_move].x;
                curr_y += neighbor[next_move].y;
                curr_pos += neighbor[next_move].len;

                if ( curr_pos >= desired_pos ) {
                        fraction = (double)(desired_pos - prev_pos) /
                                        (double)(curr_pos - prev_pos);

                        sample[i].x = prev_x + fraction*(curr_x - prev_x);
                        sample[i].y = prev_y + fraction*(curr_y - prev_y);

                        desired_pos += slen;    /* next desired position */
                        i++;
                }

                prev_x = curr_x;
                prev_y = curr_y;
                prev_pos = curr_pos;
                m++;
        }

        /*
        **      Ensure the last point is included in the samples.
        */
        sample[nsamples-1].x = (double)curr_x;  /* last sample is the last pt */
        sample[nsamples-1].y = (double)curr_y;  /* ditto */

        sample[nsamples].x = -1.;       /* array termination */
        sample[nsamples].y = -1.;       /* ditto */

        if ( curr_pos != clength )      /* inconsistent result */
                return 0;
        else
                return 1;               /* operation successful */
}

/*---------------------------------------------------------------------------
**      Function:       sdisplay
**      Purpose:        Output the sample points on screen.
**      Arguments:      curve_sample            curve sample array.
**                      nsamples                number of samples.
**                      frame_x, frame_y        coordinate of the upper-left
**                                              corner of the output frame.
*/
void
sdisplay( curve_sample, nsamples, frame_x, frame_y )
COMPLEX  *curve_sample;
int     nsamples;
int     frame_x, frame_y;
{
        int     curr_x, curr_y;
        int     prev_x, prev_y;
        int     i;

        curr_x = ROUND( curve_sample[0].x );
        curr_y = ROUND( curve_sample[0].y );
        WRITE_PIXEL( frame_x+curr_x, frame_y+curr_y );

        for ( i = 1; i < nsamples; i++ ) {
                curr_x = ROUND(curve_sample[i].x );
                curr_y = ROUND(curve_sample[i].y );
                WRITE_PIXEL( frame_x+curr_x, frame_y+curr_y );
        }
}

/*---------------------------------------------------------------------------
**      Function:       cdisplay
**      Purpose:        Output the curve on screen.
**      Arguments:      curve                   chain code of the open curve.
**                      frame_x, frame_y        coordinate of the upper-left
**                                              corner of the output frame.
*/
void
cdisplay( curve, frame_x, frame_y )
int     *curve;
int     frame_x, frame_y;
{
        int     curr_x, curr_y;
        int     next_move;
        int     m = 2;                  /* data start from curve[2] */

        curr_x = curve[0];              /* starting coordinate header */
        curr_y = curve[1];              /* ditto */

        WRITE_PIXEL( frame_x+curr_x, frame_y+curr_y );

        while ( curve[m] != CHAIN_TERMINATION ) {
                next_move = curve[m];
                curr_x += neighbor[next_move].x;
                curr_y += neighbor[next_move].y;
                WRITE_PIXEL( frame_x+curr_x, frame_y+curr_y );
                m++;
        }
}

/*---------------------------------------------------------------------------
**      Function:       cdisplay2
**      Purpose:        Output the curve on screen with linear reduction.
**      Arguments:      curve                   chain code of the open curve.
**                      frame_x, frame_y        position of the upper-left
**                                              corner of the output frame.
**                      reduction               linear reduction factor.
*/
void
cdisplay2( curve, frame_x, frame_y, reduction )
int     *curve;
int     frame_x, frame_y;
double  reduction;
{
        double  curr_x, curr_y;         /* current position */
        int     next_move;              /* direction of next movement */
        int     m = 2;                  /* data start from curve[2] */

        /*
        **      The header contains the starting position.
        */
        curr_x = (double)curve[0]/reduction;
        curr_y = (double)curve[1]/reduction;

        frame_x = ROUND((double)frame_x/reduction);
        frame_y = ROUND((double)frame_y/reduction);

        WRITE_PIXEL( frame_x+ROUND(curr_x), frame_y+ROUND(curr_y) );

        while ( curve[m] != CHAIN_TERMINATION ) {
                next_move = curve[m];
                curr_x += (double)neighbor[next_move].x/reduction;
                curr_y += (double)neighbor[next_move].y/reduction;
                WRITE_PIXEL( frame_x+ROUND(curr_x), frame_y+ROUND(curr_y) );
                m++;
        }
}
```

File: sampling.h

```
/*---------------------------------------------------------------------------
**      File:           sampling.h
**      Intent:         Open curve sampling function and display function
**                      declarations.
*/

#include "..\fft\complex.h"

#ifndef _SAMPLING_H
#define _SAMPLING_H

extern  int     csample( int *, int, double, COMPLEX ** );
extern  void    sdisplay( COMPLEX *, int, int, int );
extern  void    cdisplay( int *, int, int );
extern  void    cdisplay2( int *, int, int, double );

#endif
```

File: training.c

```
/*---------------------------------------------------------------------------
**      File:           training.c
**      Intent:         This is the main program for entering new entries to
**                      database or blending new vectors in the existing ones.
*/

#include "..\include\humanf.h"
#include "..\database\database.h"
#include "..\vector\vector.h"

void
main( argc, argv )
int     argc;
char    **argv;
```

```c
{
        char            database_name[80];
        char            buf[80], buf2[80];
        double          *vector;
        DATABASE        database;
        int             vnum;
        int             vsize;
        int             vector_ok;              /* extraction successful */
        int             new_database = 0;       /* default value */
        int             same_person = 0;        /* default value */
        int             batch = 0;              /* non-batch processing */

        if ( argc > 1 && !strcmp( argv[1], "-b" ) )     /* batch input */
                batch = 1;

        printf( "Please enter the name of the database: " );
        scanf( "%s", database_name );

        if ( !load_database( &database, database_name ) ) {
                printf( "Unable to open the database file.\n" );
                printf( "Do you wish to create one? (y/n) " );
                scanf( "%1s", buf );
                if ( *buf == 'y' ) {    /* create a new database */
                        printf( "What is the vector size for the "
                                "new database? " );
                        scanf( "%d", &vsize );
                        create_database( &database, vsize );
                        new_database = 1;
                }
                else {
                        printf( "Goodbye.\n" );
                        exit( 0 );
                }
        }

        if ( !new_database )
                printf( "Database loaded.\n" );
        else
                printf( "Database created.\n" );

        printf( "Vector size: %d\n", database.vsize );
        printf( "Number of vectors in the database: %d\n", database.nvectors );

        vector = (double *)malloc( sizeof(double)*database.vsize );

        /*
        **      Get face profile vector here.
        */
        while ( 1 ) {
                if ( get_vector( database.vsize, vector ) )
                        vector_ok = 1;
                else
                        vector_ok = 0;

#ifndef _FRAME_GRABBER
                if ( !batch )
                        getch();

                _setvideomode( _DEFAULTMODE );
#endif

                if ( !vector_ok )
                        printf( "Unable to extract vector.\n" );

                if ( !same_person ) {
                        printf( "What is the name of the person? "
                                "(first_name last_name) " );
                        scanf( "%s%s", buf, buf2 );

                        strcat( buf, " " );
                        strcat( buf, buf2 );
                }

                if ( (vnum = find_vector_id( database, buf )) == -1 ) {
                        printf( "Unable to find the name, %s, "
                                "in the database.\n", buf );
                        printf( "Do you want to add this person "
                                "to the database? (y/n) " );
                        scanf( "%1s", buf2 );

                        if ( vector_ok && *buf2 == 'y' ) {
                                add_vector( &database, vector, buf );
                                save_database( database, database_name );
                        }
                        else
                                printf( "Vector discarded.\n" );
                }
                else {
                        if ( !same_person )
                                printf( "Person's name found.\n" );

                        printf( "Number of vectors blended together: %d\n",
                                database.vector_ctr[vnum] );
                        printf( "Do you want blend this vector "
                                "into the existing one? (y/n) " );
                        scanf( "%1s", buf2 );

                        if ( vector_ok && *buf2 == 'y' ) {
                                blend_vector( &database, vector, vnum );
                                save_database( database, database_name );
                        }
                        else
                                printf( "Vector discarded.\n" );
                }

                printf( "Do you wish to continue? (y/n) " );
                scanf( "%1s", buf2 );
                if ( *buf2 == 'n' )
                        break;
                else {
                        printf( "Will that be the same person? (y/n) " );
                        scanf( "%1s", buf2 );
                        if ( *buf2 == 'y' )
                                same_person = 1;
                        else
                                same_person = 0;
                }
        }

        save_database( database, database_name );
        printf( "Goodbye.\n" );
        exit( 0 );
}
```

File: vector.c

```c
/*---------------------------------------------------------------------------
**      File:           vector.c
**      Intent:         To obtain Fourier descriptors vectors from face
**                      profile images.
**      Routines:       int     get_vector( int, double * )
*/

#include "..\include\humanf.h"
#include "..\image\image.h"
#include "..\curvelib\curvelib.h"
#include "..\extract\extract.h"
#include "..\sampling\sampling.h"
#include "..\feature\feature.h"
#include "..\fft\complex.h"
#include "..\fft\fft.h"

#define DISPLAY_OFFSET  60      /* curve display offset */
#define MAX_CHAIN_SIZE  1024    /* maximum chain size */
#define REF_NPOINTS     20      /* #points in the reference dotted line */
#define RATIO_A         0.75    /* ratio for the upper termination */
#define RATIO_B         0.5     /* ratio for the lower termination */


/*---------------------------------------------------------------------------
**      Function:       get_vector
**      Intent:         To get a Fourier descriptors vector from a
**                      face profile image.
**      Arguments:      vsize   the desired vector size. It should be
**                              FFT_SIZE/2.
**                      vector  the output vector.
**      Return value:   0       operation failed.
**                      1       operation successful.
*/
int
get_vector( vsize, vector )
int     vsize;
double  *vector;
{
        static void     draw_cross_marker( int, int );
        static void     draw_square_marker( int, int );
        static void     draw_dline( int, int, int, int, int );
        static void     trace_back( COMPLEX *, int, COMPLEX ** );

        COORD   nose_tip_pos;           /* nose tip */
        COORD   nose_bottom_pos;        /* nose bottom */
        COORD   nose_top_pos;           /* nose top */
        COORD   upper_term_pos;         /* upper terminating position */
        COORD   chin_pos;               /* chin position */
        COORD   lower_term_pos;         /* lower terminating position */
        COORD   end_pos;

        int     fft_size = vsize*2;     /* size of the FFT operation */
        int     nsamples = vsize+1;     /* number of samples on the curve */
        int     npoints;                /* number of pixels in the curve */
        int     offset;                 /* horizontal offset for display */
        double  clen;                   /* length of the profile curve */
        COMPLEX *curve_sample;          /* samples of the curve */
        COMPLEX *ncurve;
        double  *mag;                   /* magnitudes of the transformed data */
        int     *chain;                 /* chain code of the profile curve */
        int     i;

        get_image();            /* get image from the camera or an input file */

        nose_tip( &nose_tip_pos );
        nose_bottom( nose_tip_pos, &nose_bottom_pos );
        nose_top( nose_tip_pos, nose_bottom_pos, &nose_top_pos );
        upper_termination( nose_top_pos, nose_bottom_pos, &upper_term_pos );
        chin_position( nose_top_pos, nose_bottom_pos, &chin_pos );
        lower_termination( nose_top_pos, nose_bottom_pos, chin_pos,
                        &lower_term_pos );

        mag = (double *)malloc( sizeof(double)*fft_size );
        chain = (int *)malloc( sizeof(int)*MAX_CHAIN_SIZE );

        get_chain_code( upper_term_pos, WEST, CCW, IMAGE_DY-1, 9999.,
                        lower_term_pos, chain, MAX_CHAIN_SIZE, &npoints,
                        &end_pos, &clen );

        offset = DISPLAY_OFFSET;
        cdisplay( chain, offset, 0 );

        draw_square_marker( offset+nose_top_pos.x, nose_top_pos.y );
        draw_square_marker( offset+nose_tip_pos.x, nose_tip_pos.y );
        draw_square_marker( offset+nose_bottom_pos.x, nose_bottom_pos.y );
        draw_square_marker( offset+upper_term_pos.x, upper_term_pos.y );
        draw_square_marker( offset+chin_pos.x, chin_pos.y );
        draw_square_marker( offset+lower_term_pos.x, lower_term_pos.y );

        draw_dline( offset+nose_top_pos.x, nose_top_pos.y,
                    offset+nose_bottom_pos.x, nose_bottom_pos.y,
                    REF_NPOINTS );
        draw_dline( offset+nose_top_pos.x, nose_top_pos.y,
                    offset+upper_term_pos.x, upper_term_pos.y,
                    (int)((double)(REF_NPOINTS)*RATIO_A) );
        draw_dline( offset+chin_pos.x, chin_pos.y,
                    offset+lower_term_pos.x, lower_term_pos.y,
                    (int)((double)(REF_NPOINTS)*RATIO_B) );

        /*
        **      Sample the curve and obtain the Fourier descriptors vector.
        */
        csample( chain, nsamples, clen, &curve_sample );
        offset += DISPLAY_OFFSET*3/2;
        sdisplay( curve_sample, nsamples, offset, 0 );

        trace_back( curve_sample, nsamples, &ncurve );
        fft( ncurve, fft_size );
        magnitude( ncurve, mag, fft_size );

        /*
        **      Normalize the vector such that vector[0] = 1.0.
        */
        for ( i = 1; i < vsize; i++ )
                vector[i] = mag[i+1] / mag[1];
        vector[0] = 1.0;                /* = mag[1]/mag[1] */
```

```
	free( chain );
	free( mag );
	free( curve_sample );
	free( ncurve );

	return 1;			/* operation successful */
}


/*---------------------------------------------------------------------------
**	Function:	trace_back
**	Intent:		Trace the samples from the beginning to the end and
**			trace back to the beginning to form a closed boundary.
**	Arguments:	curve_sample	the coordinates of the samples.
**			nsamples	number of samples.
**			closed_curve	output closed curve samples.
*/
static void
trace_back( curve_sample, nsamples, closed_curve )
COMPLEX *curve_sample;
int	nsamples;
COMPLEX **closed_curve;
{
	COMPLEX *curve;
	int	i, j;

	*closed_curve = curve =
	(COMPLEX *)malloc( sizeof(COMPLEX)*(nsamples-1)*2 );

	for ( i = 0; i < nsamples; i++ )
		COMPLEX_ASSIGN( curve[i], curve_sample[i] );

	/*
	**	Trace back. Note that the two ending points are not included
	**	in the retrace.
	*/
	j = nsamples;
	for ( i = 1; i < nsamples-1; i++ ) {
		COMPLEX_ASSIGN( curve[j], curve_sample[nsamples-1-i] );
		j++;
	}
}


/*---------------------------------------------------------------------------
**	Function:	draw_dline
**	Intent:		Draw a dotted line between two points.
**	Arguments:	x1, y1, x2, y2	two points.
**			npoints		number of points on the dotted line.
*/
static void
draw_dline( x1, y1, x2, y2, npoints )
int	x1, y1, x2, y2;
int	npoints;
{
	double	slope;
	double	dx, dy;
	double	x, y;
	int	i;

#ifndef _FRAME_GRABBER
	_setcolor( 255 );
#endif

	if ( abs(x1-x2) > abs(y1-y2) ) {
		slope = (double)(y1-y2)/(double)(x1-x2);
		dx = (double)(x2-x1)/(double)(npoints-1);

		for ( i = 0; i < npoints; i++ ) {
			x = (double)x1 + dx*(double)i;
			y = (double)y1 - slope*((double)x1-x);
			WRITE_PIXEL( ROUND(x), ROUND(y) );
		}
	}
	else {
		slope = (double)(x1-x2)/(double)(y1-y2);
		dy = (double)(y2-y1)/(double)(npoints-1);

		for ( i = 0; i < npoints; i++ ) {
			y = (double)y1 + dy*(double)i;
			x = (double)x1 - slope*((double)y1-y);
			WRITE_PIXEL( ROUND(x), ROUND(y) );
		}
	}
}


/*---------------------------------------------------------------------------
**	Function:	draw_cross_marker
**	Intent:		To draw a cross-hair marker at the specified
**			position.
**	Arguments:	x, y	the position of the marker.
*/
static void
draw_cross_marker( x, y )
int	x, y;
{
#ifndef _FRAME_GRABBER
	_setcolor( 255 );
#endif

	WRITE_PIXEL( x, y );

	WRITE_PIXEL( x-3, y );
	WRITE_PIXEL( x-2, y );
	WRITE_PIXEL( x-2, y );
	WRITE_PIXEL( x-1, y );
	WRITE_PIXEL( x+1, y );
	WRITE_PIXEL( x+2, y );
	WRITE_PIXEL( x+3, y );

	WRITE_PIXEL( x, y-3 );
	WRITE_PIXEL( x, y-2 );
	WRITE_PIXEL( x, y-1 );
	WRITE_PIXEL( x, y+1 );
	WRITE_PIXEL( x, y+2 );
	WRITE_PIXEL( x, y+3 );
}


/*---------------------------------------------------------------------------
**	Function:	draw_square_marker
**	Intent:		To draw a square marker at the specified position.
**	Arguments:	x, y	the position of the marker.
*/
static void
draw_square_marker( x, y )
int	x, y;
{
#ifndef _FRAME_GRABBER
	_setcolor( 255 );
#endif

	WRITE_PIXEL( x-2, y-2 );
	WRITE_PIXEL( x-1, y-2 );
	WRITE_PIXEL( x, y-2 );
	WRITE_PIXEL( x+1, y-2 );
	WRITE_PIXEL( x+2, y-2 );
	WRITE_PIXEL( x+2, y-1 );
	WRITE_PIXEL( x+2, y );
	WRITE_PIXEL( x+2, y+1 );
	WRITE_PIXEL( x+2, y+2 );
	WRITE_PIXEL( x+1, y+2 );
	WRITE_PIXEL( x, y+2 );
	WRITE_PIXEL( x-1, y+2 );
	WRITE_PIXEL( x-2, y+2 );
	WRITE_PIXEL( x-2, y+1 );
	WRITE_PIXEL( x-2, y );
	WRITE_PIXEL( x-2, y-1 );
}
```

File: vector.h

```
/*---------------------------------------------------------------------------
**	File:		vector.h
**	Intent:		Function declaration for a high-level function which
**			obtains Fourier descriptors vectors from face profile
**			images.
*/

#ifndef _VECTOR_H
#define _VECTOR_H

extern	int	get_vector( int, double * );

#endif
```