CERTIFICATION OF APPROVAL

BIOMETRICS: FINGERPRINT RECOGNITION

by

Raja Mohd. Firdaus Raja Aljunid

A project dissertation submitted to the
Electical and Electronics Engineering Programme
Universiti Teknologi PETRONAS
In partial fulfillment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved by,

[illegible]

Mr. Patrick Sebastian
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
DECEMBER 2004

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done unspecified sources or persons.

[illegible]

Raja Mohd. Firdaus Raja Aljunid

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

I want to praise to the AL-Mighty Allah s.w.t for giving me the chance to finish my project for this whole year. I want to thank to my family for supporting me for the whole 5 year when I'm studying in University Technology of PETRONAS. I want to give my greatest gratitude to my supervisor, Mr. Patrick Sebastian from Electrical & Electronics Department for supervising me and helping me during this year for this projet. My gratitude also for Mr. Luigi Rosa, an engineer for SIEMENS in Italy for helping me with the programming codes. Dr. Salil Prabhakar for the theory of the image processing especially in fingerprint recognition. My technician, Miss Siti Hawa from the Electrical & Electronics Department of UTP. My friends for supporting and helping me with this project. Far and foremost, the University Technology of PETRONAS that has given me the chance to study in engineering here. Thank you.

ABSTRACT

Accurate automatic personal identification is critical in a variety of applications in our electronically interconnected society. Biometrics, which refers to identification based on physical or behavioral characteristics, is being increasingly adopted to provide positive identification with a high degree of confidence. Among all the biometric techniques, fingerprint-based authentication systems have received the most attention because of the long history of fingerprints and their extensive use in forensics.
However, the numerous fingerprint systems currently available still do not meet the stringent performance requirements of several important civilian applications. To assess the performance limitations of popular minutiae-based fingerprint verification system, we theoretically estimate the probability of a false correspondence between two fingerprints from different fingers based on the minutiae representation of fingerprints. Due to the limited amount of information present in the minutiae-based representation, it is desirable to explore alternative representations of fingerprints.

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Fingerprint identification is one of the most important biometric methods used in security. The task in this project is to develop and implement a program that is able to identify or verify a fingerprint.

1.2 Objectives and Scope of Study

The scope of study is to design and construct an algorithm on identifying a person using fingerprints. The objective of the project is to develop and understand the algorithm in a fingerprint identification system.

The scope of the study is in what are/is the program being used in the development stage. For this Final Year Design Project, MATLAB has been choose to design the project and develop the program.

1.3 Background of Study

With the advent of electronic banking, e-commerce, and smartcards and an increased emphasis on the privacy and security of information stored in various databases, automatic personal identification has become a very important topic. Accurate automatic personal identification is now needed in a wide range of civilian applications involving the use of passports, cellular telephones, automatic teller machines, and driver licenses. Traditional knowledge-based (password or Personal Identification Number (PIN)) and token-based (passport, driver license, and ID card) identifications are prone to fraud because PINs may be forgotten or guessed by an imposter and the tokens may be lost or stolen. Therefore, traditional knowledge-based and token-based approaches are unable to satisfy the security requirements of our electronically interconnected information society (see Figure 1.1).

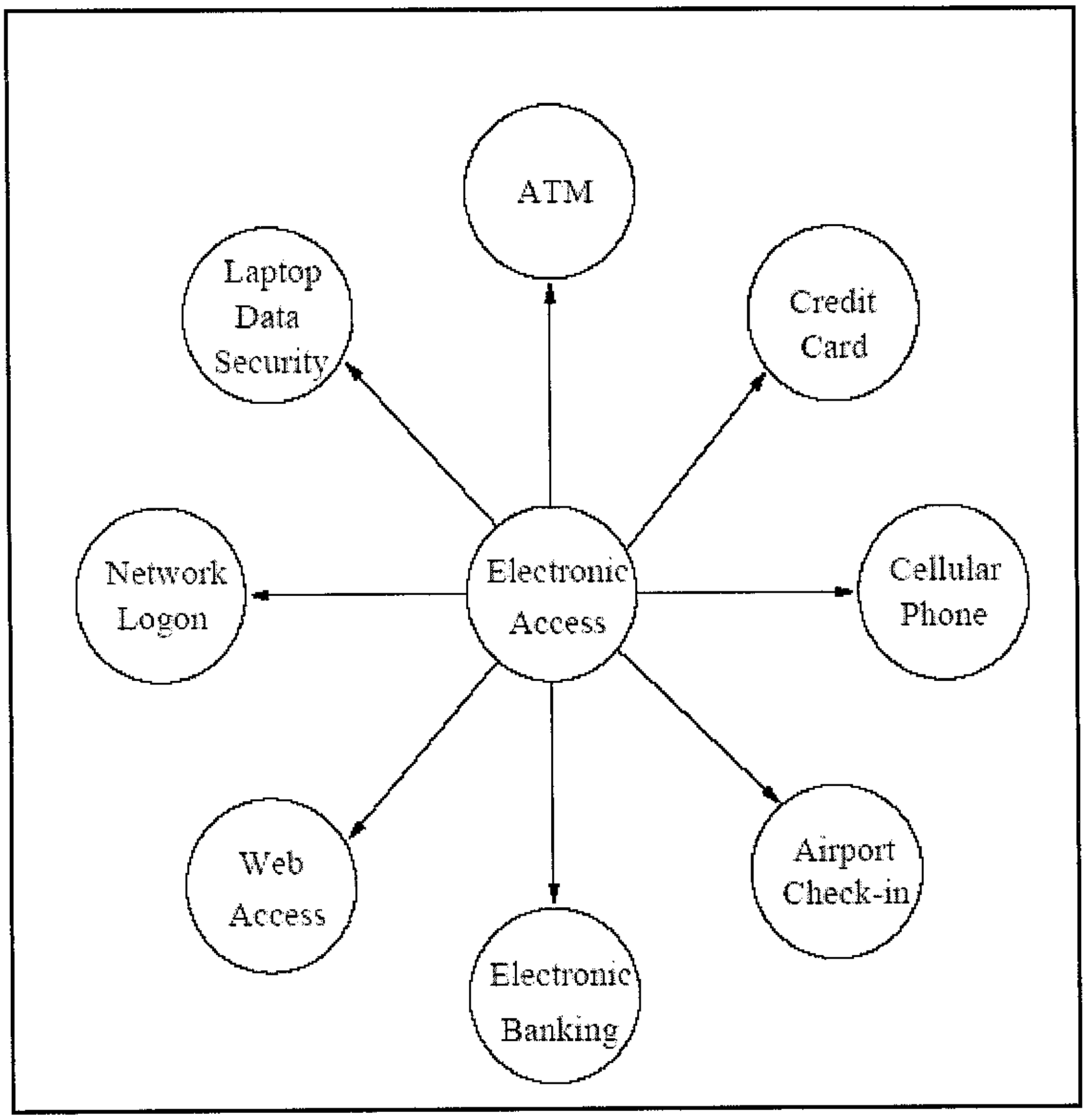


Figure 1.1: Various electronic access applications in widespread use that require automatic authentication.

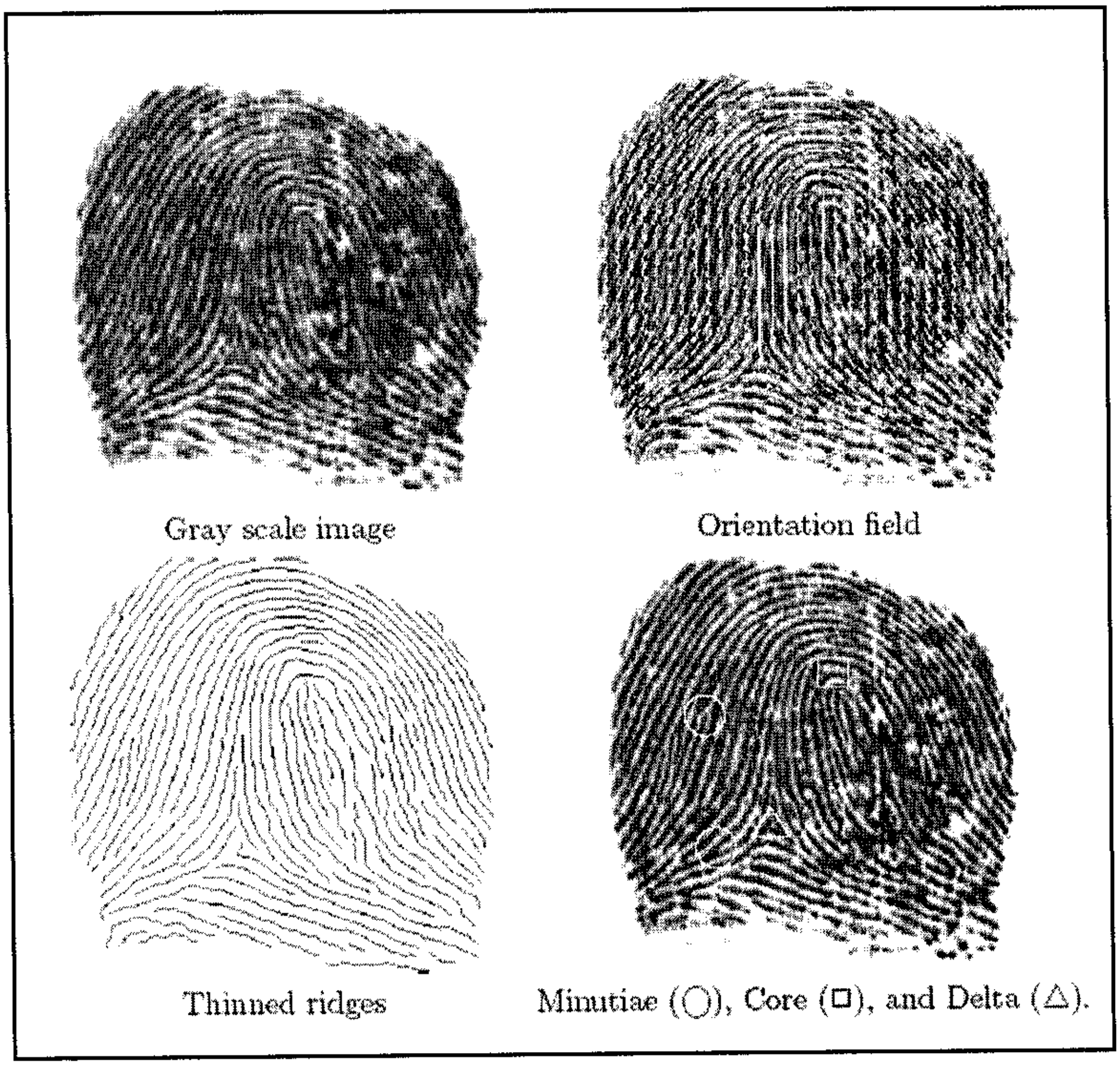


Figure 1.2: Orientation field, thinned ridges, minutiae, and singular points.

1.3.1 Introduction to Biometrics

Biometrics, which refers to identifying an individual based on his or her physiological or behavioral characteristics, has the capability to reliably distinguish between an authorized person and an imposter. Since biometric characteristics are distinctive, can not be forgotten or lost, and the person to be authenticated needs to be physically present at the point of identification, biometrics is inherently more reliable and more capable than traditional knowledge-based and token-based techniques. Biometrics also has a number of disadvantages. For example, if a password or an ID card is compromised, it can be easily replaced. However, once a biometrics is compromised, it is not possible to replace it. Similarly, users can have a different password for each account, thus if the password for one account is compromised, the other accounts are still safe. However, if a biometrics is compromised, all biometrics-based accounts can be broken-in. Among all biometrics (e.g., face, fingerprint, hand geometry, iris, retina, signature, voice print, facial, thermogram, hand vein, gait, ear, odor, keystroke dynamics, etc.), fingerprint-based identification is one of the most mature and proven technique.

1.3.1.1 Application of Biometrics

Biometrics has been widely used in forensics applications such as criminal identification and prison security. The biometric technology is rapidly evolving and has a very strong potential to be widely adopted in civilian applications such as electronic banking, e-commerce, and access control. Due to a rapid increase in the number and use of electronic transactions, electronic banking and electronic commerce are becoming one of the most important emerging applications of biometrics. These applications include credit card and smart card security, ATM security, check cashing and fund transfers, online transactions and web access. The physical access control applications have traditionally used token-based authentication. With the progress in biometric technology, these applications will increasingly use biometrics for authentication. Remote login and data access applications have traditionally used knowledge-based authentication. These applications have already started using biometrics for person authentication. The use of biometrics will become more widespread in coming years as the technology matures and becomes more trust worthy. Other biometric applications include welfare disbursement, immigration checkpoints, national ID, voter and driver registration, and time and attendance.

1.3.2 Fingerprint

Fingerprints are the ridge and furrow patterns on the tip of the finger and have been used extensively for personal identification of people. Figure 1.2 shows an example of a fingerprint. The biological properties of fingerprint formation are well understood and fingerprints have been used for identification purposes for centuries. Since the beginning of the 20th century, fingerprints have been extensively used for identification of criminals by the various forensic departments around the world. Due to its criminal connotations, some people feel uncomfortable in providing their fingerprints for identification in civilian applications. However, since fingerprint-based biometric systems offer positive identification with a very high degree of confidence, and compact solid state fingerprint sensors can be embedded in various systems (e.g., cellular phones), fingerprint-based authentication is becoming more and more popular in a number of civilian and commercial applications such as, welfare disbursement, cellular phone access, and laptop computer log-in. The availability of cheap and compact solid state scanners as well as robust fingerprint matchers are two important factors in the popularity of fingerprint-based identification systems. Fingerprints also have a number of disadvantages as compared to other biometrics. For example, approximately 4% of the population does not have good quality fingerprints, manual workers get regular scratches on their fingers which poses a difficulty to the matching system, finger skin peels off due to weather, fingers develop natural permanent creases, temporary creases are formed when the hands are immersed in water for a long time, and dirty fingers can not be properly imaged with the existing fingerprint sensors. Further, since fingerprints can not be captured without the user's knowledge, they are not suited for certain applications such as surveillance.

1.3.2.1 Fingerprint Formation

Fingerprints are fully formed at about seven months of fetus development and finger ridge configurations do not change throughout the life of an individual except due to accidents such as bruises and cuts on the finger tips. This property makes fingerprints a very attractive biometric identifier. Biological organisms, in general, are the consequence of the interaction of genes and environment. It is assumed that the phenotype is uniquely determined by the interaction of a specific genotype and a specific environment. Physical appearance and fingerprints are, in general, a part of an individual's phenotype. In the case of fingerprints, the genes determine the general characteristics of the pattern. Fingerprint formation is similar to the growth of capillaries and blood vessels in angiogenesis. The general characteristics of the fingerprint emerge as the skin on the fingertip begins to differentiate. However, the flow of amniotic fluids around the fetus and its position in the uterus change during the differentiation process. Thus, the cells on the fingertip grow in a microenvironment that is slightly different from hand to hand and finger to finger. The finer details of the fingerprints are determined by this changing microenvironment. A small difference in microenvironment is amplified by the differentiation process of the cells. There are so many variations during the formation of fingerprints that it would be virtually impossible for two fingerprints to be alike. But since the fingerprints are differentiated from the same genes, they will not be totally random patterns either. We could say that the fingerprint formation process is a chaotic system rather than a random one.

CHAPTER 2

LITERATURE REVIEW

Necessary information on program structure, design of the system and the flow of the system must be acquired. Another important aspect of the identification and recognition is the part of comparison and recognition result. Below is the program flow of the system.

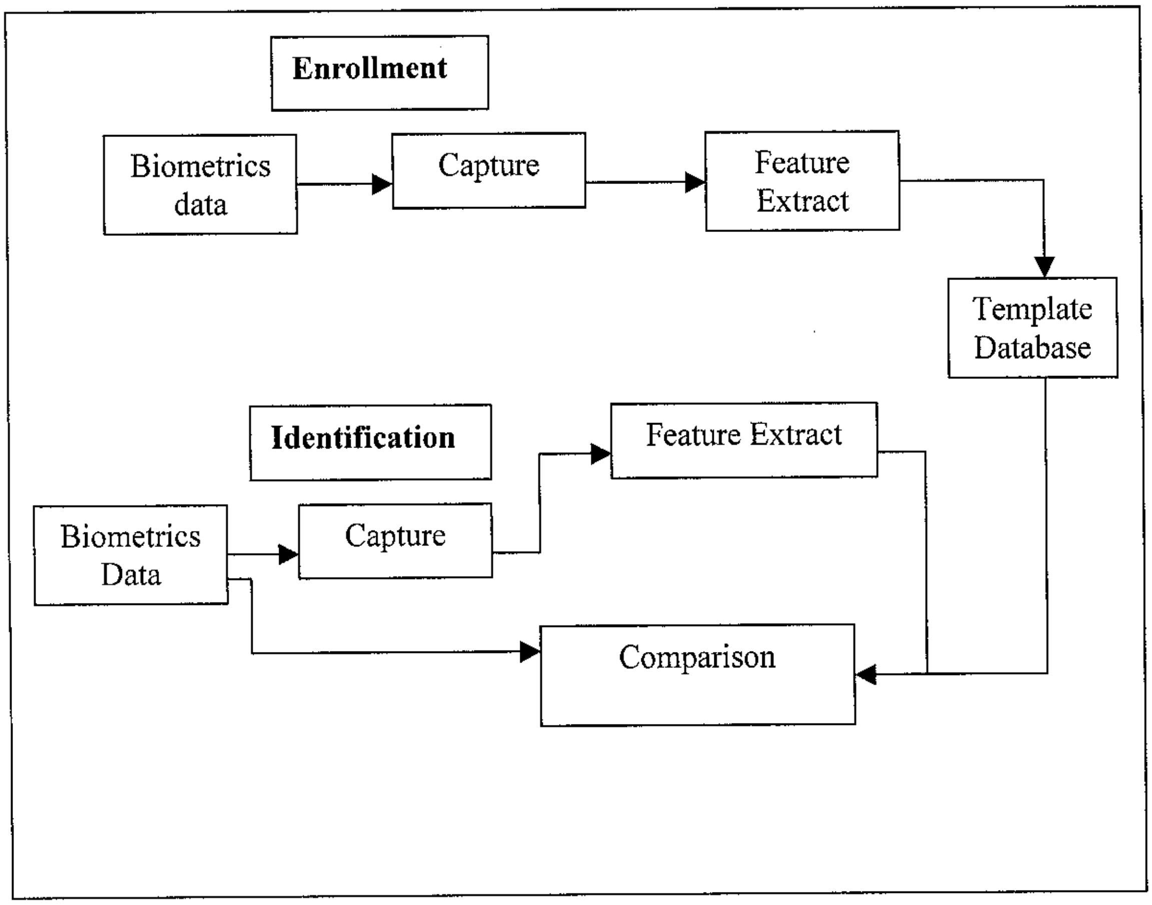


Figure 2.1: The basic theory or flow of the fingerprint recognition

Generally, all biometrics system contain two parts, enrollment and identification part. The enrollment part functions to have a user's characteristic registered so that it can be used as a criterion when identification is performed; whereas the identification part provides the user interface to have then end user's characteristics capture and verified.

- *Capture stage*

  This is the process that a physical or behavioral sample is input to the system. Different system use different devices to get the sample. Generally, physical biometrics data are capture by some type of cameras and the sample is stored as a digital image for processing.

- *Feature Extraction Stage*

  This is the process that the unique data are extracted from the sample and a template is created. The template for any two persons should be different and different sample from same person should be similar enough.

- *Comparison Stage*

  It is the process that the newly extracted template from a sample is compared to a registered in the system. Because even the samples from the same person may vary from time to time, the comparison algorithm should tolerate the tiny changes from the same person yet distinguish different persons. For example, the finger may contact with the live scanner at different place, direction and pressure, so in practice, no two exact same samples.

- *Decision Stage*

  This stage is the process that the system decides whether the template extracted from the new sample matches the registered one.

As was stated previously, this system will be able to identify the owner of a fingerprint with reasonable accuracy and will have the ability to reject a fingerprint when the system is "unsure" of it's results. The system presented has been divided into three stages: preprocessing of a fingerprint image, extraction of the features that represent the fingerprint, and the classification of the fingerprint for a decision or a rejection.

CHAPTER 3

METHODOLOGY

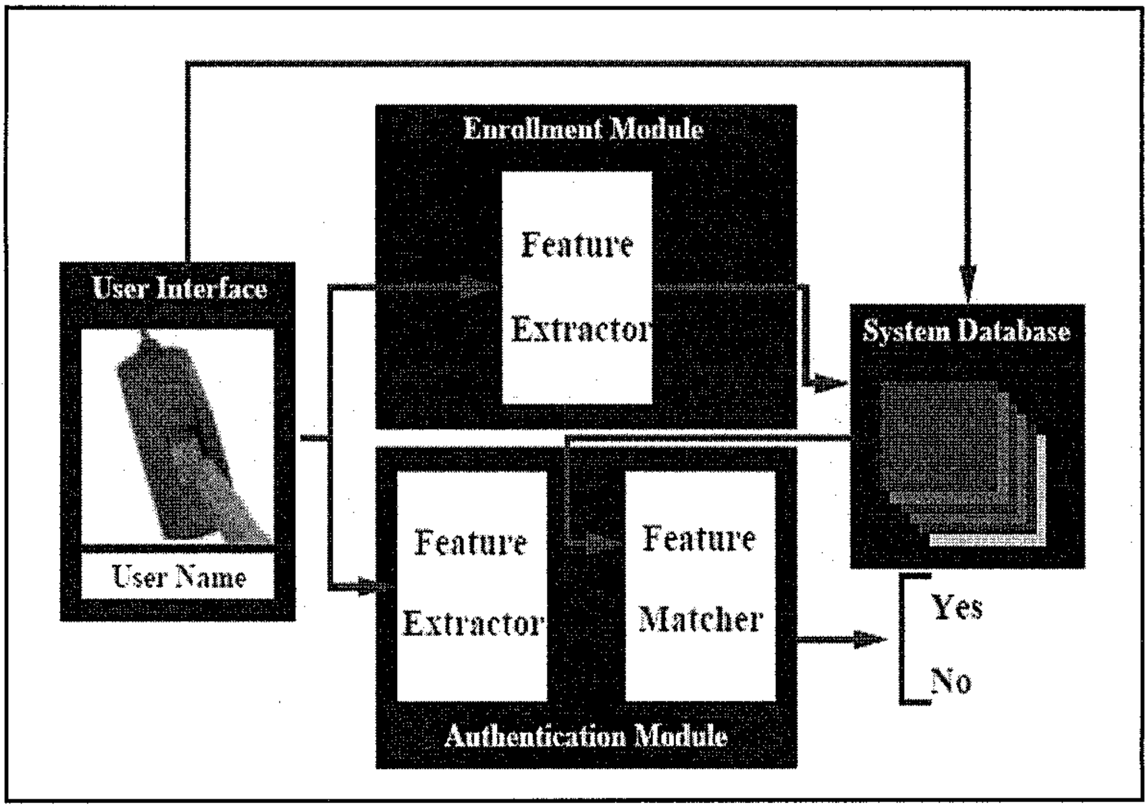


Figure 3.1: System diagram for an automatic verification system.

The figure above, which is figure 3.1, is the standard working or flow of the fingerprint identification. A biometric system can be operated in two modes: 1) verification m ode a nd 2) i dentification m ode. I n t he v erification m ode, a biometric system either accepts or rejects a user's claimed identity while a biometric system operating in the identification mode establishes the identity of the user without a claimed identity. Fingerprint identification is a more difficult problem than fingerprint verification because a huge number of comparisons need to be performed in identification. In this report, I have focused on a biometric system operating in a verification mode and an indexing scheme (fingerprint classification) that can be used in an identification system.

The first part or step of all is to choose the correct programming language to use for this programming. There are many type of language can be use, C++, C language, JAVA, Visual Basic, Visual Basic.Net and many more.

Therefore, for this particular programming, it required a necessary programming tool that has image processing toolbox. So the software or program that will be chosen is MATLAB. I am currently using MATLAB 6 version 12. This software is chosen because it has the capability to process the image using the image processing toolbox given by default. It also has the capability to process high definition of images.

The other programming language is not quite suitable such as C++ or C because the programming itself. We need to define everything including the libraries that are needed. For the GUI programming, it would be really hard because the statement are using a lot of **IF** statements.

For this particular part, a simple GUI is enough for this study. As the objective stated, student just need to show the program is running and the methods used.

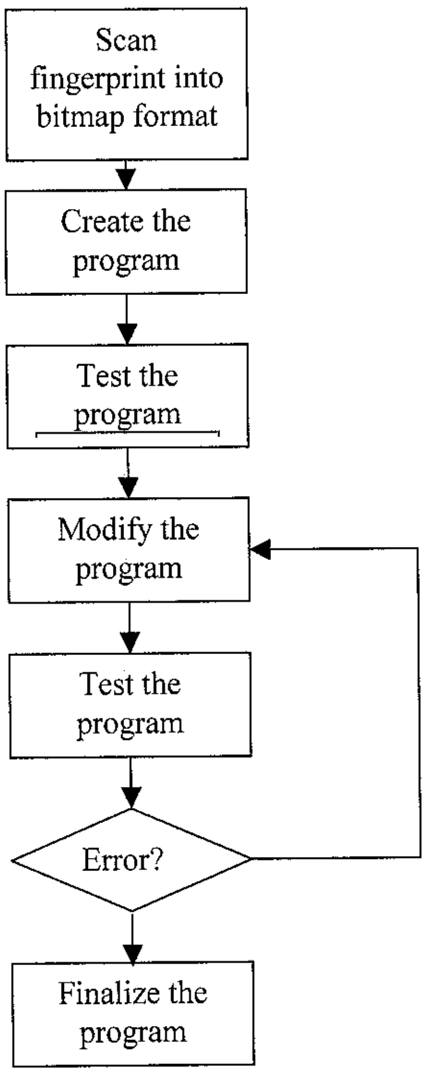


Figure 3.2: Flow chart showing the flow of the project

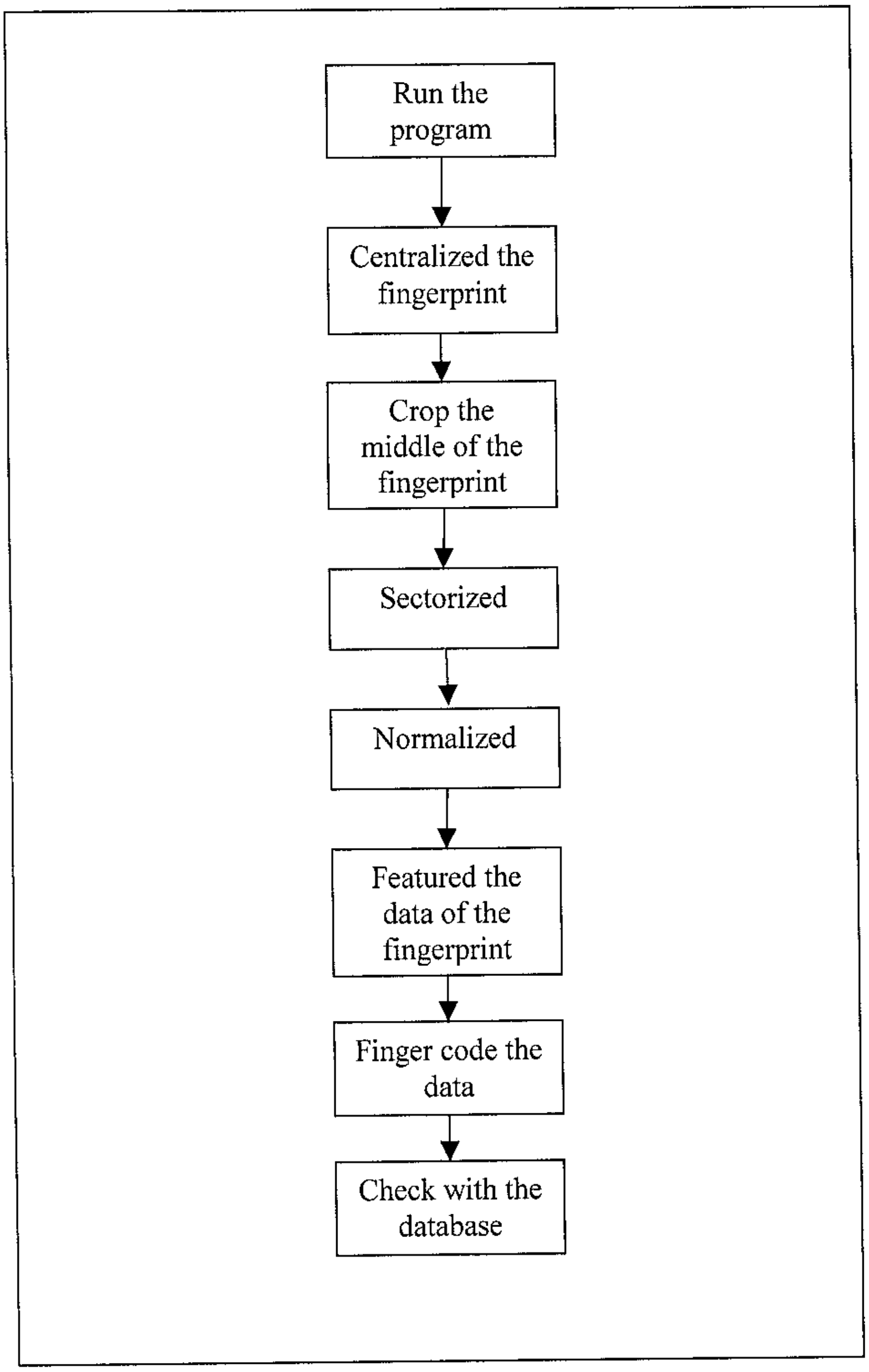

Run the program
Centralized the fingerprint
Crop the middle of the fingerprint
Sectorized
Normalized
Featured the data of the fingerprint
Finger code the data
Check with the database

Figure 3.3: Flow chart showing the flow of the program.

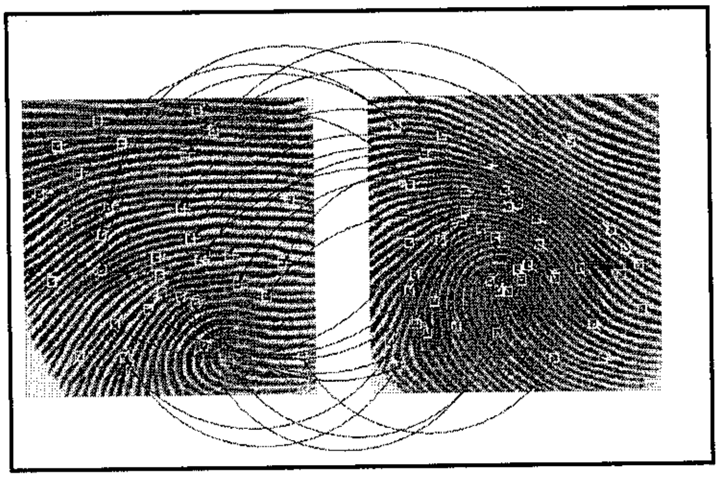
CHAPTER 4

DISCUSSION

4.1 Fingerprint Matching

Among all the biometric techniques, fingerprint-based identification is the oldest method which has been successfully used in numerous applications. Everyone is known to have unique, immutable fingerprints. A fingerprint is made of a series of ridges and furrows on the surface of the finger. The uniqueness of a fingerprint can be determined by the pattern of ridges and furrows as well as the minutiae points. Minutiae points are local ridge characteristics that occur at either a ridge bifurcation or a ridge ending.

Fingerprint matching techniques can be placed into two categories: minutae-based and correlation based. Minutiae-based techniques first find minutiae points and then map their relative placement on the finger. However, there are some difficulties when using this approach. It is difficult to extract the minutiae points accurately when the fingerprint is of low quality. Also this method does not take into account the global pattern of ridges and furrows. The correlation-based method is able to overcome some of the difficulties of the minutiae-based approach. However, it has some of its own shortcomings. Correlation-based techniques require the precise location of a registration point and are affected by image translation and rotation.



Fingerprint matching based on minutiae has problems in matching different sized (unregistered) minutiae patterns. Local ridge structures can not be completely characterized by minutiae. We are trying an alternate representation of fingerprints which will capture more local information and yield a fixed length code for the fingerprint. The matching will then hopefully become a relatively simple task of calculating the Euclidean distance will between the two codes.

4.2 Fingerprint Classification

Large volumes of fingerprints are collected and stored everyday in a wide range of applications including forensics, access control, and driver license registration. To reduce the search time and computational complexity, it is desirable to classify these fingerprints in an accurate and consistent manner so that the input fingerprint is required to be matched only with a subset of the fingerprints in the database.
Large volumes of fingerprints are collected and stored everyday in a wide range of applications, including forensics, access control, and driver license registration. Automatic identity recognition based on fingerprints requires that the input fingerprint be matched with a large number of fingerprints stored in a database (the FBI database currently contains more than 630 million fingerprints!). To reduce the search time and computational complexity, it is desirable to classify these fingerprints in an accurate and consistent manner such that the input fingerprint needs to be matched only with a subset of the fingerprints in the database. Fingerprint classification is a technique used to assign a fingerprint into one of the several pre-specified types already established in the literature (and used in forensic applications) which can provide an indexing mechanism. Fingerprint classification can be viewed as a coarse level matching of the fingerprints. An input fingerprint is first matched to one of the pre-specified types and then it is compared to a subset of the database corresponding to that fingerprint type. To increase the search efficiency, the fingerprint classification algorithm can classify a fingerprint into more than one class. For example, if the fingerprint database is binned into five classes, and a fingerprint classifier outputs two classes (primary and

secondary) with high accuracy, then the identification system will only need to search two of the five bins, thus decreasing the search space 2.5 folds. Continuous classification of fingerprints is also very attractive for indexing where fingerprints are not partitioned in non-overlapping classes, but each fingerprint is characterized with a numerical vector summarizing its main features. The continuous features obtained are used for indexing fingerprints through spatial data structures and for retrieving fingerprints by means of spatial queries. In this report, I have concentrated on an exclusive fingerprint classification and classify fingerprints into five distinct classes, namely, *whorl* (*W*), *right loop* (*R*), *left loop* (*L*), *arch* (*A*), and *tented arch* (*T*) (Figure 4.2). The five classes are chosen based on the classes identified by the National Institute of Standards and Technology (NIST) to benchmark automatic fingerprint classification algorithms. The natural proportion of occurrence of these five major classes of fingerprints is 0:3252, 0:3648, 0:1703, 0:0616, and 0:0779 for whorl, right loop, left loop, arch, and tented arch, respectively. There are two main types of features in a fingerprint: (*i*) global ridge and furrow structures which form special patterns in the central region of the fingerprint, and (*ii*) local ridge and furrow minute details (see Figure 4.3). A fingerprint is classified based on only the first type of features and is uniquely identified based on the second type of features (ridge endings and bifurcations, also known as minutiae). See Figure 4.3 for examples of ridges, minutiae, orientation field and singular points in a fingerprint image.

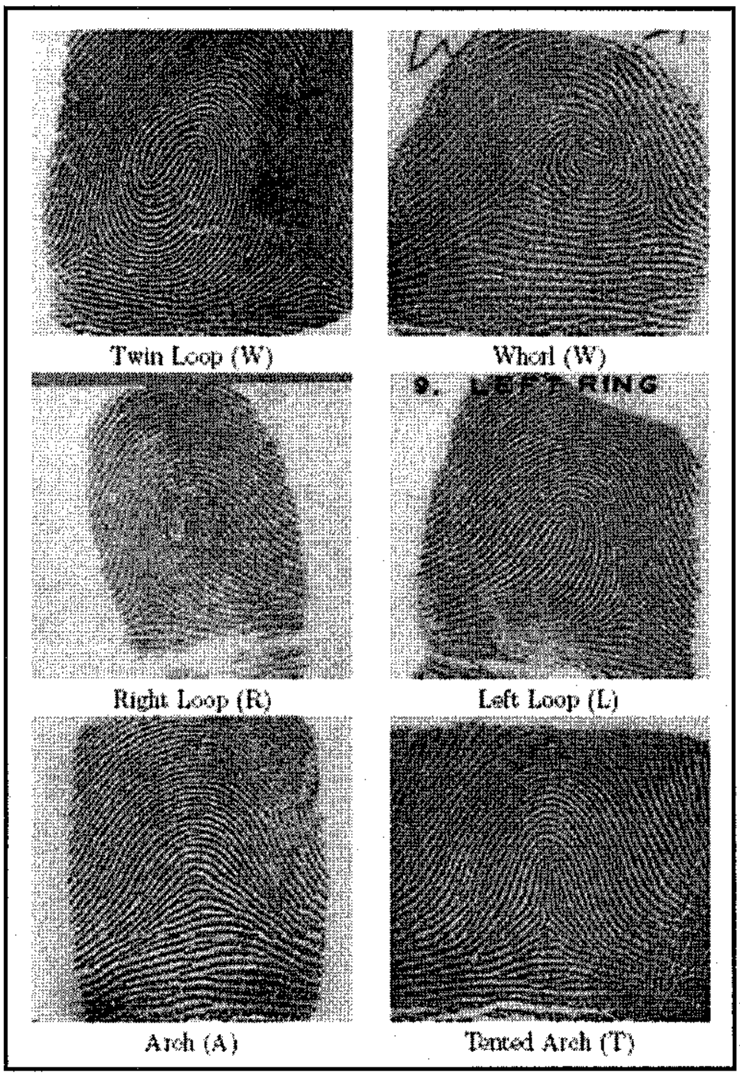


Figure 4.2: Six major fingerprint classes. Twin loop images are labeled as whorl in the NIST-4 database.

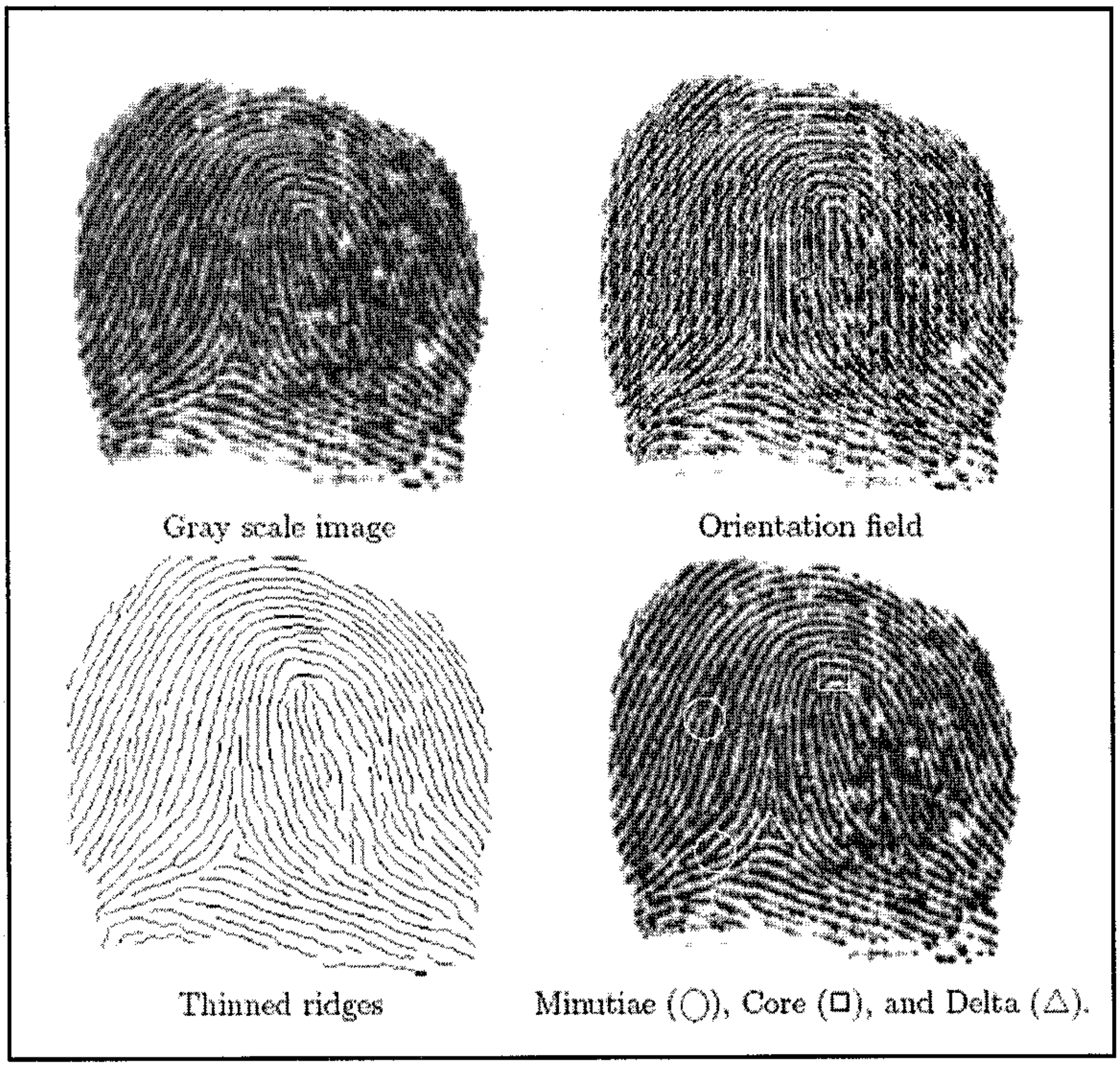


Figure 4.3: Orientation field, thinned ridges, minutiae, and singular points.

4.3 Fingerprint Image Enhancement

A critical step in automatic fingerprint matching is to automatically and reliably extract minutiae from the input fingerprint images. However, the performance o f a minutiae extraction algorithm relies heavily on the quality of the input fingerprint images. In order to ensure that the performance of an automatic fingerprint identification/verification system will be robust with respect to the quality of the fingerprint images, it is essential to incorporate a fingerprint enhancement algorithm in the minutiae extraction module.

4.3.1 Methods

The most common method of generating a template emulates the traditional method of matching "minutiae"— bifurcations, divergences, enclosures, endings, and valleys in the ridge pattern. Each minutia is described by a set of numeric variables. Approximately 80 percent of biometric vendors use minutiae in some fashion. Other methods include using "traditional" pattern matching techniques and using moiré fringe patterns. The fingerprint has one of the largest biometric templates, ranging from 250 bytes (minutiae) to over 1,000 bytes (pattern matching). Note that the template holds only particular data about the fingerprint (the minutiae), not the image of the fingerprint itself, nor can the full fingerprint be reconstructed from the template.

Minutiae points are referred to as 'points' because the fingerprint scanner assigns locations (points) to the minutiae using X, Y and directional variables. Minutiae points are and can be made up of the following characteristics:

1. Bifurcation

    - the point at which a ridge splits into multiple ridges, called branches

2. Divergence

    - this is the point where parallel ridges either spread apart or come together

3. Enclosure

    - occurs when a ridge splits into two branches and then comes together again shortly thereafter

4. Ending

    - occurs when a ridge terminates

5. Valley

    - spaces or gaps that are on either side of a ridge

Other methods of identifying a person's fingerprint include counting the number of ridges between points, processing the fingerprint image and recording the print's sound waves.

Fingerprint imaging technology is based on two electronic capturing methods: optical and capacitive. Optical fingerprint technologies require the user to place his or her finger on a glass substrate at which point an internal light source from the fingerprint device is projected onto the fingerprint. The image is then captured by a charge-coupled device (CCD). Optical methods have been used extensively and have been in existence for the past decade. They are proven but are on the expensive side and are not always reliable due to environmental conditions. A build up of dirt, grime, and oil from one's finger can leave a "ghost" image which is referred to as a "latent image".

On the other hand, capacitive imaging looks to make fingerprint imaging available to the masses by making fingerprint imaging devices (hardware) more compact in size, less expensive, and more reliable. Capacitive systems analyze one's fingerprint by detecting the electrical field around the fingerprint using a sensor chip and an array of circuits.

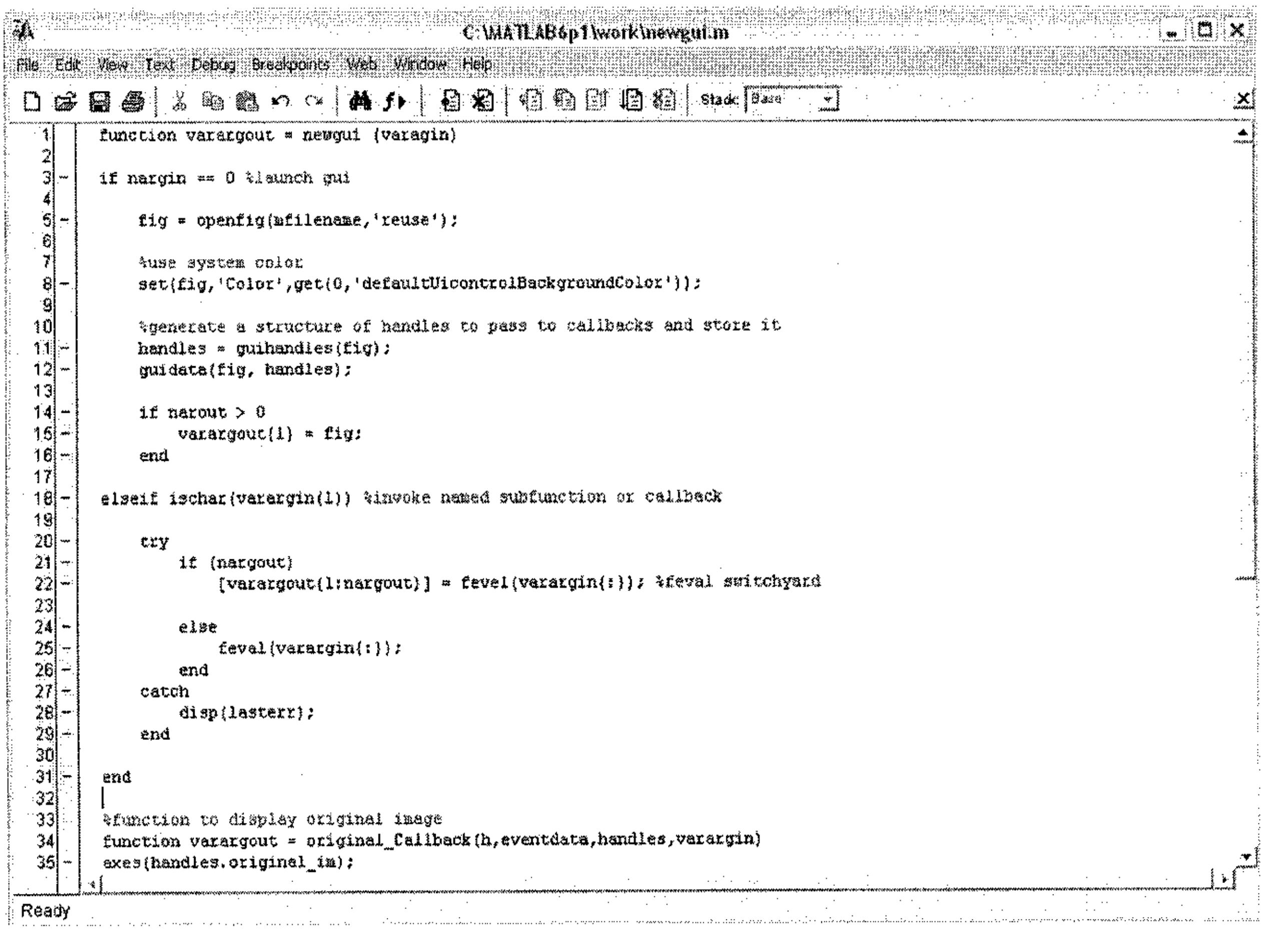
When a person's fingerprint is initially captured, a 'template' is constructed and stored in a data storage system or database. This 'template' is then used to compare against a person's fingerprint for each subsequent time he or she scans their finger. The fingerprint requires one of the largest data templates in the biometric field. The finger data template can range anywhere from several hundred bytes to over 1,000 bytes depending upon the level of security that is required and the method that is used to scan one's fingerprint.

After the first half or Final Year Project Part 1 has finished, the second part of my FYP project is to produce the GUI, the Graphic User Interface. This where everything will be displayed: the original image, grayscale image, and the results from the database or fingerprint matching.

4.4 Graphic User Interface (GUI)

The GUI is has its own main role in any program. The purpose of GUI is to show the user where or something that is visible to their eyes.

This is the proposed GUI, the rectangle shape. But there is some problem in this source codes. There are lines that are unrecognizable by the MATLAB.


C:\MATLAB6p1\work\newgui.m
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
Ready

```
function varargout = newgui (varagin)

if nargin == 0 %launch gui

    fig = openfig(mfilename,'reuse');

    %use system color
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

    %generate a structure of handles to pass to callbacks and store it
    handles = guihandles(fig);
    guidata(fig, handles);

    if narout > 0
        varargout(1) = fig;
    end

elseif ischar(varargin(1)) %invoke named subfunction or callback

    try
        if (nargout)
            [varargout(1:nargout)] = fevel(varargin{:}); %feval switchyard

        else
            feval(varargin{:});
        end
    catch
        disp(lasterr);
    end

end

%function to display original image
function varargout = original_Callback(h,eventdata,handles,varargin)
axes(handles.original_im);
```

Figure 4.4: The source code of the GUI at first attempt.

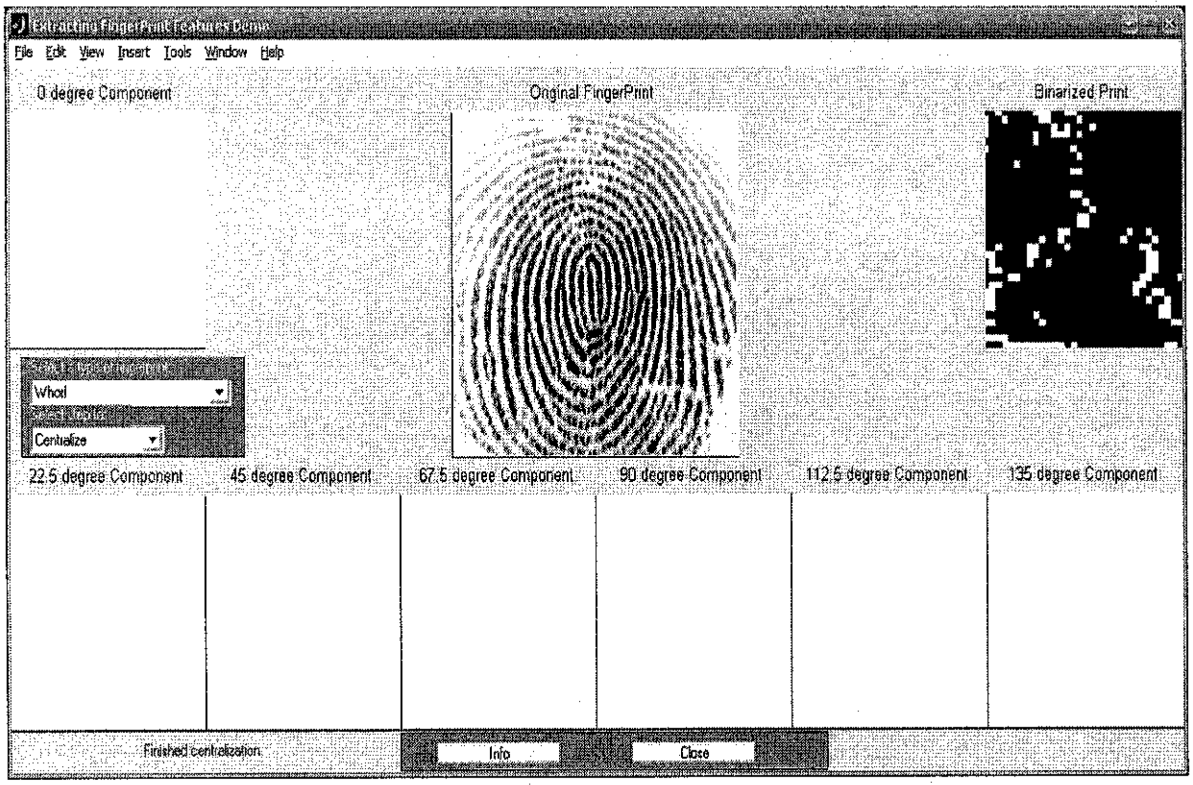


Figure 4.5: The GUI of the Fingerprint Recognition.

4.5 Displaying the Original Image

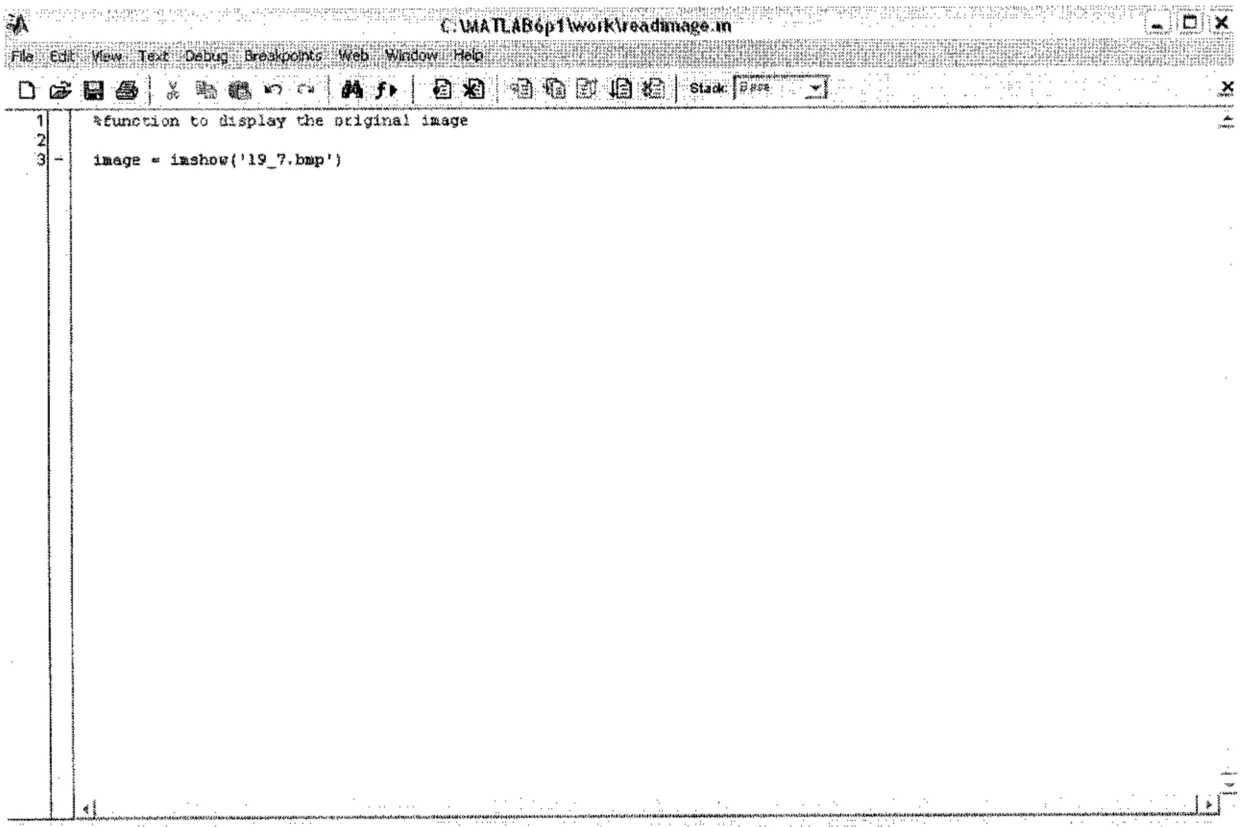
In this part, the original image of the thumbprint will be displayed using the:

imread command syntax.

The syntax should be written as follows:

image = imread ('filename.extension')

This is what is written in the MATLAB FILE.


C:\MATLAB6p1\work\readimage.m
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
Ready

```
%function to display the original image

image = imshow('19_7.bmp')
```

Figure 4.6: the code in MATLAB which is used to call the image and display the image.

This should be the result of the above action:

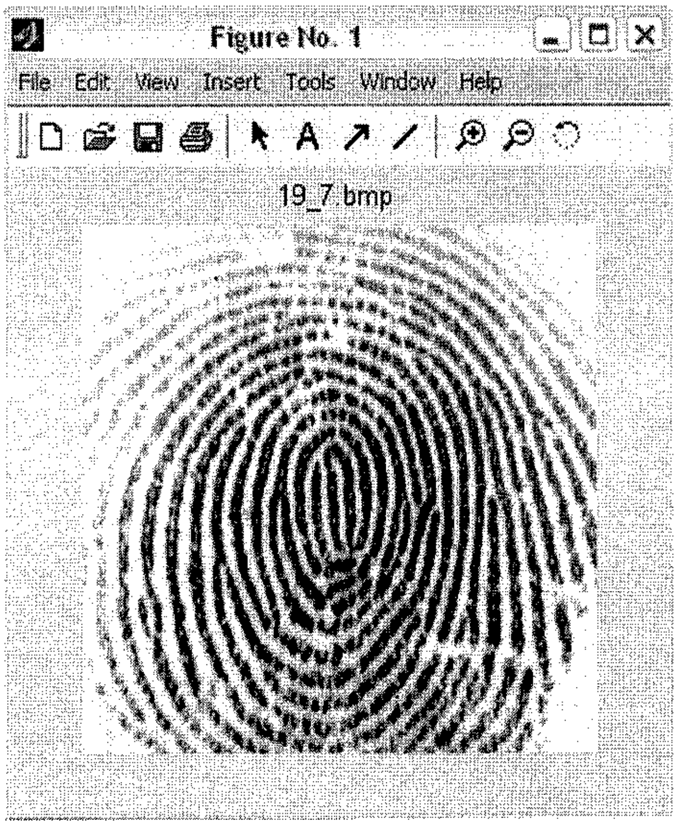
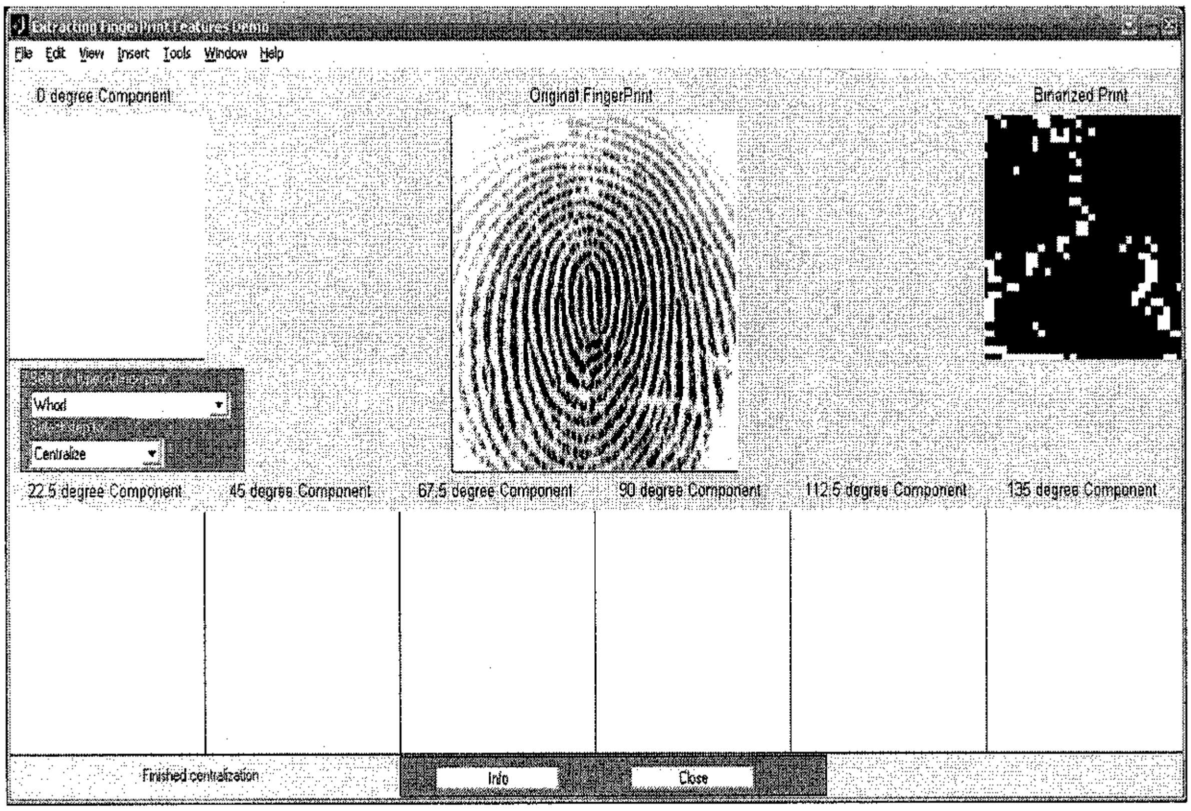


Figure 4.7: The original image is displayed with the MATLAB image viewer.



Whorl

Centralize

Finished centralization

Figure 4.8: The original image of the fingerprint is display at the center of the GUI.

4.6 Centralizing Function

A function which accept an input image and determines the coordinates of the core point. The core point is determinated by complex filtering. The region of interest is determinated fixing a minimum threshold value for the variance. Input image is divided into non-overlapping blocks and only blocks with a variance smaller than this threshold value are considered background. The logical matrix (associated to the region of interest) is first closed (Matlab function imclose), then eroded (Matlab function imerode) with two given structuring elements. The image is "mirrored" before convolution with complex filter, and then it is re-cropped to its original sizes.

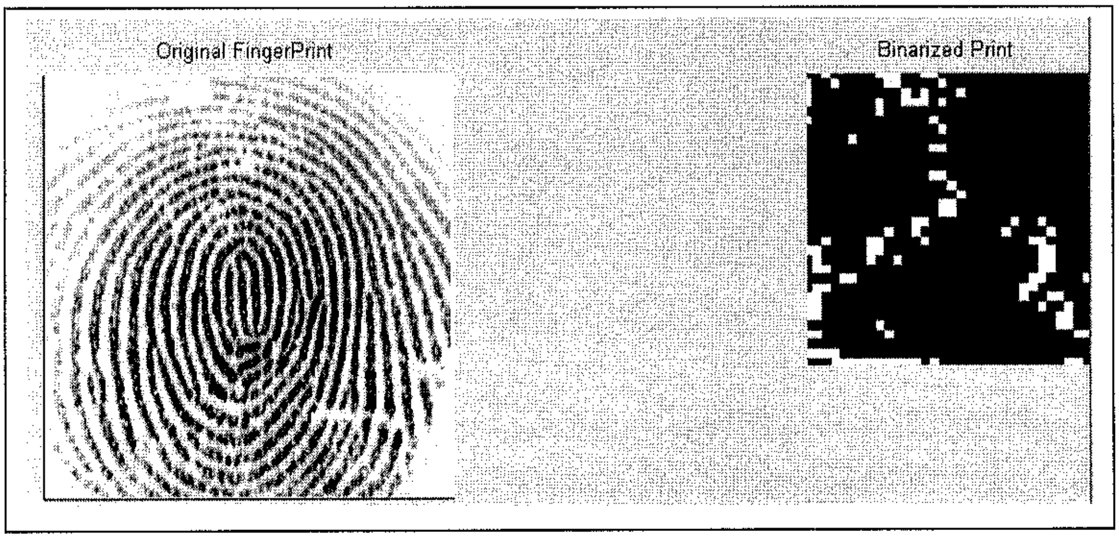


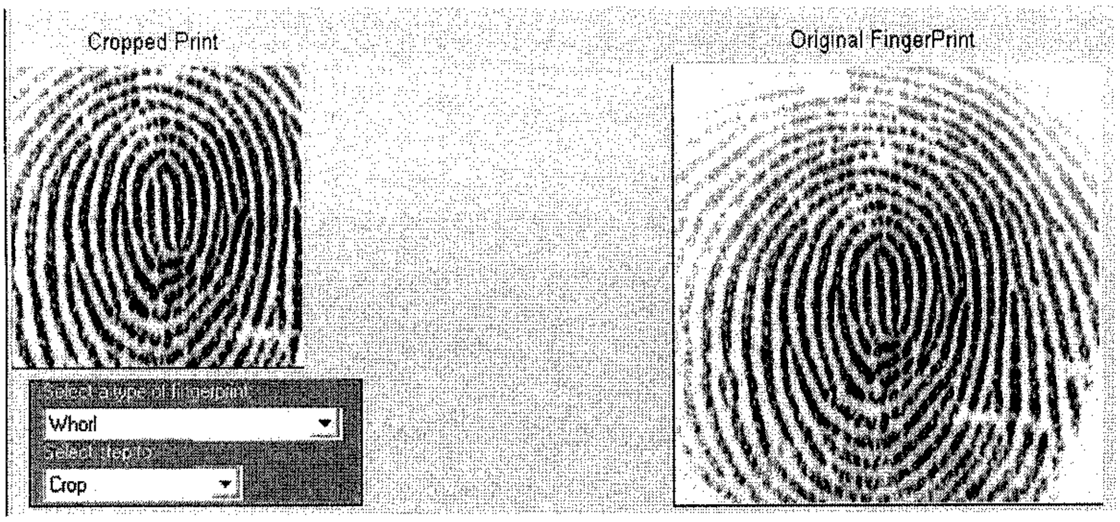
Figure 4.9: The centralizing function displays the image and binarized it.

The main usage for this function is to centralize everything that is not centralized during scanning process to capture the fingerprint image. The function will automatically determine the centre of the fingerprint and determined the core as explained in the above paragraph. The portion of the source code use for this program is as shown in the next page.

```
function Centralize(DemoFig)

load 'informations.dat' -mat

if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Centralizing..., please wait !!!');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);
fingerprint = fingerprint*graylevmax;

[BinarizedPrint,XofCenter,YofCenter] = centralizing(fingerprint,0);

set(get(ud.hComponent8Axes, 'title'), 'string', 'Binarized Print');
set(ud.hComponent8Image, 'Cdata', BinarizedPrint);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished centralization');
ud.OriginalImageIsStale = 0;
set(DemoFig, 'UserData', ud);
drawnow
```

4.7 Crop

For the crop function, it is used to crop the original fingerprint and make it to a smaller image. The image that is crop is basically and usually the center of the thumbprint or fingerprint where the calculation will be made to determined the owner of the fingerprint. The basic identification will be determined by minuate, ridges and valleys of the fingerprint.



Select a type of fingerprint
Whorl
Select step to
Crop

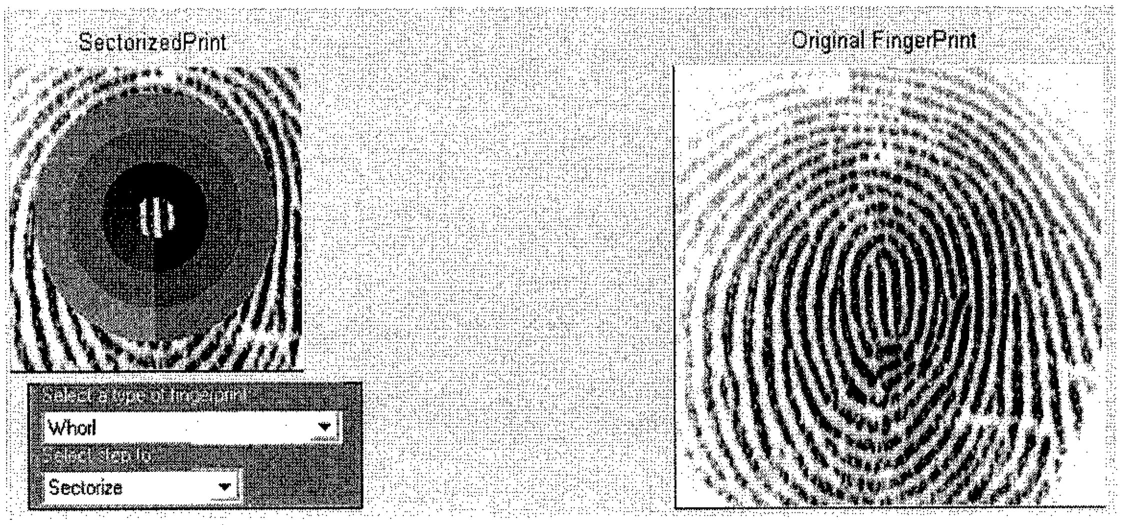
Figure 4.10: The crop image of the centralized fingerprint. The crop function will only extract the fingerprint from the centre to a certain pixels.

The portion of the codes used for this section is shown in the next page.

```
function Crop(DemoFig)
%
load 'informations.dat' -mat
if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Cropping..., please wait !!!');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

fingerprint = fingerprint*graylevmax;

[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);

CroppedPrint = double(CroppedPrint)/graylevmax;
set(get(ud.hComponent1Axes, 'title'), 'string', 'Cropped Print');
set(ud.hComponent1Image, 'Cdata', CroppedPrint);
set(get(ud.hComponent8Axes, 'title'), 'string', 'Binarized Print');
set(ud.hComponent8Image, 'Cdata', BinarizedPrint);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished Crop');
ud.Component1ImageIsStale = 0;
set(DemoFig, 'UserData', ud);
drawnow
```

4.8 Sectorized

Sectorization is used to determine the sector where we want to analyze. The purpose of this is to make sure that the only specified area of interest will be analyzed. The unwanted sectors have already been cropped earlier in the crop section.



Select a type of fingerprint
Whorl
Select step to
Sectorize

Figure 4.11: The sectorized print is show is the smaller figure. The circle indicating that the the area is sectored and will be analyzed.

The portion of the codes used for this section is shown in the next page.

```
function Sectorize(DemoFig)

%
load 'informations.dat' -mat


if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Sectorizing..., please wait !!!');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);


fingerprint = fingerprint*graylevmax;


[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
for ( i=1:1:175*175)
   tmp=CroppedPrint(i);
   CroppedPrint(i)=whichsector(i);
   if (CroppedPrint(i)==36 | CroppedPrint(i)==37)
      CroppedPrint(i)=tmp/graylevmax;
   else
      CroppedPrint(i)=CroppedPrint(i)/64;
   end

end
```

```
set(get(ud.hComponent1Axes, 'title'), 'string', 'SectorizedPrint');
set(ud.hComponent1Image, 'Cdata', CroppedPrint);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished Sectorization');
ud.Component1ImageIsStale = 0;
set(DemoFig, 'UserData', ud);
drawnow
```

4.9 Normalized

Normalizing means that the fingerprint is being sectorized than transfer the image to grayscale. Then the fingerprint is being enhanced for a clearer view of the pattern for the upcoming calculation of the program for identification purpose.

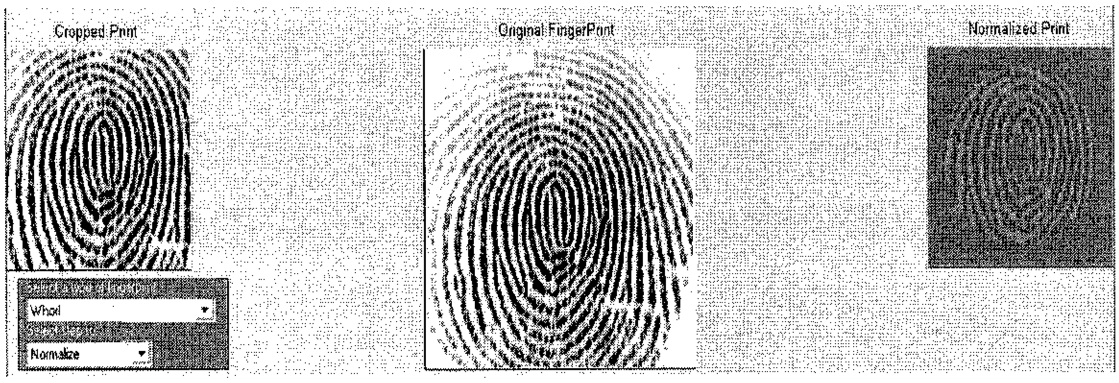


Figure 4.12: The small picture shows the normalized fingerprint after normalization.

The portion of the codes used for this section is shown in the next page.

```
function Normalize(DemoFig)
%
load 'informations.dat' -mat

if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Normalizing..., please wait !!!');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

fingerprint = fingerprint*graylevmax;


[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
[NormalizedPrint,vector] = sector_norm( CroppedPrint , 0 , 0);

CroppedPrint = double(CroppedPrint)/graylevmax;
NormalizedPrint = double(NormalizedPrint)/100;
set(get(ud.hComponent1Axes, 'title'), 'string', 'Cropped Print');
set(ud.hComponent1Image, 'Cdata', CroppedPrint);
set(get(ud.hComponent8Axes, 'title'), 'string', 'Normalized Print');
set(ud.hComponent8Image, 'Cdata', NormalizedPrint);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished normalization');
ud.Component1ImageIsStale = 0;
set(DemoFig, 'UserData', ud);
drawnow
```

4.10 Featured Data

For this part, the feature or the data from the fingerprint is being extracted from the fingerprint. The six (6) crop images below shown are the convoluted data with certain angle.

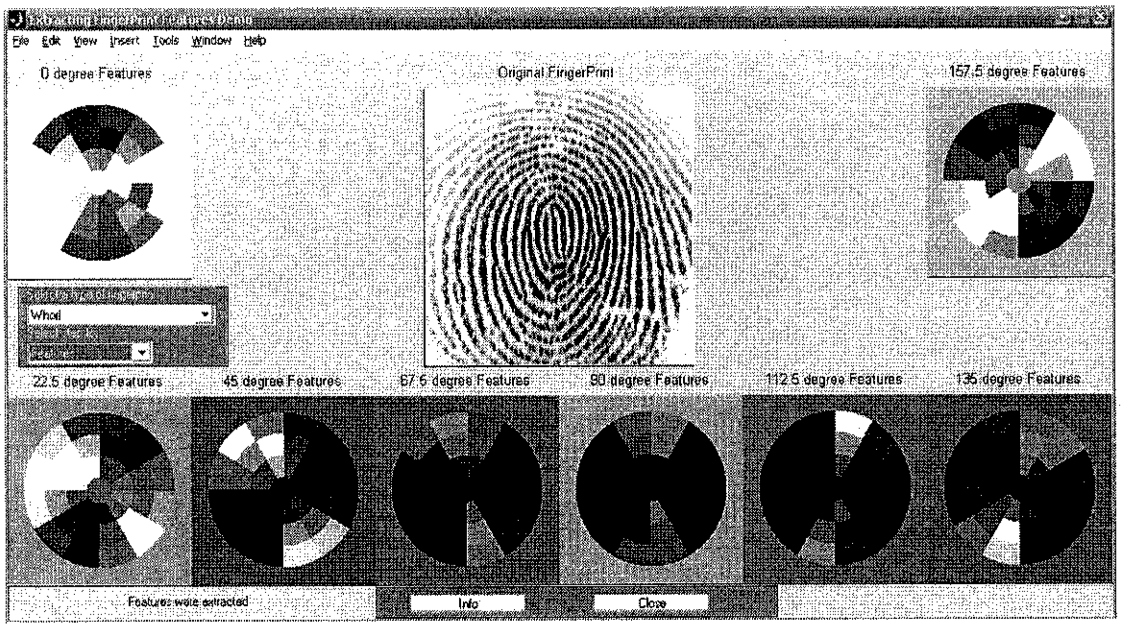


Figure 4.13: The feature of the fingerprint is being extracted.

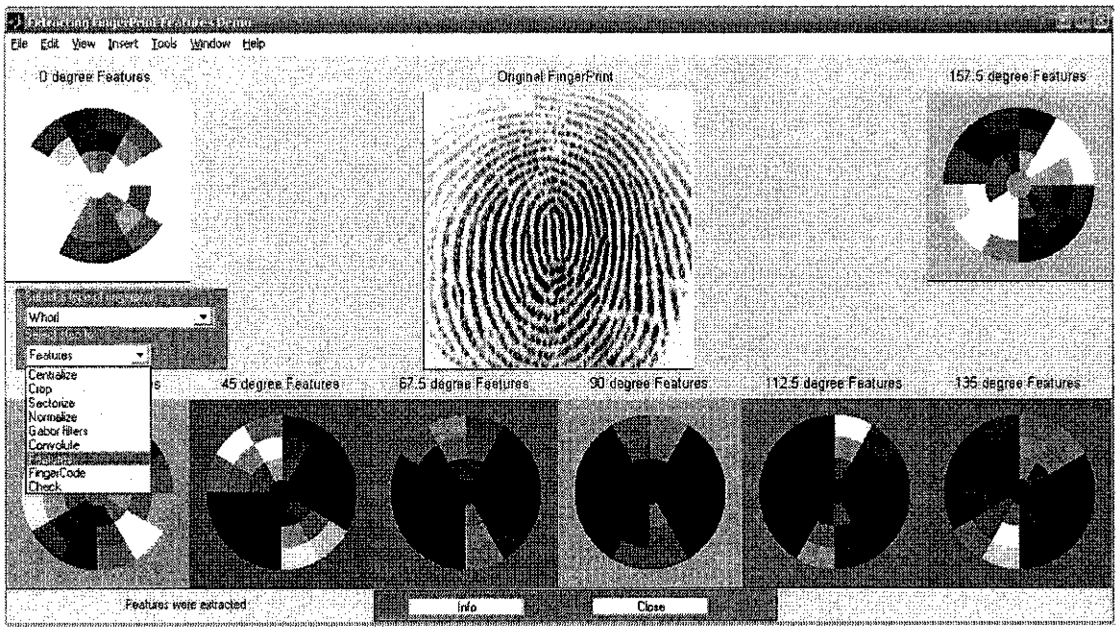
The portion of the codes used for this section is shown in the next page.

```
function Features(DemoFig)
%
load 'informations.dat' -mat

if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Convoluting with eight Gabor filters in process...');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

fingerprint = fingerprint*graylevmax;

N=175;
num_disk=8;

[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
[NormalizedPrint,vector]=sector_norm(CroppedPrint,0,1);

for (angle=0:1:num_disk-1)

    gabor=gabor2d_sub(angle,num_disk);
    z2=gabor;
    z1=NormalizedPrint;
    z1x=size(z1,1);
    z1y=size(z1,2);
    z2x=size(z2,1);
    z2y=size(z2,2);
```

```
  ComponentPrint=real(ifft2(fft2(z1,z1x+z2x-1,z1y+z2y-1).*fft2(z2,z1x+z2x-
1,z1y+z2y-1)));
  px=((z2x-1)+mod((z2x-1),2))/2;
  py=((z2y-1)+mod((z2y-1),2))/2;
  ComponentPrint=ComponentPrint(px+1:px+z1x,py+1:py+z1y);

  [disk,vector]=sector_norm(ComponentPrint,1,0);

  img = double(ComponentPrint)/graylevmax;
  img1 = double(disk)/51200;
  switch angle<8
    case (angle==0),
      set(get(ud.hComponent1Axes, 'title'), 'string', '0 degree Features');
      set(ud.hComponent1Image, 'Cdata', img1);
    case (angle==1),
      set(get(ud.hComponent2Axes, 'title'), 'string', '22.5 degree Features');
      set(ud.hComponent2Image, 'Cdata', img1);
    case (angle==2),
      set(get(ud.hComponent3Axes, 'title'), 'string', '45 degree Features');
      set(ud.hComponent3Image, 'Cdata', img1);
    case (angle==3),
      set(get(ud.hComponent4Axes, 'title'), 'string', '67.5 degree Features');
      set(ud.hComponent4Image, 'Cdata', img1);
    case (angle==4),
      set(get(ud.hComponent5Axes, 'title'), 'string', '90 degree Features');
      set(ud.hComponent5Image, 'Cdata', img1);
    case (angle==5),
      set(get(ud.hComponent6Axes, 'title'), 'string', '112.5 degree Features');
      set(ud.hComponent6Image, 'Cdata', img1);
    case (angle==6),
      set(get(ud.hComponent7Axes, 'title'), 'string', '135 degree Features');
      set(ud.hComponent7Image, 'Cdata', img1);
    case (angle==7),
      set(get(ud.hComponent8Axes, 'title'), 'string', '157.5 degree Features');
      set(ud.hComponent8Image, 'Cdata', img1);
    otherwise
      error('Nothing !');
  end

end

set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Features were extracted');
set(DemoFig, 'UserData', ud);
drawnow
```

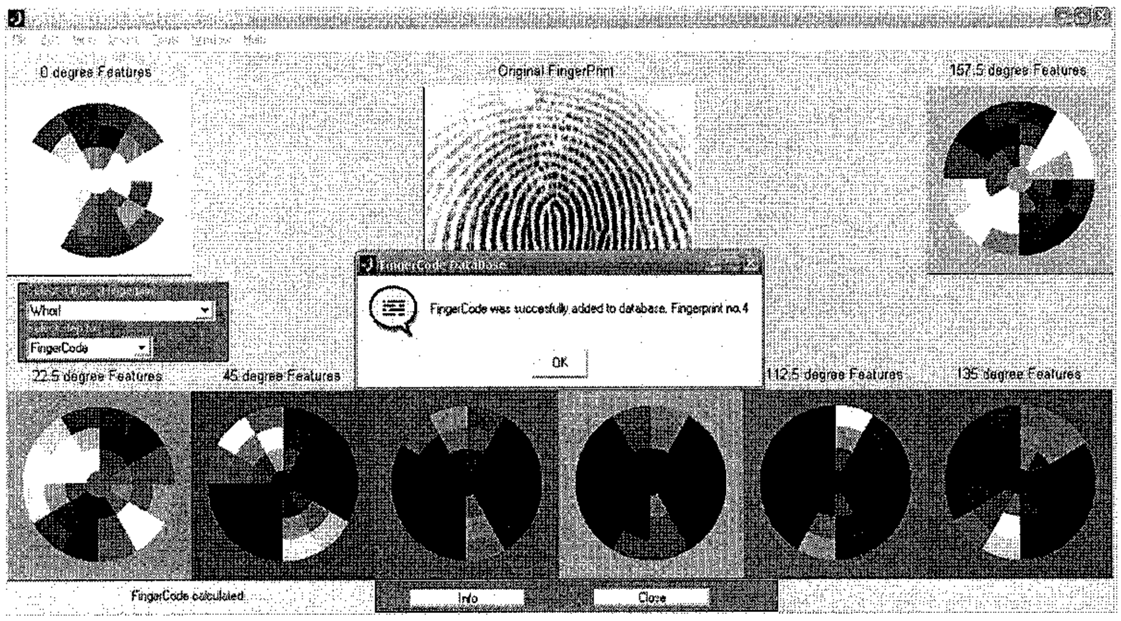
4.11 Finger Code

This is where the database of the fingerprint(s) is saved. The function used to make the database will save the fingerprint data in *.DAT format.



Features were extracted

Figure 4.14: The feature of the fingerprint is being saved in database.



FingerCode was succesfully added to database. Fingerprint no.4

FingerCode calculated

Figure 4.15: The fingerprint data is successfully saved in database as *.DAT format.

4.12 Checking

For this part, the program will check the database and look for the identical or nearly same fingerprint as been added in the database earlier.

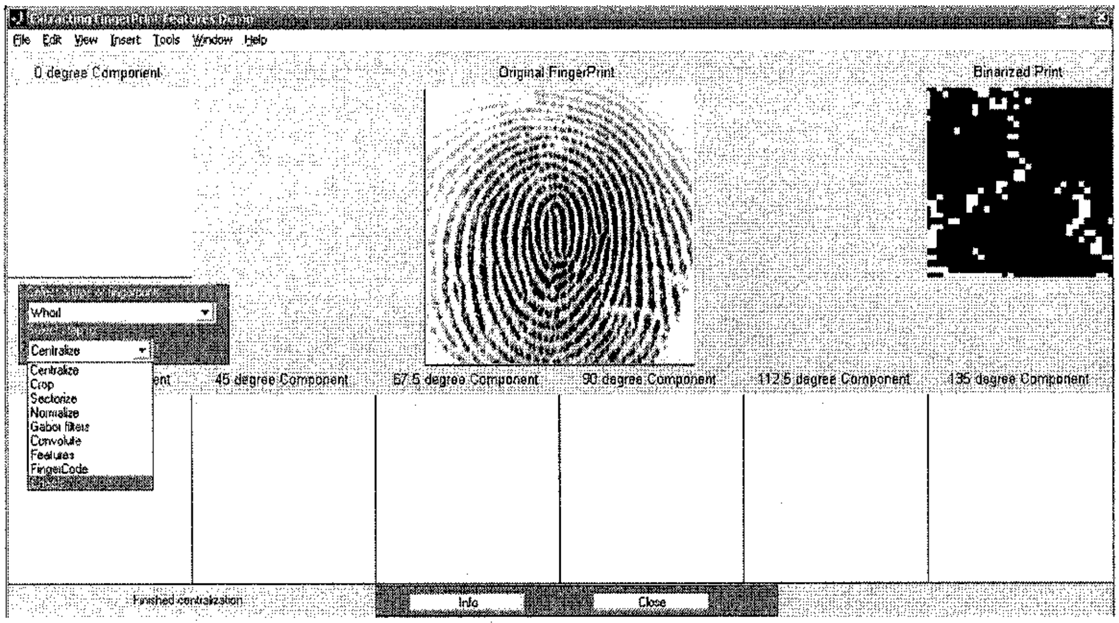
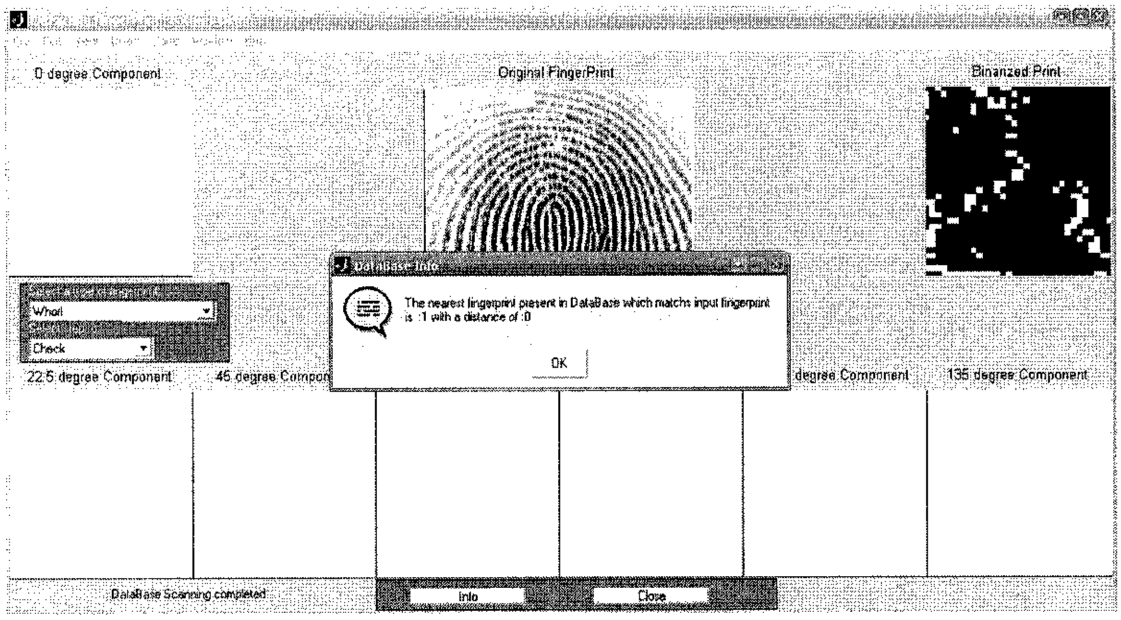


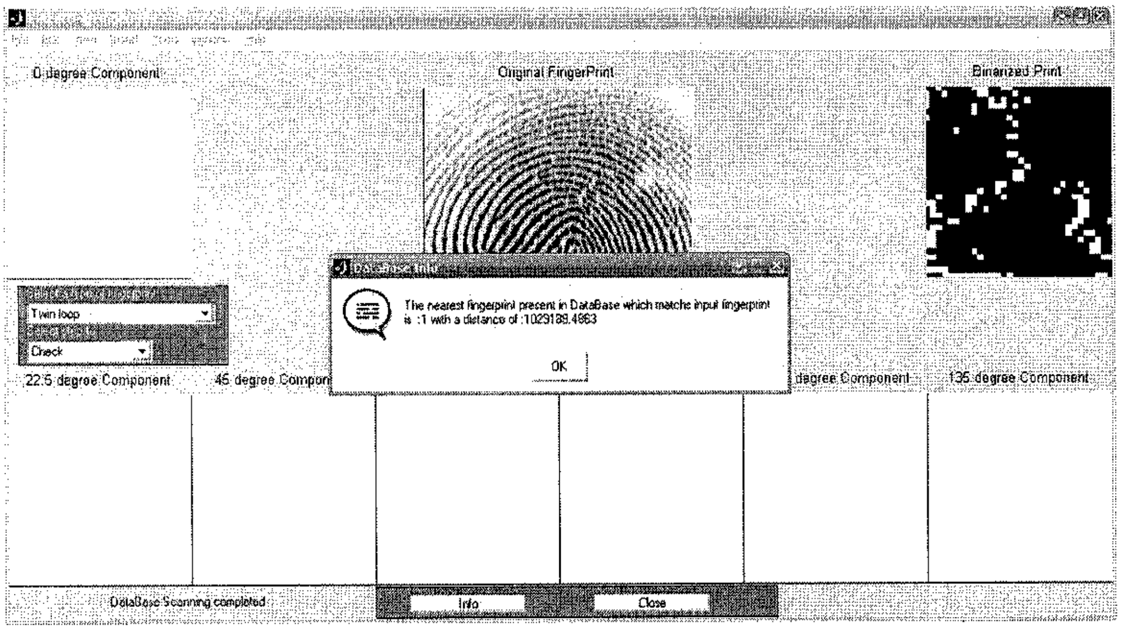
Figure 4.16: The tab to select the checking of the fingerprint.



The nearest fingerprint present in DataBase which matchs input fingerprint is :1 with a distance of :0

Figure 4.17: The result of fingerprint scanning.

If the fingerprint is being check and have the database in the program, the MATLAB will automatically display the result of it. The result shown that the fingerprint that is check matches with the database no 1 with the distance 0. That means that the fingerprint is the same with the database and exactly 100% match. But is the database is not same or exactly like the database have, the result will display like this:



The nearest fingerprint precent in DataBase which matche input fingerprint is :1 with a distance of :1029189.4863

Figure 4.18: The result of database scanning that is not the same or exist in the database.

If the database is not exist or the fingerprint is not likely the same or nearly, the display of the result will be like matching fingerprint 1 but the distance is 1029289.*, means that the fingerprint is not exactly or nearly match in the database.

CHAPTER 5

CONCLUSION

For t he c onclusion, t he o bjective o f t his F inal Y ear Design Project is reached. The objective is to design or build a program that can check the fingerprint whether it is match, exist or not in the database.

Although the objective has been reached, the program is still having flaws. The program can be modified more to ensure that the program will run smoothly in the future. Some of the function are not well organized, programmed. For upcoming future, hopefully there will be a student or students that are willing to take this project and enhance it furthermore in this BIOMETRICS TECHNOLOGY.

BIBLIOGRAPHY

A. C. Bovik, M. Clark, and W. S. Geisler, "Multichannel Texture Analysis Using Localized Spatial Filters," *IEEE Trans. Pattern Anal. and Machine Intell.*, Vol. 12, No. 1, pp. 55-73, January 1990.

Access the Web with your face.
http://www.miros.com/web access demo page.htm.

A. K. Hrechak and J. A. McHugh, "Automated Fingerprint Recognition Using Structural Matching," *Pattern Recognition*, Vol. 23, pp. 893-904, 1990.

A. K. Jain, A. Ross, and S. Prabhakar, "Fingerprint Matching Using Minutiae and Texture Features", to appear in the *International Conference on Image Processing (ICIP)*, Greece, October 7-10, 2001.

A. K. Jain and D. Zongker, "Feature Selection: Evaluation, Application, and Small Sample Performance", *IEEE Trans. Pattern Anal. Machine Intell.*, Vol. 19, No. 2, pp. 153-158, 1997.

A. K. Jain, L. Hong, S. Pankanti, and Ruud Bolle, "An Identity Authentication System Using Fingerprints," *Proceedings of the IEEE*, Vol. 85, No. 9, pp. 1365-1388, 1997.

A. K. Jain, L. Hong, and R. Bolle, "On-line Fingerprint Verification," *IEEE Trans. Pattern Anal. and Machine Intell.*, Vol. 19, No. 4, pp. 302-314, 1997.

A. K. Jain, R. M. Bolle, and S. Pankanti (editors), *Biometrics: Personal Identification in a Networked Society*, Kluwer Academic Publishers, 1999.

A. K. Jain, S. Prabhakar, and A. Ross, "Fingerprint Matching: Data Acquisition and Performance Evaluation", *MSU Technical Report TR99-14*, 1999.

American National Standard for Information Systems – Data Format for the Interchange of Fingerprint Information, Doc No. ANSINIST-CSL 1-1993, American National Standards Institute, New York, 1993.

A. P. Fitz and R. J. Green, "Fingerprint Classification Using Hexagonal Fast Fourier Transform," *Pattern Recognition*, Vol. 29, No. 10, pp. 1587-1597, 1996.

B. G. Sherlock and D. M. Monro, "A Model for Interpreting Fingerprint Topology," *Pattern Recognition*, Vol. 26, No. 7, pp. 1047-1055, 1993.
[35] B. Moayer and K. Fu, "A Tree System Approach for Fingerprint Pattern Recognition," *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. 8 no. 3, pp. 376-388, 1986.

D. B. G. Sherlock, D. M. Monro, and K. Millard, "Fingerprint Enhancement by Directional Fourier Filtering," *Proc. Inst. Elect. Eng. Visual Image Signal Processing*, Vol. 141, No. 2, pp. 87-94, 1994.

Digital Biometrics, Inc., Biometric Identification Products. Available at: http://www.digitalbiometrics.com/

Digital Persona, Inc., Fingerprint-based Biometric Authentication. http://www.digitalpersona.com/

E. Newham, *The Biometric Report*. New York: SBJ Services, 1995. http://www.sjb.co.uk/.

E. P. Richards, "Phenotype vs. Genotype: Why Identical Twins Have Different Fingerprints?" http://www.forensic-evidence.com/site/ID Twins.html.

Federal Bureau of Investigation. www.fbi.gov

Identix Incorporated. www.identix.com

REFERENCES

1. AUTOMATED BIOMETRICS Technologies and Systems; David D Zhang – KLUMWER ACADEMIC PUBLISHERS

2. MATLAB SOFTWARE Release 12, HELP FILE

THE PROGRAM

```
function fpextractdemo(action, varargin)

if nargin<1,
   action='InitializeFPEXTRACTDEMO';
end;

feval(action,varargin{:})
return;

%%%
%%%  Sub-function - InitializeFPEXTRACTDEMO
%%%

function InitializeFPEXTRACTDEMO()

% If fpextractdemo is already running, bring it to the foreground
h = findobj(allchild(0), 'tag', 'Extracting FingerPrint Features Demo');
if ~isempty(h)
   figure(h(1))
   return
end

screenD = get(0, 'ScreenDepth');
if screenD>8
   grayres=256;
else
   grayres=128;
end


FpextractDemoFig = figure( ...
   'Name','Extracting FingerPrint Features Demo', ...
   'NumberTitle','off', 'HandleVisibility', 'on', ...
   'tag', 'Extracting FingerPrint Features Demo', ...
   'Visible','off', 'Resize', 'off',...
   'BusyAction','Queue','Interruptible','off', ...
   'Color', [.8 .8 .8], ...
   'IntegerHandle', 'off', ...
   'Colormap', gray(grayres));

figpos = get(FpextractDemoFig, 'position');
figpos(3:4) = [1050 525];
% Adjust the size of the figure window
horizDecorations = 10;  % resize controls, etc.
```

```
vertDecorations = 45;   % title bar, etc.
screenSize = get(0,'ScreenSize');

dx = screenSize(3) - figpos(1) - figpos(3) - horizDecorations;
dy = screenSize(4) - figpos(2) - figpos(4) - vertDecorations;
if (dx < 0)
   figpos(1) = max(5,figpos(1) + dx);
end
if (dy < 0)
   figpos(2) = max(5,figpos(2) + dy);
end
set(FpextractDemoFig, 'position', figpos);

rows = figpos(4);
cols = figpos(3);

% Colors
bgcolor = [0.45 0.45 0.45];  % Background color for frames
wdcolor = [.8 .8 .8];  % Window color
fgcolor = [1 1 1];  % For text

hs = (cols-(6*175)) / 5;       % Horizantal Spacing
vs = (rows)/8;                 % Vertical Spacing

%===========================================
% Parameters for all buttons and menus

Std.Interruptible = 'off';
Std.BusyAction = 'queue';

% Defaults for image axes
Ax = Std;
Ax.Units = 'Pixels';
Ax.Parent = FpextractDemoFig;
Ax.ydir = 'reverse';
Ax.XLim = [.5 128.5];
Ax.YLim = [.5 128.5];
Ax.CLim = [0 1];
Ax.XTick = [];
Ax.YTick = [];

Img = Std;
Img.CData = [];
Img.Xdata = [1 128];
Img.Ydata = [1 128];
Img.CDataMapping = 'Scaled';
Img.Erasemode = 'none';

Ctl = Std;
Ctl.Units = 'Pixels';
Ctl.Parent = FpextractDemoFig;

Btn = Ctl;
Btn.Style = 'pushbutton';
Btn.Enable = 'off';

Edit = Ctl;
Edit.Style = 'edit';
Edit.HorizontalAlignment = 'right';
Edit.BackgroundColor = 'white';
Edit.ForegroundColor = 'black';

Menu = Ctl;
Menu.Style = 'Popupmenu';

Text = Ctl;
Text.Style = 'text';
Text.HorizontalAlignment = 'left';
Text.BackgroundColor = bgcolor;
Text.ForegroundColor = fgcolor;

%==================================
% 0 degree Component
ud.hComponent1Axes = axes(Ax, ...
   'Position', [0*vs/6 5*vs-vs/6 175 175]);
title('0 degree Component');
ud.hComponent1Image = image(Img, ...
   'Parent', ud.hComponent1Axes);
%==================================
% Original FingerPrint
ud.hOriginalAxes = axes(Ax, ...
   'Position', [cols/2-128 5*vs-vs/6-81 256 256]);
title('Original FingerPrint');
ud.hOriginalImage = image(Img, ...
   'Parent', ud.hOriginalAxes);
ud.OriginalImageIsStale = 1;
```

```
%================================================
% 157.5 degree Component
ud.hComponent8Axes = axes(Ax, ...
   'Position', [cols-175 5*vs-vs/6 175 175]);
title('157.5 degree Component');
ud.hComponent8Image = image(Img, ...
   'Parent', ud.hComponent8Axes);
%================================================
% 22.5 degree Component
ud.hComponent2Axes = axes(Ax, ...
   'Position', [hs vs/2 175 175]);
title('22.5 degree Component');
ud.hComponent2Image = image(Img, ...
   'Parent', ud.hComponent2Axes);
%================================================
% 45 degree Component
ud.hComponent3Axes = axes(Ax, ...
   'Position', [2*hs+1*175 vs/2 175 175]);
title('45 degree Component');
ud.hComponent3Image = image(Img, ...
   'Parent', ud.hComponent3Axes);
%================================================
% 67.5 degree Component
ud.hComponent4Axes = axes(Ax, ...
   'Position', [3*hs+2*175 vs/2 175 175]);
title('67.5 degree Component');
ud.hComponent4Image = image(Img, ...
   'Parent', ud.hComponent4Axes);
%================================================
% 90 degree Component
ud.hComponent5Axes = axes(Ax, ...
   'Position', [4*hs+3*175 vs/2 175 175]);
title('90 degree Component');
ud.hComponent5Image = image(Img, ...
   'Parent', ud.hComponent5Axes);
%================================================
% 112.5 degree Component
ud.hComponent6Axes = axes(Ax, ...
   'Position', [5*hs+4*175 vs/2 175 175]);
title('112.5 degree Component');
ud.hComponent6Image = image(Img, ...
   'Parent', ud.hComponent6Axes);
```

```
%=====================================
% 135 degree Component
ud.hComponent7Axes = axes(Ax, ...
   'Position', [6*hs+5*175 vs/2 175 175]);
title('135 degree Component');
ud.hComponent7Image = image(Img, ...
   'Parent', ud.hComponent7Axes);

%=====================================
%  The frame
ud.hControlFrame = uicontrol(Std, ...
   'Parent', FpextractDemoFig, ...
   'Style', 'Frame', ...
   'Units', 'pixels', ...
   'Position', [vs/6 5*vs-vs/6-81 200 vs+vs/8], ...
   'BackgroundColor', bgcolor);

%=========================================
% Image popup menu
ud.hImgPop = uicontrol(Menu, ...
   'Position',[vs/6+vs/8 5*vs-2*vs/3+7 180 vs/16], ...
   'String','Whorl|Twin loop|Left loop|Right loop|Other image', ...
   'Callback','fpextractdemo("LoadNewImage")');
% Text label for Image Menu Popup
uicontrol( Text, ...
   'Position',[vs/6+vs/8 5*vs-vs/6-vs/3-2 180 vs/4], ...
   'String','Select a type of fingerprint:');

%=========================================
% Extracting Step popup menu
ud.hSelectStepPop = uicontrol(Menu, ...
   'Position',[vs/6+vs/8 4*vs-7 120 vs/16], ...
   'String','Centralize|Crop|Sectorize|Normalize|Gabor
filters|Convolute|Features|FingerCode|Check', ...
   'Callback','fpextractdemo("SelectExtractingStep")');
% Text label for Extracting Step Menu Popup
uicontrol( Text, ...
   'Position',[vs/6+vs/8 4*vs-4 90 vs/4], ...
   'String','Select step to:');
```

```
%=====================================
% Frame for Info and Close
ud.hInfoCloseFrame = uicontrol(Std, ...
   'Parent', FpextractDemoFig, ...
   'Style', 'Frame', ...
   'Units', 'pixels', ...
   'Position', [3*hs+2*175 2 vs/2+2*175 vs/2-4], ...
   'BackgroundColor', bgcolor);

%=====================================
% Buttons - Info and Close
ud.hInfo=uicontrol(Btn, ...
   'Position',[3*hs+2*175+vs/2 7 vs/8+135-vs/2 vs/4], ...
   'String','Info', ...
   'Callback','helpwin fpextractdemo');

ud.hClose=uicontrol(Btn, ...
   'Position',[4*hs+3*175+vs/2 7 vs/8+135-vs/2 vs/4], ...
   'String','Close', ...
   'Callback','close(gcbf)');
%=====================================
% Status bar
ud.hStatus = uicontrol(Std, ...
   'Parent', FpextractDemoFig, ...
   'Style','text', ...
   'Units','pixels', ...
   'Position',[hs vs/8 2*175-vs/8 vs/4], ...
   'Foreground', [.8 0 0], ...
   'Background',wdcolor, ...
   'Horiz','center', ...
   'Tag', 'Status', ...
   'String','Initializing fpextractdemo...');

set(FpextractDemoFig, 'UserData', ud);
set(FpextractDemoFig, 'visible','on','HandleVisibility','callback');
set([ud.hInfo ud.hClose], 'Enable', 'on');

LoadNewImage(FpextractDemoFig);
SelectExtractingStep(FpextractDemoFig);
return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%%%  Sub-Function - LoadNewImage
%%%

function LoadNewImage(DemoFig)
% Load a new image from a mat-file

if nargin<1
   DemoFig = gcbf;
end

set(DemoFig,'Pointer','watch');
ud=get(DemoFig,'Userdata');
v = get(ud.hImgPop,{'value','String'});
name = deblank(v{2}(v{1},:));
drawnow

switch name
   case 'Right loop',
      namefile='37_7.bmp';
      [img,map]=imread(namefile);
   case 'Whorl',
      namefile='19_7.bmp';
      [img,map]=imread(namefile);
   case 'Left loop',
      namefile='37_3.bmp';
      [img,map]=imread(namefile);
   case 'Twin loop',
      namefile='37_5_2.bmp';
      [img,map]=imread(namefile);
   case 'Other image',
      [namefile,pathname]=uigetfile('*.bmp','Chose BMP GrayScale Image');
      if namefile~=0
         [img,map]=imread(strcat(pathname,namefile));
      else
         disp('   Chose a file!  ');
         [img,map]=imread('37_7.bmp');
      end
   otherwise
      error('fpextractdemo: Unknown Image Option!');
end
% If image is N x M with  mod(N,8)~=0 or mod(M,8)~=0
% input image is resized.
imgN=size(img,1);
imgM=size(img,2);
```

```
modN=mod(imgN,8);
modM=mod(imgM,8);

%-----------------------------------------
% save informations in informations.dat
if isa(img,'uint8')
   graylevmax=2^8-1;
end
if isa(img,'uint16')
   graylevmax=2^16-1;
end
if isa(img,'uint32')
   graylevmax=2^32-1;
end
save('informations.dat','graylevmax','img');

%-----------------------------------------
% resize
%-----------------------------------------
img=img(modN+1:imgN,modM+1:imgM);
%-----------------------------------------
img = double(img)/graylevmax;
set(get(ud.hOriginalAxes, 'title'), 'string', 'Original FingerPrint');
set(get(ud.hComponent1Axes, 'title'), 'string', '0 degree Component');
set(get(ud.hComponent6Axes, 'title'), 'string', '112.5 degree Component');
set(ud.hOriginalImage, 'Cdata', img);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Please select a step to process...');
return;

%==============================================
%%%
%%%  Sub-Function - SelectExtractingStep
%%%

function SelectExtractingStep(DemoFig)
% Load a step

if nargin<1
   DemoFig = gcbf;
end

set(DemoFig,'Pointer','watch');
ud=get(DemoFig,'Userdata');
v = get(ud.hSelectStepPop,{'value','String'});
name = deblank(v{2}(v{1},:));
```

```
drawnow

switch name
   case 'Centralize',
      Centralize(DemoFig);
   case 'Crop',
      Crop(DemoFig);
   case 'Sectorize',
      Sectorize(DemoFig);
   case 'Normalize',
      Normalize(DemoFig);
   case 'Gabor filters',
      Gaborfilter(DemoFig);
   case 'Convolute',
      Convolute(DemoFig);
   case 'Features',
      Features(DemoFig);
   case 'FingerCode',
      Fingercode(DemoFig);
   case 'Check',
      Check(DemoFig);
   otherwise
      error('fpextractdemo: Unknown Image Option!');
end

return;

%=====================================================================
%%%
%%%  Sub-Function - Centralize
%%%

function Centralize(DemoFig)

load 'informations.dat' -mat

if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Centralizing..., please wait !!!');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);
fingerprint = fingerprint*graylevmax;

[BinarizedPrint,XofCenter,YofCenter] = centralizing(fingerprint,0);
```

```
set(get(ud.hComponent8Axes, 'title'), 'string', 'Binarized Print');
set(ud.hComponent8Image, 'Cdata', BinarizedPrint);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished centralization');
ud.OriginalImageIsStale = 0;
set(DemoFig, 'UserData', ud);
drawnow

%===========================================================================
%%%
%%%  Sub-Function - Crop
%%%

function Crop(DemoFig)
%
load 'informations.dat' -mat
if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Cropping..., please wait !!!');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

fingerprint = fingerprint*graylevmax;

[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);

CroppedPrint = double(CroppedPrint)/graylevmax;
set(get(ud.hComponent1Axes, 'title'), 'string', 'Cropped Print');
set(ud.hComponent1Image, 'Cdata', CroppedPrint);
set(get(ud.hComponent8Axes, 'title'), 'string', 'Binarized Print');
set(ud.hComponent8Image, 'Cdata', BinarizedPrint);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished Crop');
ud.Component1ImageIsStale = 0;
set(DemoFig, 'UserData', ud);
drawnow
```

```
%================================================================
%%%
%%%  Sub-Function - Sectorize
%%%

function Sectorize(DemoFig)
%
load 'informations.dat' -mat

if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Sectorizing..., please wait !!!');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

fingerprint = fingerprint*graylevmax;

[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
for ( i=1:1:175*175)
   tmp=CroppedPrint(i);
   CroppedPrint(i)=whichsector(i);
   if (CroppedPrint(i)==36 | CroppedPrint(i)==37)
      CroppedPrint(i)=tmp/graylevmax;
   else
      CroppedPrint(i)=CroppedPrint(i)/64;
   end

end

set(get(ud.hComponent1Axes, 'title'), 'string', 'SectorizedPrint');
set(ud.hComponent1Image, 'Cdata', CroppedPrint);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished Sectorization');
ud.Component1ImageIsStale = 0;
set(DemoFig, 'UserData', ud);
drawnow
```

```
%=====================================================================
%%%
%%% Sub-Function - Normalize
%%%

function Normalize(DemoFig)
%
load 'informations.dat' -mat

if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Normalizing..., please wait !!!');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

fingerprint = fingerprint*graylevmax;


[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
[NormalizedPrint,vector] = sector_norm( CroppedPrint , 0 , 0);

CroppedPrint = double(CroppedPrint)/graylevmax;
NormalizedPrint = double(NormalizedPrint)/100;
set(get(ud.hComponent1Axes, 'title'), 'string', 'Cropped Print');
set(ud.hComponent1Image, 'Cdata', CroppedPrint);
set(get(ud.hComponent8Axes, 'title'), 'string', 'Normalized Print');
set(ud.hComponent8Image, 'Cdata', NormalizedPrint);
set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished normalization');
ud.Component1ImageIsStale = 0;
set(DemoFig, 'UserData', ud);
drawnow

%=====================================================================
%%%
%%% Sub-Function - Gaborfilter
%%%

function Gaborfilter(DemoFig)
%

if nargin<1
   DemoFig = gcbf;
```

```
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Gabor filter will be shown..., please wait !!!');
ud=get(DemoFig,'Userdata');

num_disk=8;

for (angle=0:1:num_disk-1)

  gabor=gabor2d_sub(angle,num_disk);
  gabor=gabor*128;
  switch angle<num_disk
    case (angle==0),
      set(get(ud.hComponent1Axes, 'title'), 'string', '0 degree gabor');
      set(ud.hComponent1Image, 'Cdata', gabor);
    case (angle==1),
      set(get(ud.hComponent2Axes, 'title'), 'string', '22.5 degree gabor');
      set(ud.hComponent2Image, 'Cdata', gabor);
    case (angle==2),
      set(get(ud.hComponent3Axes, 'title'), 'string', '45 degree gabor');
      set(ud.hComponent3Image, 'Cdata', gabor);
    case (angle==3),
      set(get(ud.hComponent4Axes, 'title'), 'string', '67.5 degree gabor');
      set(ud.hComponent4Image, 'Cdata', gabor);
    case (angle==4),
      set(get(ud.hComponent5Axes, 'title'), 'string', '90 degree gabor');
      set(ud.hComponent5Image, 'Cdata', gabor);
    case (angle==5),
      set(get(ud.hComponent6Axes, 'title'), 'string', '112.5 degree gabor');
      set(ud.hComponent6Image, 'Cdata', gabor);
    case (angle==6),
      set(get(ud.hComponent7Axes, 'title'), 'string', '135 degree gabor');
      set(ud.hComponent7Image, 'Cdata', gabor);
    case (angle==7),
      set(get(ud.hComponent8Axes, 'title'), 'string', '157.5 degree gabor');
      set(ud.hComponent8Image, 'Cdata', gabor);
    otherwise
      error('Nothing !');
  end

end

set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Gabor Filters were shown');
ud.OriginalImageIsStale = 0;
set(DemoFig, 'UserData', ud);
```

```
drawnow

%=======================================================================
%%%
%%%  Sub-Function - Convolute
%%%

function Convolute(DemoFig)
%
load 'informations.dat' -mat

if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Convoluting with eight Gabor filters in process...');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

load 'informations.dat' -mat
fingerprint = fingerprint*graylevmax;

N=175;
num_disk=8;

[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
[NormalizedPrint,vector]=sector_norm(CroppedPrint,0,1);
for (angle=0:1:num_disk-1)

    gabor=gabor2d_sub(angle,num_disk);
    z2=gabor;
    z1=NormalizedPrint;
    z1x=size(z1,1);
    z1y=size(z1,2);
    z2x=size(z2,1);
    z2y=size(z2,2);
    ComponentPrint=real(ifft2(fft2(z1,z1x+z2x-1,z1y+z2y-1).*fft2(z2,z1x+z2x-
1,z1y+z2y-1)));
    px=((z2x-1)+mod((z2x-1),2))/2;
    py=((z2y-1)+mod((z2y-1),2))/2;
    ComponentPrint=ComponentPrint(px+1:px+z1x,py+1:py+z1y);


    [disk,vector]=sector_norm(ComponentPrint,1,0);
    img = double(ComponentPrint)/graylevmax;
```

```
  switch angle<8
    case (angle==0),
      set(get(ud.hComponent1Axes, 'title'), 'string', '0 degree Component');
      set(ud.hComponent1Image, 'Cdata', img);
    case (angle==1),
      set(get(ud.hComponent2Axes, 'title'), 'string', '22.5 degree Component');
      set(ud.hComponent2Image, 'Cdata', img);
    case (angle==2),
      set(get(ud.hComponent3Axes, 'title'), 'string', '45 degree Component');
      set(ud.hComponent3Image, 'Cdata', img);
    case (angle==3),
      set(get(ud.hComponent4Axes, 'title'), 'string', '67.5 degree Component');
      set(ud.hComponent4Image, 'Cdata', img);
    case (angle==4),
      set(get(ud.hComponent5Axes, 'title'), 'string', '90 degree Component');
      set(ud.hComponent5Image, 'Cdata', img);
    case (angle==5),
      set(get(ud.hComponent6Axes, 'title'), 'string', '112.5 degree Component');
      set(ud.hComponent6Image, 'Cdata', img);
    case (angle==6),
      set(get(ud.hComponent7Axes, 'title'), 'string', '135 degree Component');
      set(ud.hComponent7Image, 'Cdata', img);
    case (angle==7),
      set(get(ud.hComponent8Axes, 'title'), 'string', '157.5 degree Component');
      set(ud.hComponent8Image, 'Cdata', img);
    otherwise
      error('Nothing !');
  end

end

set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Finished Convolution');
set(DemoFig, 'UserData', ud);
drawnow

%===========================================================================
%%%
%%%  Sub-Function - Features
%%%

function Features(DemoFig)
%
load 'informations.dat' -mat
```

```
if nargin<1
   DemoFig = gcbf;
end
set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'Convoluting with eight Gabor filters in process...');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

fingerprint = fingerprint*graylevmax;

N=175;
num_disk=8;

[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
[NormalizedPrint,vector]=sector_norm(CroppedPrint,0,1);

for (angle=0:1:num_disk-1)

   gabor=gabor2d_sub(angle,num_disk);
   z2=gabor;
   z1=NormalizedPrint;
   z1x=size(z1,1);
   z1y=size(z1,2);
   z2x=size(z2,1);
   z2y=size(z2,2);
   ComponentPrint=real(ifft2(fft2(z1,z1x+z2x-1,z1y+z2y-1).*fft2(z2,z1x+z2x-
1,z1y+z2y-1)));
   px=((z2x-1)+mod((z2x-1),2))/2;
   py=((z2y-1)+mod((z2y-1),2))/2;
   ComponentPrint=ComponentPrint(px+1:px+z1x,py+1:py+z1y);

   [disk,vector]=sector_norm(ComponentPrint,1,0);

   img = double(ComponentPrint)/graylevmax;
   img1 = double(disk)/51200;
   switch angle<8
      case (angle==0),
         set(get(ud.hComponent1Axes, 'title'), 'string', '0 degree Features');
         set(ud.hComponent1Image, 'Cdata', img1);
      case (angle==1),
         set(get(ud.hComponent2Axes, 'title'), 'string', '22.5 degree Features');
         set(ud.hComponent2Image, 'Cdata', img1);
      case (angle==2),
         set(get(ud.hComponent3Axes, 'title'), 'string', '45 degree Features');
         set(ud.hComponent3Image, 'Cdata', img1);
```

```
    case (angle==3),
      set(get(ud.hComponent4Axes, 'title'), 'string', '67.5 degree Features');
      set(ud.hComponent4Image, 'Cdata', img1);
    case (angle==4),
      set(get(ud.hComponent5Axes, 'title'), 'string', '90 degree Features');
      set(ud.hComponent5Image, 'Cdata', img1);
    case (angle==5),
      set(get(ud.hComponent6Axes, 'title'), 'string', '112.5 degree Features');
      set(ud.hComponent6Image, 'Cdata', img1);
    case (angle==6),
      set(get(ud.hComponent7Axes, 'title'), 'string', '135 degree Features');
      set(ud.hComponent7Image, 'Cdata', img1);
    case (angle==7),
      set(get(ud.hComponent8Axes, 'title'), 'string', '157.5 degree Features');
      set(ud.hComponent8Image, 'Cdata', img1);
    otherwise
      error('Nothing !');
  end

end

set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'Features were extracted');
set(DemoFig, 'UserData', ud);
drawnow
%==========================================================================

%==========================================================================
%%%
%%%  Sub-Function - FingerCode
%%%


function Fingercode(DemoFig)
%
load 'informations.dat' -mat

if nargin<1
  DemoFig = gcbf;
end


set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'FingerCode in process...');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);
fingerprint = fingerprint*graylevmax;

N=175;
num_disk=8;


[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
[NormalizedPrint,vector]=sector_norm(CroppedPrint,0,1);

for (angle=0:1:num_disk-1)
   gabor=gabor2d_sub(angle,num_disk);
   z2=gabor;
   z1=NormalizedPrint;
   z1x=size(z1,1);
   z1y=size(z1,2);
   z2x=size(z2,1);
   z2y=size(z2,2);
   ComponentPrint=real(ifft2(fft2(z1,z1x+z2x-1,z1y+z2y-1).*fft2(z2,z1x+z2x-
1,z1y+z2y-1)));
   px=((z2x-1)+mod((z2x-1),2))/2;
   py=((z2y-1)+mod((z2y-1),2))/2;
   ComponentPrint=ComponentPrint(px+1:px+z1x,py+1:py+z1y);
   [disk,vector]=sector_norm(ComponentPrint,1,0);
   %img = double(ComponentPrint)/graylevmax;
   %img1 = double(disk)/51200;
   finger_code1{angle+1}=vector(1:36);
end

load('informations.dat','img','-mat');
img=imrotate(img,22.5/2);
imgN=size(img,1);
imgM=size(img,2);
modN=mod(imgN,8);
modM=mod(imgM,8);
fingerprint=double(img(modN+1:imgN,modM+1:imgM));

[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
[NormalizedPrint,vector]=sector_norm(CroppedPrint,0,1);

for (angle=0:1:num_disk-1)
   gabor=gabor2d_sub(angle,num_disk);
   z2=gabor;
   z1=NormalizedPrint;
   z1x=size(z1,1);
   z1y=size(z1,2);
   z2x=size(z2,1);
   z2y=size(z2,2);
   ComponentPrint=real(ifft2(fft2(z1,z1x+z2x-1,z1y+z2y-1).*fft2(z2,z1x+z2x-
1,z1y+z2y-1)));
   px=((z2x-1)+mod((z2x-1),2))/2;
   py=((z2y-1)+mod((z2y-1),2))/2;
   ComponentPrint=ComponentPrint(px+1:px+z1x,py+1:py+z1y);
   [disk,vector]=sector_norm(ComponentPrint,1,0);
   %img = double(ComponentPrint)/graylevmax;
   %img1 = double(disk)/51200;
   finger_code2{angle+1}=vector(1:36);
end
% FingerCode added to database
if (exist('fp_database.dat')==2)
   load('fp_database.dat','-mat');
   fp_number=fp_number+1;
   data{fp_number,1}=finger_code1;
   data{fp_number,2}=finger_code2;
   save('fp_database.dat','data','fp_number','-append');
else
   fp_number=1;
   data{fp_number,1}=finger_code1;
   data{fp_number,2}=finger_code2;
   save('fp_database.dat','data','fp_number');
end

message=strcat('FingerCode was succesfully added to database. Fingerprint no.
',num2str(fp_number));
msgbox(message,'FingerCode DataBase','help');

set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'FingerCode calculated');
set(DemoFig, 'UserData', ud);
drawnow
%==========================================================================

%==========================================================================
%%%
%%%  Sub-Function - Check
%%%

function Check(DemoFig)
%
load 'informations.dat' -mat
```

```
if nargin<1
   DemoFig = gcbf;
end


set(DemoFig,'Pointer','watch');
setstatus(DemoFig,'DataBase Scanning...');
ud=get(DemoFig,'Userdata');
fingerprint = getimage(ud.hOriginalImage);

fingerprint = fingerprint*graylevmax;

N=175;
num_disk=8;


[BinarizedPrint,XofCenter,YofCenter]=centralizing(fingerprint,0);
[CroppedPrint]=cropping(XofCenter,YofCenter,fingerprint);
[NormalizedPrint,vector]=sector_norm(CroppedPrint,0,1);

for (angle=0:1:num_disk-1)
   gabor=gabor2d_sub(angle,num_disk);
   z2=gabor;
   z1=NormalizedPrint;
   z1x=size(z1,1);
   z1y=size(z1,2);
   z2x=size(z2,1);
   z2y=size(z2,2);
   ComponentPrint=real(ifft2(fft2(z1,z1x+z2x-1,z1y+z2y-1).*fft2(z2,z1x+z2x-
1,z1y+z2y-1)));
   px=((z2x-1)+mod((z2x-1),2))/2;
   py=((z2y-1)+mod((z2y-1),2))/2;
   ComponentPrint=ComponentPrint(px+1:px+z1x,py+1:py+z1y);
   [disk,vector]=sector_norm(ComponentPrint,1,0);
   %img = double(ComponentPrint)/graylevmax;
   %img1 = double(disk)/51200;
   finger_code{angle+1}=vector(1:36);
end


% FingerCode of input fngerprint has just been calculated.
% Checking with BadaBase
if (exist('fp_database.dat')==2)
   load('fp_database.dat','-mat');
   %---- alloco memoria ----------------------------------
```

```
ruoto1=zeros(36,1);
ruoto2=zeros(36,1);
vettore_d1=zeros(12,1);
vettore_d2=zeros(12,1);
best_matching=zeros(fp_number,1);
% start checking ----------------------------------------
for scanning=1:fp_number
   fcode1=data{scanning,1};
   fcode2=data{scanning,2};
   for rotazione=0:1:11
      d1=0;
      d2=0;
      for disco=1:8
         f1=fcode1{disco};
         f2=fcode2{disco};
         % ora ruoto f1 ed f2  della rotazione ciclica ---------
         for old_pos=1:12
            new_pos=mod(old_pos+rotazione,12);
            if (new_pos==0)
               new_pos=12;
            end
            ruoto1(new_pos)=f1(old_pos);
            ruoto1(new_pos+12)=f1(old_pos+12);
            ruoto1(new_pos+24)=f1(old_pos+24);
            ruoto2(new_pos)=f2(old_pos);
            ruoto2(new_pos+12)=f2(old_pos+12);
            ruoto2(new_pos+24)=f2(old_pos+24);
         end
         %---------------------------------------------------
         d1=d1+norm(finger_code{disco}-ruoto1);
         d2=d2+norm(finger_code{disco}-ruoto2);
      end
      vettore_d1(rotazione+1)=d1;
      vettore_d2(rotazione+1)=d2;
   end
   [min_d1,pos_min_d1]=min(vettore_d1);
   [min_d2,pos_min_d2]=min(vettore_d2);
   if min_d1<min_d2
      minimo=min_d1;
   else
      minimo=min_d2;
   end
   best_matching(scanning)=minimo;
end
[distanza_minima,posizione_minimo]=min(best_matching);
beep;
```

```
   message=strcat('The nearest fingerprint present in DataBase which matchs input
fingerprint is  : ',num2str(posizione_minimo),...
                    ' with a distance of : ',num2str(distanza_minima));
   msgbox(message,'DataBase Info','help');
   %--------------------------------------------------

else
   message='DataBase is empty. No check is possible.';
   msgbox(message,'FingerCode DataBase Error','warn');
end



set(DemoFig,'Pointer','arrow');
setstatus(DemoFig,'DataBase Scanning completed');
set(DemoFig, 'UserData', ud);
drawnow
%===================================================================
```