IMPLEMENTATION OF HAND VEIN BIOMETRIC AUTHENTICATION ON FPGA-BASED EMBEDDED SYSTEM

LOH YEE KOON

UNIVERSITI TEKNOLOGI MALAYSIA
Declaration of Thesis / Undergraduate Project Paper and Copyright

Author's full name: LOH YEE KOOK
Date of birth: 11 July 1986
Title: Implementation of Hand Vein Biometric Authentication on FPGA-based Embedded System

Academic Session: 2009/2010

I declare that this thesis is classified as:

☐ CONFIDENTIAL (Contains confidential information under the Official Secret Act 1972)*
☐ RESTRICTED (Contains restricted information as specified by the organisation where research was done)*
☒ OPEN ACCESS I agree that my thesis to be published as online open access (full text)

I acknowledged that Universiti Teknologi Malaysia reserves the right as follows:

1. The thesis is the property of Universiti Teknologi Malaysia.
2. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

SIGNATURE
860711-38-5868
(NEW IC NO. /PASSPORT NO.)

SIGNATURE OF SUPERVISOR
PROF. DR. MOHAMED KHALIL MOHD. HANI
NAME OF SUPERVISOR

Date: 20 APRIL 2010

NOTES: * If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter from the organisation with period and reasons for confidentiality or restriction.
“I hereby declare that I have read this thesis and in my opinion this thesis is sufficient in terms of scope and quality for the award of the degree of Bachelor of Engineering (Microelectronics)”

Signature: ____________________________________________
Supervisor: Prof. Dr. Mohamed Khalil Mohd. Hani
Date: 20 APRIL 2010
IMPLEMENTATION OF HAND VEIN BIOMETRIC AUTHENTICATION
ON FPGA-BASED EMBEDDED SYSTEM

LOH YEE KOON

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Bachelor of Engineering (Microelectronics)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

APRIL 2010
I declare that this thesis entitled “Implementation of Hand Vein Biometric Authentication on FPGA-based Embedded System” is the result of my own research except as cited in references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature : ______________________________
Name of Candidate : LOH YEE KOON
Date : 20 APRIL 2010
To my beloved father and mother
ACKNOWLEDGEMENT

First of all, I would like to extend my deepest appreciation to my supervisor, Professor Dr. Haji Mohamed Khalil bin Haji Mohd Hani for his patience, support, motivation and guidance throughout my work. Without his constant support and understanding, it is impossible for me to realize my thesis.

I would also like to express my gratitude to my seniors Mohd Annuar and Pei Chee for their tirelessly guidance. They have always been patient, caring and helpful in the process of guiding my work. I would also like to express my appreciation to my comrades Moganeshwaran, Samuel Loh and Gam Meng. I truly appreciate their sharing of ideas as well as their concern.

My warmest regards to my beloved family for their caring encouragement and moral support throughout this journey. Last but not least, I would like to thank to all whoever directly or indirectly help in making this thesis successful.
ABSTRACT

With the use of contemporary electronic technology, biometric system has been introduced to be implemented for personal authentication and security clearance purpose. However, biometric authentication system is not widely implemented on embedded system. This project proposed a personal authentication embedded system based on one of the newest biometric techniques which is the vein patterns in the back of the hand. This system is running Real Time Operating System (RTOS) Nios2-Linux, implemented using Altera Stratix II board. The system comprises of two processes which are enrolment and verification. The system consists of four modules which are image acquisition module, image pre-processing module, feature extraction module and matching module. A low cost vein capturing device has been successfully developed using webcam to take snapshots of hand vein pattern under a source of infrared radiation. A small area (70 x 70 pixels) is located as the Region of Interest (ROI) to get the main vein pattern in the back of hand. Image pre-processing module involves median filtering, canny edge detection, resizing, thresholding, opening, closing and thinning algorithms. Minutiae extraction technique was used to extract the feature of hand vein pattern and stored as template in database. The matching module was conducted by comparing the template generated in verification process with the template in system database to verify the user. The database of 120 images of 30 candidates shows promising result of the equal error rate 3.68%.
ABSTRAK

# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DECLARATION</td>
<td>II</td>
</tr>
<tr>
<td></td>
<td>DEDICATION</td>
<td>III</td>
</tr>
<tr>
<td></td>
<td>ACKNOWLEDGEMENT</td>
<td>IV</td>
</tr>
<tr>
<td></td>
<td>ABSTRACT</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>ABSTRAK</td>
<td>VI</td>
</tr>
<tr>
<td></td>
<td>TABLE OF CONTENTS</td>
<td>VII</td>
</tr>
<tr>
<td></td>
<td>LIST OF TABLES</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>LIST OF FIGURES</td>
<td>XI</td>
</tr>
<tr>
<td></td>
<td>LIST OF SYMBOLS</td>
<td>XIV</td>
</tr>
</tbody>
</table>

1 INTRODUCTION 1

1.1 Problem Background 1
1.2 Problem Statement 2
1.3 Objectives 3
1.4 Scope of Work 4
1.5 Thesis Organization 5

2 LITERATURE REVIEW 7

2.1 Biometric recognition system 7
2.2 FPGA-based Palm-print Biometric System 8
2.3 Region of Interest (ROI) detection 13
2.4 Vein recognition system 17
   2.4.1 The concept 18
   2.4.2 Hardware setup 19
   2.4.3 Image processing algorithms 20

3 RESEARCH METHODOLOGY 23

3.1 Project Workflow 23
3.2 Research Timeline 25
3.3 Tools and Software 27

4 THEORIES AND CONCEPT 29

4.1 Fundamentals of image processing 29
4.2 Image Pre-processing Module 31
   4.2.2 Median Filtering 31
   4.2.3 Canny Edge Detection 33
   4.2.4 Local Thresholding 35
   4.2.5 Morphological Image Processing 37
   4.2.6 Opening 39
   4.2.7 Closing 40
   4.2.8 Thinning 41
4.3 Feature Extraction Module 43
4.4 Matching Module 44

5 SYSTEM DESIGN AND IMPLEMENTATION 47
5.1 System architecture design
   5.1.1 Image Capturing Module
   5.1.2 Image Pre-processing Module
   5.1.3 Feature Extraction Module
   5.1.4 Template of Minutiae Extraction
   5.1.5 Matching Module
   5.1.6 Threshold and EER calculation

5.2 System Demonstration

6 RESULTS AND DISCUSSION

   6.1 Image Pre-processing Module
   6.2 Feature Extraction Module
   6.3 Template Generation
   6.4 Performance Analysis
      6.4.1 Accuracy
      6.4.2 Speed

7 CONCLUSION

   7.1 Contribution
   7.2 Limitation
   7.3 Recommendation for future work

REFERENCES

APPENDIX A
APPENDIX B
### LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE NO</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Properties of cross number</td>
<td>43</td>
</tr>
<tr>
<td>6.1</td>
<td>Detailed techniques and output images in image pre-processing module.</td>
<td>71</td>
</tr>
<tr>
<td>6.2</td>
<td>Execution time for each image pre-processing blocks</td>
<td>75</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE NO</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Hand Vein Biometric Authentication System</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>System Architecture for overall project</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Results for each stage in image pre-processing module</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Sample output of extracted features</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>The key points based on finger boundary</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>The central part for feature extraction</td>
<td>15</td>
</tr>
<tr>
<td>2.6</td>
<td>Skin Colour Thresholding</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>Competitive valley detection algorithm</td>
<td>15</td>
</tr>
<tr>
<td>2.8</td>
<td>ROI location technique</td>
<td>16</td>
</tr>
<tr>
<td>2.9</td>
<td>Defining the ROI</td>
<td>17</td>
</tr>
<tr>
<td>2.10</td>
<td>Optical window used for vein pattern detection</td>
<td>18</td>
</tr>
<tr>
<td>2.11</td>
<td>Various arrangement of LEDs used in vein pattern acquisition</td>
<td>19</td>
</tr>
<tr>
<td>2.12</td>
<td>The wireframe section of the vein scanning device</td>
<td>20</td>
</tr>
<tr>
<td>2.13</td>
<td>The block diagram of image processing algorithms</td>
<td>21</td>
</tr>
<tr>
<td>2.14</td>
<td>Output images of image processing algorithms</td>
<td>22</td>
</tr>
<tr>
<td>3.1</td>
<td>Project Workflow</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Research Timeline</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Arrays of pixels arranged in rows and columns</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>A grayscale image with sample pixel value</td>
<td>30</td>
</tr>
<tr>
<td>4.3</td>
<td>A 3*3 window</td>
<td>32</td>
</tr>
</tbody>
</table>
4.4 Original image before median filtering.
4.5 Image resulted from median filtering
4.6 The convolution mask of Sobel filter in x and y direction
4.7 Image before Canny edge detection
4.8 Image after Canny edge detection
4.9 Example before and after local thresholding
4.10 Morphological operations
4.11 Examples of opening operation on a binary image
4.12 Example of closing operation on binary image
4.13 Designation of nine pixels in a 3*3 window.
4.14 Example before and after thinning operation
4.15 Comparison of two sets of points using MHD
5.1 System architecture of hand vein biometric authentication
5.2 IR LEDs arranged in concentric manner
5.3 Sample of captured hand vein image
5.4 Code fragment of jpg to bmp conversion
5.5 Behavioral flow chart and C code fragment of median filtering
5.6 Behavioral flow chart and C code fragment of Canny edge detection
5.7 Behavioral flow chart and C code fragment of ROI detection
5.8 Behavioral flow chart and C code fragment of local thresholding
5.9 Behavioral flow chart and C code fragment of opening
5.10 Behavioral flow chart and C code fragment of closing
5.11 Behavioral flow chart and C code fragment of thinning algorithm
5.12 Behavioral flow chart and C code fragment for minutiae extraction
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.13</td>
<td>C code fragment of minutiae template</td>
<td>59</td>
</tr>
<tr>
<td>5.14</td>
<td>Behavioral flow chart and C code fragment of MHD</td>
<td>60</td>
</tr>
<tr>
<td>5.15</td>
<td>Behavioral flow chart and C code fragment of threshold and EER calculation</td>
<td>61</td>
</tr>
<tr>
<td>5.16</td>
<td>The main menu and demonstration of selection 1</td>
<td>62</td>
</tr>
<tr>
<td>5.17</td>
<td>Demonstration of selection 2</td>
<td>63</td>
</tr>
<tr>
<td>5.18</td>
<td>Matching of images to get the client and imposter scores</td>
<td>64</td>
</tr>
<tr>
<td>5.19</td>
<td>Calculation of threshold and EER</td>
<td>64</td>
</tr>
<tr>
<td>5.20</td>
<td>Example of successful authentication</td>
<td>65</td>
</tr>
<tr>
<td>5.21</td>
<td>Example of failed authentication</td>
<td>66</td>
</tr>
<tr>
<td>5.22</td>
<td>Database list of the system</td>
<td>67</td>
</tr>
<tr>
<td>6.1</td>
<td>Captured raw image in jpeg format</td>
<td>69</td>
</tr>
<tr>
<td>6.2</td>
<td>Input image in bmp format</td>
<td>69</td>
</tr>
<tr>
<td>6.3</td>
<td>Output image for each process in image pre-processing module</td>
<td>70</td>
</tr>
<tr>
<td>6.4</td>
<td>Output image of minutiae extraction</td>
<td>73</td>
</tr>
<tr>
<td>6.5</td>
<td>Sample of template</td>
<td>74</td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS

PIN - Personal Identification Number
ROI - Region Of Interest
FPGA - Field Programmable Gate Array
EER - Equal Error Rate
LED - Light Emitting Diode
IR - Infra Red
PC - Personal Computer
MHD - Modified Hausdorff Distance
RTOS - Real Time Operating System
CHAPTER 1

INTRODUCTION

This research presents details of research work done in the development of a back hand vein biometric authentication embedded system implemented in Altera Nios II prototyping system with the use of Altera Stratix II Board. The system is running with 100MHz frequency and applied using RTOS which is Nios2-Linux.

1.1 Project Background

Generally, there are two major methods of personal authentication and recognition for security purposes. The common one is the traditional authentication method based on “what you have” which involves the use of keys, cards, and “what you know” such as password, personal identification numbers (PINs). The current advancement in electronic technology enables biometric system to be implemented for personal authentication and security clearance purpose. In order to ensure the highest standard of security especially for access into high profile area/zone, personal recognition has to be performed based on physiological and behavioral features of authenticated personnel such as vein, fingerprint, iris, handwriting, voice, face etc.
1.2 Problem Statement

In the contemporary material world, demand for higher level of security to protect personal or organizational properties is ever growing. The use of traditional security clearance tools such as keys, passwords and personal identification codes have failed to provide the required level of security especially in the effort to allow access of only authorized personnel into places of utmost importance such as armory, nuclear missile control room, high profile research laboratories, and bank vaults. This is due to higher capability to forge physical keys and advancement in hackers’ ability to acquire passwords and PIN codes. Hence, personnel authentication method based on physiological features such as palm prints, finger prints, voice and iris pattern which are characteristics for a specific person is essential in order to ensure grant of access to authorized personnel only.

On the other hand, automatic and reliable personal identification is essential in our daily lives due to the growing importance of information technology. The use of traditional authentication method for personal identification purpose such as keys and cards can be easily forgotten, duplicated or left at home whereby passwords or PINs can be shared or forgotten. Therefore, it is more secure to use biometric recognition because biometric is based upon a distinguishable trait possessed by each individual. The unique trait is part of us.

One of the most effective and highly secure authentication methods is by using hand vein as basis for personal identification. Hand vein recognition refers to the technique that automatically checks hand vein samples and gives the answer whether the samples are from same hand source.
One of the major issues related to the use of other biometrics such as fingerprint, palm print and iris is they do not show the aliveness of authenticated personnel. Fingerprint, palm print and iris can be easily extracted from dead authorized personnel for fraudulent access and security clearance. Hence, this results in lower level of security especially in high security profile area. Moreover, finger and palm are highly susceptible to external damages which may cause distortion to these forms of biometric features resulting in anticipated problem during authentication process. Additionally, several existing biometric features are visible to the naked eyes or photographic devices which make them easy to replicate.

1.3 Objectives

From the discussion above, this project has set three objectives.

1. To design and develop an image capturing device that can capture hand vein image of users.

2. To design and develop a hand vein biometric system prototype that can process every image inserted by the user and extract the features from the image.

3. To implement the hand vein biometric authentication system in FPGA-based embedded system on Altera Stratix II board.
1.4 Scope of Work

A hand vein biometric system consists of many stages. Figure 1.1 shows a typical hand vein biometric system which consists of two main modules, enrolment module and authentication module. Enrolment module is the process to obtain the data of the user to keep in the system as database. Verification Module is to verify the identity of the user who tries to access the system. The matching module is conducted by comparing the feature extracted in authentication module with the data in system database created during enrolment stage.

The followings are the steps involved towards development of the system:

1. The input images are in colour jpeg format captured using a low cost capturing device.

2. Several image pre-processing processes are carried out to enhance and outline the hand vein pattern before the features are extracted for database and authentication purpose.

3. For the feature extraction module, this project only extracts the most prominent type, which is ridge ending and ridge bifurcation.

4. A simple matching system using Modified Hausdorff Distance (MHD) is designed to validate the system.

5. The hand vein biometric authentication system is implemented on an Altera Stratix II board, running RTOS Nios2-linux with 100MHz frequency.
1.5 Thesis Organization

This thesis is organized into seven chapters. The first chapter reviews the background, problem statement, objectives of the project and scope of work followed by the second chapter which presents the background of biometric system and previous work concerning hand vein recognition system.

Chapter 3 presents the research methodology, system design procedures and tools that have been used in this project. The theories and concepts of the techniques used in this system are discussed in details in Chapter 4. Chapter 5 gives the general idea of how the FPGA-based Hand Vein Biometric System is being implemented. It describes the design of system architecture of the system and the demonstration of the system prototype is discussed here.
The results for each stage in image pre-processing module, feature extraction module and the performance analysis conducted in this work are discussed in Chapter 6 whereas the research work is summarized in the final chapter. Suggestions and recommendations for future work are also stated here.
This chapter presents the background of the biometric system and review of a FPGA-based palm-print biometric system. Previous work on vein recognition system is reviewed also.

2.1 Biometric recognition system

Biometrics is the automated method of recognizing a person based on a physiological or behavioral characteristic. Biometric technologies are becoming the foundation of an extensive array of highly secure identification and personal verification solutions.

Biometric technologies should be considered and evaluated giving full consideration to the following characteristics:

- **Universality**: which means that every person must have the characteristic
- **Uniqueness**: which indicates that two distinct people cannot have the same characteristic.

- **Permanence**: which means that the characteristic cannot change according to the time.

- **Collectability**: which indicates that this characteristic can be measured quantitatively.

- **Performance**: where the identification process must present an acceptable results.

- **Circumvention**: referring to the ability to get destroyed

(http://www.globalsecurity.org)

### 2.2 FPGA-based Palm-print Biometric System

Two final year projects which were developed by undergraduate students of Universiti Teknologi Malaysia are reviewed. The titles of the projects are as follow:
• Lim Min Yen [1]. “FPGA-based Palm-Print Biometric System – Image Pre-processing module and Application”. Universiti Teknologi Malaysia.


Both projects are integrated to form a FPGA-based Palm-print Biometric System and being applied in an attendance registration system for verification purposes. In the project developed by Lim Min Yen [1], the objectives are:

1. To design and develop an image pre-processing module that is capable of enhancing image data at high speed to integrate with feature extraction module and form the enrolment module of a complete palm-print recognition system.

2. To enhance raw images from the webcam utilizing various image pre-processing algorithms in order to increase the efficiency of the feature extraction module and for a better analysis by users.

3. To build an application for FPGA-based palm-print biometric system functioning as an attendance registration system.

In the image pre-processing module performed in the project of Lim Min Yen [1], a webcam was used to capture the image of the palm-print and the image processing stages involved were colour to grayscale conversion, brightness correction, contrast correction, histogram equalization and image size alteration. Figure 2.1 shows the system architecture for overall project and Figure 2.2 shows each stage in the image pre-processing module.
Figure 2.1 System Architecture for overall project. [1]
Figure 2.2 Results for each stage in image pre-processing module. [1]

In the work done by Soon Boon Hooi [2], the objectives are:
1. To design a Features Extraction Module to extract the palm-print features and translate into a template representing each individual’s identity using Texture-based Algorithm.

2. To design a Matching Module to identify the person who use the system.

3. To integrate Features Extraction and Matching Modules with Image Pre-processing Module to develop an Attendance Registration System for verification purpose.

In the feature extraction module done by Soon Boon Hooi [2], the technique used was texture-based feature measurement. It involves extraction of local attributes of a palm that provides the detailed information of a palm-print. The local texture energy was introduced to define the local features of a palm-print. The features were represented in one-dimensional (1-D) array which consists of 64 integers. Figure 2.3 shows the sample output of features extracted by Soon Boon Hooi [2].

Figure 2.3 Sample output of extracted features. [2]
In matching module, dissimilarity measurement was compared with the pre-defined threshold which was 8 in this project, to determine whether the user should be accepted or not.

In conclusion, they had successfully developed a prototype of a FPGA-based Palm-print Biometric System as well as an attendance registration system for verification purpose. However, a better capture device is recommended to be implemented in order to equalize the uneven image brightness. Besides, their research study did not implement a boundary tracking algorithm to reduce inaccuracy caused by image positioning. The palm-print biometric system can be integrated with other biometric identification, such as fingerprint, iris and hand geometry.

2.3 Region of Interest (ROI) detection

ROI detection is a very common and important step in image processing. It is used to process a single sub region of an image, leaving other regions unchanged. In biometric system, ROI refers to the part that contains the most important and useful information. The region will then be cropped and implemented with some image processing processes.

In the survey of palm-print by Kong et al. [8], an overview of current palm-print research describing in particular capture devices, preprocessing, verification algorithms, palm print-related fusion and algorithms was reported.

Aforementioned preprocessing procedure is used to align different palm-print images and to segment the center for feature extraction. Most of the preprocessing
algorithms employ the key points between fingers to set up a coordinate system. Hand tracking and Region of Interest (ROI) extraction involves five steps:

1. binarize the palm images,
2. extract the contour of hand and/or fingers,
3. detect the key points,
4. establish a coordination system
5. extract the central parts

Figure 2.4 illustrates the key points and Figure 2.5 shows a preprocessed image. The first and second steps in all the preprocessing algorithms are similar. However, the third step has several different implementations including tangent, bisector and finger-based to detect the key points between fingers.
In the research performed by Michael et al. [9], the hand tracking and ROI extraction steps proposed by Michael and his team consist of three stages. First, the hand image is segmented from the background by using the skin-color thresholding method. After that, a valley detection algorithm is used to find the valleys of the fingers. These valleys serve as the base points to locate the palm print region.

Figure 2.5 The central part for feature extraction.[8]

Figure 2.6 Skin colour thresholding. (a) The original hand image; (b) Segmented hand image in binary form.

Figure 2.7 Competitive valley detection algorithm
In another project done by Wang et al. [15], the ROI detection was defined by four steps. Firstly, the contour of the hand was extracted using Sobel filter. Then for each point on the hand contour, the distance between this point and the mid-point of wrist was calculated and the valleys points were found in the distance profile. A ROI was detected as a rectangular region from the valley points. Figure 2.9 shows the steps taken in the project.
Figure 2.9 Defining the ROI. (a) A FIR hand image. (b) Hand contour. (c) Distance profile between the contour points and the mid-point of wrist. (d) Locating the ROI.[15]

2.4 Vein recognition system

A few research works on vein recognition system have been reviewed. The concept of vein detection, hardware setup and the image processing techniques used in the research work done were focused in the following sections.
2.4.1 The concept

A vein detection process consists of a capturing device to take snapshots of vein under a source of infrared radiation. The infrared radiation is absorbed by tissues in our blood. In order to achieve visual penetration through the respective tissue, lighting should be performed under a very tight optical window namely 740nm up to 760nm. According to the research done by Septimiu Crisan et al.\cite{16}, the reason for using the aforementioned wavelength is due to the fact that the deoxidized hemoglobin [Hb] in the vein vessels absorbs almost completely the radiation while the oxidized hemoglobin [HbO] in the arteries becomes almost transparent. Figure 2.10 shows the optical window used for detecting vein pattern.

![Figure 2.10 Optical window used for vein pattern detection.](image)

Figure 2.10 Optical window used for vein pattern detection.
2.4.2 The hardware setup

In the early experiment done by Septimiu Crisan et al. [16], it was found that various matrix arrangements of the LEDs would modify the distribution of the radiation intensity. The aforementioned configurations are 2D single or double arrays, rectangular array and concentric array. Figure 2.11 shows various near infrared LEDs array used in vein pattern acquisition. It was concluded in the experiment that a double or triple concentric circles of LED array as a light source provides a very good uniformity of radiation even with a reduced number of diffusers.

Figure 2.11 Various arrangement of LEDs used in vein pattern acquisition.
   a) double line  b) rectangular  c) concentric array

In another experiment done by Septimiu Crisan et al [16], 18 near infrared LEDs model ELD-740-524 were used as the illumination source. The LEDs were placed in three concentric circles in the same focal plane. The purpose of using this arrangement of lighting system was to ensure the lighting to be constant all over the region. Figure 2.12 show the vein scanning device.
2.4.3 Image processing algorithms

The image processing algorithms implemented in the experiment by Septimiu Crisan et al.[16] included negative operation and edge detection, thresholding, noise elimination and thinning. Figure 2.13 shows the block diagram of the image processing algorithm.
Figure 2.13 The block diagram of image processing algorithms. [16]
In a study of hand vein recognition method published by Yuhang Ding et al.[17], threshold segmentation, filtering, and thinning algorithms were discussed in deep. The algorithms introduced in the research work have a comparatively good recognition effect. Figure 2.14 shows the output images of the algorithms.

![Output images of image processing algorithms.](image)

Figure 2.14 Output images of image processing algorithms.

a) The standard image  b) Threshold segmentation  c) Filtering  d) Thinning
CHAPTER 3

RESEARCH METHODOLOGY

This chapter describes the methodology used in this project. It begins with the project workflow and followed by software and tools that are applied in this project.

3.1 Project Workflow

This project is about the implementation of hand vein biometric authentication on embedded system. The system is downloaded in an embedded system using Altera Stratix II, running RTOS Nios2-linux.

The work begins with the purpose of doing this project. Problem statement and scope of work are being defined before developing the system architecture. During the system architecture development, there are 2 main parts including learning process and software development. Several tools and software need to be familiarized and mastered to design and complete the system. In this project, the four
modules are written in C programming. Preliminary study is done on the current algorithm available.

For software development process, the algorithms for image pre-processing, feature extraction and matching module are developed and designed in C programming for linux. Database and compilation synthesis are developed also.

Next, the system is being tested to check if the system is giving desire results. Finally, performance measurement and analysis are done on the overall system.

Literature review on biometrics and image-processing is being carried out throughout the whole project. At the same time, documentation and report writing are processes being conducted since the beginning of the project until the end of the project. The overall project workflow is shown in Figure 3.1.
3.2 Research Timeline

Timeline is another important concern in carrying out a research project. Therefore, timeline for each process need to be planned out at the beginning of the project. Gantt chart shows the research timeline for this project.
<table>
<thead>
<tr>
<th>ACTIVITIES</th>
<th>TIMELINE (WEEK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literature Review</td>
<td></td>
</tr>
<tr>
<td>Project Planning</td>
<td></td>
</tr>
<tr>
<td>Preliminary Study</td>
<td></td>
</tr>
<tr>
<td>Development of image capturing device</td>
<td></td>
</tr>
<tr>
<td>Development of Image Pre-processing Module</td>
<td></td>
</tr>
<tr>
<td>Development of Feature extraction Module</td>
<td></td>
</tr>
<tr>
<td>Development of Matching Module</td>
<td></td>
</tr>
<tr>
<td>Integration and Testing</td>
<td></td>
</tr>
<tr>
<td>Result Analysis</td>
<td></td>
</tr>
<tr>
<td>Reporting</td>
<td></td>
</tr>
<tr>
<td>Finalization of Report for Submission</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2 Research Timeline
3.3 Tools and Software

The main components needed in this project are the software development tools, a personal computer and webcams to complete and implement the system. The tools and software needed in this project are as follows:

(i) Modified Infrared Webcam – By modifying a webcam into infrared webcam with infrared LEDs as radiation source, a capturing device is used to capture the hand vein image.

(ii) Pendrive – Pendrive is used to store the software coding and transfer to FPGA board.

(iii) GNU C++ Compiler 5.02 – A C and C++ compiler used with the kernel Linux.

(iv) Personal Computer – used for image capturing module and display the images.

(v) Altera FPGA Stratix II Board – A FPGA hardware platform to prototype embedded systems which is pre-configured with a 32-bit Nios II Processor hardware reference design and a software reference design stored in flash memory.

(vi) Altera Quartus II version 9.0 – A hardware design tool for design entry, compilation, simulation and implementation software tool.
(vii) Nios II SOPC builder – A wizard to create Nios II system module. Nios II processor designs are created using Nios II SOPC builder system design tool.
CHAPTER 4

THEORIES AND CONCEPT

This chapter presents the fundamental of image processing and the techniques used in image pre-processing module. The technique used in feature extraction and matching module are discussed as well.

4.1 Fundamentals of Image Processing

An image is an array, or a matrix, of square pixels (picture elements) arranged in columns and rows. It is considered to be a function of two real variable at coordinate position (x,y). Pixel is a computer word formed from PICture ELement, because a pixel is the smallest element of the image. Images consist of pixels, and are dimensioned in pixels. Pixels are parameterized by position, intensity and time. A grayscale image each picture element has an assigned intensity that ranges from 0 to 255. A normal grayscale image has 8 bit colour depth whereas a “true colour” image has 24 bit colour depth.
The majority of image processing tasks belongs in one of the following categories:

- **Enhancement** - contrast improvement, smoothing, sharpening
• **Restoration** - noise reduction, deblurring

• **Segmentation** - counting, identification, separation

• **Registration** - comparison

• **Compression** - transmission, storage, fast processing

• **Pattern recognition** - classification, matching

• **Others** - tomography, morphing, inpainting, color correction etc.

4.2 Image Pre-processing Module

4.2.2 Median Filtering

Median filter is used to reduce the noise and burrs in an image. The median filter is a sliding-window spatial filter of size w*w. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the middle pixel with the median value. An example of median filtering of a single 3x3 window of values is shown in Figure 4.3 below. Figure 4.4 and 4.5 shows example before and after median filtering respectively.
Figure 4.3 A 3*3 window. (a) unfiltered value in 3 x 3 window. (b) Median filtered.

Figure 4.4 Original image before median filtering.

Figure 4.5 Image resulted from median filtering.
4.2.3 Canny Edge Detection

Canny edge detection is done by first filter out any noise in the original image before trying to locate and detect any edges. The Gaussian filter can be computed using a simple mask, therefore it is used exclusively in the Canny algorithm. The Gaussian smoothing can be performed using standard convolution methods. A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulating a square of pixels at a time.

The next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image. The Sobel operator uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows).

The edge strength of the gradient is then approximated using the formula:

$$|G| = |G_x| + |G_y|$$  \hspace{1cm} (4.1)
Once the gradient in x and y directions are known, the edge direction is found by the following formula:

\[ \theta = \text{invtan} \left( \frac{G_y}{G_x} \right) \]  

(4.2)

The algorithm continues by doing non maximum suppression. Non maximum suppression is used to trace along the edge in the edge direction and suppress any pixel value (sets it equal to 0) that is not considered to be an edge. The gradient array is now further reduced by hysteresis. Hysteresis is used to track along the remaining pixels that have not been suppressed. Hysteresis uses two thresholds and if the magnitude is below the first threshold, it is set to zero (made a non-edge). If the magnitude is above the high threshold, it is made an edge. Figure 4.7 and 4.8 show the example before and after Canny edge detection.

Figure 4.7 Image before Canny edge detection.
Local thresholding is used to convert the grayscale image into a bi-level representation which are black with ‘0’ pixel and white with ‘255’ pixel. This technique applied on the vein image in order to extract and outline the vein pattern. In this project, a simple and effective local dynamic threshold method by K.J. Wang et al. [11] is adopted.

Let $g(x, y) \in [0,255]$ be the intensity of a pixel at location $(x, y)$ in a grayscale image. In local dynamic thresholding techniques, the threshold, $t(x, y)$ for each pixel has to be calculated such that,

$$
\text{out}(x, y) = \begin{cases} 
0 & \text{if } g(x, y) \leq t(x,y) \\
255 & \text{otherwise}
\end{cases} \quad (4.3)
$$
where \( t(x, y) \) is its threshold.

The threshold, \( t(x,y) \) is calculated using equation below:

\[
t(x,y) = m(x,y) + k \times s(x,y)
\]  

(4.4)

where \( m(x, y) \) and \( s(x, y) \) is the mean and standard deviation of the pixel intensities in a \( w \times w \) size window centered on the pixel \( (x, y) \), and \( k \) is the coefficient correction constant.

After calculating the threshold value, each pixel \( (x,y) \) is then replaced with \( out(x,y) \) which is the output pixel using equation (4.3). Local threshold determination using (4.4) required high computation power as it involved a lot of multiplication, division, power, and square root arithmetic operation when calculating the \( s(x,y) \).

In this project, the threshold is calculated using equation (4.5) as proposed by P.C. Eng and M. Khalil-Hani [18]. The window size used in this project is 13.

\[
t(x,y) = m(x,y)
\]  

(4.5)
4.2.5 Morphological Image Processing

Morphological processing is described as operations on sets of pixels. Only morphological operations for binary images will be discussed here. Morphological processing for grayscale images requires more sophisticated mathematical development. It was primarily developed in the 60’s by French mathematicians Jean Serra and Georges Matheron. The basic idea in binary morphology is to probe an image with a simple, pre-defined shape, which is called the structured element, drawing conclusions on how this shape fits or misses the shapes in the image.

Four basic morphological operations are used in the processing of binary images:

- **Erosion**: shrink objects
- **Dilation**: grow objects
- **Opening**: erosion followed by dilation (disconnect parts)

- **Closing**: dilation followed by erosion (remove holes)

In erosion, every object pixel that is touching a background pixel is changed into a background pixel. In dilation, every background pixel that is touching an object pixel is changed into an object pixel. Erosion makes the objects smaller, and can break a single object into multiple objects. Dilation makes the objects larger, and can merge multiple objects into one. Figure 4.10 shows the morphological operations.

Figure 4.10 Morphological operations. [10]
4.2.6 Opening

Opening is defined as erosion followed by dilation. Opening is used to handle noise in image and smooth the contour of image. It tends to remove some of the foreground (bright) pixels from the edges of regions of foreground pixels. As with other morphological operators, the exact operation is determined by a structuring element. The effect of the operator is to preserve foreground regions that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of foreground pixels. Figure 4.11 shows the examples of opening operation on a binary image. Opening is defined as the equation below.

\[ A \circ B = (A \ominus B) \oplus B \quad (4.6) \]

Figure 4.11 (a) Original binary image. (b) Result of an opening with a 3×9 vertically oriented structuring element (c) Result of an opening with a 9×3 horizontally oriented structuring element.
4.2.7 Closing

A closing is defined as dilation followed by erosion. Closing tends to enlarge the boundaries of foreground (bright) regions in an image (and shrink background color holes in such regions), but it is less destructive of the original boundary shape. The effect of the operator is to preserve background regions that have a similar shape to this structuring element, or that can completely contain the structuring element, while eliminating all other regions of background pixels. Closing can sometimes be used to selectively fill in particular background regions of an image. A closing is defined as the equation below.

\[ A \cdot B = (A \oplus B) \ominus B \quad (4.7) \]

Figure 4.12 Example of closing operation on binary image.

Figure 4.12 shows an example of closing operation on a binary image. The original image (a) contains large and small holes. A closing with a disk-shaped structuring element with a diameter larger than the small holes, but smaller than the
large holes can remove the small holes while retaining the large holes as shown in
the result image (b).

4.2.8 Thinning

Thinning is another morphological operation that is used to remove selected
foreground pixels from binary images. It is very useful for skeletonization. Thinning
is normally only applied to binary images, and produces another binary image as
output.

In this project, in order to extract the skeleton image of the vein pattern which
consist only a single pixel wide texture, a fast parallel algorithm for thinning digital
patterns proposed by T.Y Zhang and C.Y Suen [12] is used. A 3*3 window size as
illustrated in Figure 4.13 is used. Pixel P0 will be removed if it meets the following
thinning conditions. Two iterations involved to preserve the connectivity of the
skeleton. The first two thinning condition for both iteration is the same. Figure 4.14
shows an example of thinning operation.

Thinning conditions, first iteration,

i. \[ 2 \leq B(P0) \leq 6 \]
ii. \[ A(P0) = 1 \]
iii. \[ P1 \cdot P3 \cdot P5 = 0 \]
iv. \[ P3 \cdot P5 \cdot P7 = 0 \]

Thinning conditions, second iteration,

i. \[ 2 \leq B(P0) \leq 6 \]
ii. \( A(P0) = 1 \)
iii. \( P1 \times P3 \times P7 = 0 \)
iv. \( P1 \times P5 \times P7 = 0 \)

Figure 4.13 Designation of nine pixels in a 3*3 window.

Figure 4.14 Example before and after thinning operation.
4.3 Feature Extraction Module

Minutiae feature extraction technique proposed by L. Y. Wang et al.[15] was used in this project. The technique utilizes the minutiae features extracted from the vein pattern for recognition. The most widely used method for minutiae feature extraction is the cross number (CN) concept. Cross number is defined as number of transition from 0 to 1 (and vice versa) for the surrounding pixel around P0. A 3*3 window size from P1 to P8 is shown in Figure 4.13. Cross number is defined as the following equation:

\[
CN (P0) = \sum_{i=1}^{8} |P_i - P_{i+1}|
\]  

(4.8)

Table 4.1 shows the property of cross number of a pixel for different kind of minutiae point. The minutiae features extracted in this project include bifurcation points and ending points. A template was generated for each image to store the minutiae features information which include the x and y location of the point and the type of point.

<table>
<thead>
<tr>
<th>CN</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>Isolated point</td>
</tr>
<tr>
<td>2,3</td>
<td>Ridge ending point</td>
</tr>
<tr>
<td>4,5</td>
<td>Connecting point</td>
</tr>
<tr>
<td>6,7</td>
<td>Bifurcation point</td>
</tr>
<tr>
<td>8</td>
<td>Cross point</td>
</tr>
</tbody>
</table>

Table 4.1 Properties of cross number
4.4 Matching Module

Since the vein pattern is represented as a set of two-dimensional points, matching of a pair of such pattern can be achieved by measuring the Hausdorff distance between the two minutiae sets as proposed by O. Jesorsky et al. [13].

Hausdorff distance measures distance between two sets of points. For two point sets \( A = \{a_1, \ldots, a_m\} \) and \( B = \{b_1, \ldots, b_n\} \), Hausdorff distance is defined as,

\[
H(A, B) = \max( h(A, B), h(B, A) ) \tag{4.8}
\]

where

\[
h(A, B) = \max_{a \in A} \min_{b \in B} ||a - b|| \tag{4.9}
\]

The smaller the value of \( H \), the more similar the two point sets are. However, this original form of Hausdorff distance is very sensitive to outlier points. The Modified Hausdorff Distance (MHD) introduced by Dubuisson et al. [14] overcomes this problem. In MHD, the directed Hausdorff Distance is defined as

\[
h(A, B) = \frac{1}{m} \sum_{a \in A} \min_{b \in B} ||a - b|| \tag{4.10}
\]

The MHD takes the average value of all distance from point in A to its nearest neighbor in B, rather than taking the farthest point as defined in (4.10). This method is less sensitive to outlier point and gives best performance. Figure 4.15 below illustrates the comparison of two sets of points using MHD.
Given two sets of points $A$ and $B$, find $h(A, B)$.

Compute the distance between $a_i$ and $b_j$'s.

... and keep the shortest.
Figure 4.15 Comparing two sets of points using MHD
CHAPTER 5

SYSTEM DESIGN AND IMPLEMENTATION

This chapter describes the implementation of hand vein biometric authentication on FPGA-based embedded system. It includes the overall system architecture and detailed design of the four main modules in the system. The last part of this chapter demonstrates the system prototype.

5.1 System architecture design

The system architecture of hand vein biometric authentication system consists of four main parts. They are Image capturing Module, Image Pre-processing Module, Feature Extraction Module, and Matching Module as shown in Figure 4.1. The image capturing module was done on personal computer (PC), the other modules which were image pre-processing, feature extraction and matching were download on embedded system and processed using the Nios2 processor. Details of each module are described in the following section.
Figure 5.1 System architecture of hand vein biometric authentication.

5.1.1 Image Capturing Module

In this project, a low cost vein capturing device was built with webcam and 18 near infrared LEDs model SFH485 from OSRAM Opto Semiconductor. The webcam was modified to become an infrared webcam using photographic film. The device is similar to the device developed by [2] but the LEDs are 5mm casing medium angle IR emitter with the peak wavelength at 880nm. The LEDs were placed in concentric circles as shown in Figure 4.2. It had been shown in [2] that concentric array of LEDs is accurate enough and has good intensity distribution. The first circle uses 5 LEDs, the middle circle uses 6 LEDs and the outer circles uses 7 circles. In the soldering process, 18 LEDs in circle of 6 LEDs were connected in parallel with battery.
This was done to ensure the brightness of every LEDs or every concentric circle of LEDs have the same intensity and brightness because the current was the same for three circles. A potentiometer was used to control the brightness of the LEDs so that they would not cause glare on the image. The camera was placed in the middle of the LEDs circles. User’s hands were required to be located at distance of approximately 30cm below the infrared radiation source and camera in order to ensure that a full hand image was captured. In this system, only left hand vein was captured. Image captured was in colored jpeg format, 320 pixels x 240 pixels with 24 bits per pixels. Figure 4.3 shows the example of hand vein image captured by the device.

Figure 5.2 IR LEDs arranged in concentric manner.

Figure 5.3 Sample of captured hand vein image.
5.1.2 Image Pre-processing Module

Several image pre-processing algorithms had been implemented in this system before the extraction of biometric feature for database and authentication purpose. The image captured was in JPEG format and therefore the very first step was to convert the raw image into BMP format. The other image pre-processing algorithms are median filtering, Canny edge detection, ROI detection, local thresholding, opening, closing and thinning. Behavioral flow chart and C code fragment of each algorithm are presented. For full coding, please refer to Appendix A.

(a) JPG to BMP conversion

The image was converted from colour image of 24 bits per pixel in JPEG format to grayscale image of 8 bits per pixel in BMP format using libjpeg. By this conversion, the image in BMP is easier to be manipulated than it is in JPEG image. The code fragment is shown in Figure 5.4.

```c
#include ".jpeg/djpeg.h"
...
...
char *inCommand[6]= {"./djpeg", "-bmp", "-grayscale", "-outfile", sUserImgOutDir,
                    sUserImgInDir};
decompress(6, inCommand);
```

Figure 5.4 Code fragment of jpg to bmp conversion.
Median filtering had been done on the image to filter the noise of the image before going through other processes.

![Behavioral flow chart and C code fragment of median filtering](attachment:image.png)

Figure 5.5 Behavioral flow chart and C code fragment of median filtering
Canny edge detection was performed in order to get the hand boundary so that ROI detection can be easily applied. Figure 4.6 shows the behavioral flow chart and C code fragments of this process.

```c
void cannyEdgeDetector(SDL_Surface *surfaceIn, SDL_Surface *surfaceOut, int lowT, int highT, unsigned int sig)
{
    unsigned int i=0, x=0, y=0; //looping parameter
    ...
    unsigned int width = surfaceIn->w;
    unsigned int height = surfaceIn->h;
    int **arrayOriPad;
    arrayOriPad = (int **) malloc((width+6)*sizeof(int));
    arrayOriPad[i] = (int *) malloc((height+6)*sizeof(int));
    ...
    int dG[7] = { -0133, -1080, -2420, 0, 2420, 1080, 0133 };
    sy[x][y] = (dG[0]*arrayOriPad[x-3+3][y+3]+...
    ...
    sx[x][y] = (dG[0]*arrayOriPad[x+3][y-3+3]+...
    ...
    sNorm[x][y] = (sx[x][y]*sx[x][y]+sy[x][y]*sy[x][y])/10000;
    ...
    int gDiff1 = 0, gDiff2 = 0;
    sAngle = (arctan2(sy[x][y],sx[x][y])*57)/(10000*10000);
    if (sAngle < 0) sAngle = sAngle+180;
    ...
    if (sAngle > 135) sDiscreteAngles[x][y] = 4;
    ...
    if (x==0) sDiscreteAngles[x][y] = 0;
    ...
    ...
    if (sNorm[x][y] >= lowT && sNorm[x][y] >= highT)
    {
        arrayOriPad[x+1][y+1] = 1; //putpixel(surfaceOut, x, y, 255);
    }
    ...
    ...
}
```

Figure 5.6 Behavioral flow chart and C code fragment of Canny edge detection.
ROI detection was performed on the hand edge image by determination of the centroid coordinate of the image first. X coordinate and Y coordinate value of the centroid of image was 170 pixel positive offset from the X coordinate of the tip of middle finger and equal to the Y value of the wrist of midpoint of wrist respectively. A 70x70 pixel image was then cropped based on the centroid of image.

Figure 5.7 Behavioral flow chart and C code fragment of ROI detection.
A threshold was applied on the image to binarize the image to outline the vein pattern.

```c
void medianFilterRGB (SDL_Surface *surfaceIn, SDL_Surface *surfaceOut, unsigned int wSize, unsigned int repeat) {
    ...  
    unsigned int i=0, j=0, x=0, y=0, loop=0;  // looping parameter
    ...  
    unsigned int width = surfaceIn->w;
    unsigned int height= surfaceIn->h;
    unsigned int **arrayOriPad;
    arrayOriPad = (unsigned int **) malloc((padW)*sizeof(unsigned int));
    arrayOriPad[i] = (unsigned int *) malloc ( (padH) * sizeof(unsigned int) );
    ...  
    unsigned int padBoth = wSize-1;
    unsigned int padSingle = padBoth/2;
    unsigned int padW = width + padBoth;
    unsigned int padH = height + padBoth;
    unsigned int wArea = wSize*wSize;
    ...  
    sum+=arrayOriPad[x+i][y+j];
    ...  
    mean=sum/wArea;

    pixelOut= (arrayOriPad[x+padSingle][y+padSingle]<=mean)?0:255;
    putpixel(surfaceOut, x, y, pixelOut);
    ...  
}
```

Figure 5.8 Behavioral flow chart and C code fragment of local thresholding.
(f) Opening and Closing

One of the morphology based operation which is opening and closing was applied on the image. Opening can separate the connected vein pattern in the image while the closing can fill in small hole in the vein image. Behavioral flow chart and C code fragment for opening and closing are shown in Figure 5.8 and 5.9 respectively.

![Behavioral flow chart and C code fragment of opening and closing.](image)

Figure 5.9 Behavioral flow chart and C code fragment of opening.
Figure 5.10 Behavioral flow chart and C code fragment of closing.
(g) Thinning

To extract the skeleton image of the vein texture which consists only a single pixel wide texture, a fast parallel algorithm for thinning digital patterns proposed by T.Y Zhang and C.Y Suen [12] was used in this project.

Figure 5.11 Behavioral flow chart and C code fragment of thinning algorithm.
5.1.3 Feature extraction module

The technique used in feature extraction module is the minutiae extraction. Only the bifurcation and ending point were extracted.

![Behavioral flow chart and C code fragment for minutiae extraction.](image)

```c
void minutiaeExtraction(SDL_Surface *surfaceIn, SDL_Surface *surfaceOut, const char *outFileInfo) {
    ...
    unsigned int oriPixel, zeroOnePixel;
    unsigned int Cn;
    unsigned int padW = width+2;
    unsigned int padH = height+2;
    int P[9];
    ...
    ...
    ...
    zeroOnePixel = oriPixel&1;
    arrayMinutiaeOri[x][y]=zeroOnePixel;
    ...
    ...
    Cn += abs(P[i]-P[i+1]);
    ...
    Cn += abs(P[8]-P[1]);
    ...
    ...
    if (Cn == 2 || Cn == 3)
        minutiae[count].type = endRidge;
        minutiae[count].x = x-1;
        minutiae[count].y = y-1;
        ...
    ...
    if (Cn == 6 || Cn == 7)
        minutiae[count].type = biRidge;
        minutiae[count].x = x-1;
        minutiae[count].y = y-1;
        ...
    ...
    ...
}
```

Figure 5.12 Behavioral flow chart and C code fragment for minutiae extraction.
5.1.4 Template of minutiae point

The minutiae extracted from the image was saved as template with the type, x and y location of the minutiae point. Total minutiae points was also saved at the end of the template.

```c
FILE *outFileMinutiae;
outFileMinutiae = fopen(outFileInfo,"w+");

for (i=0; i<count; i++) {
    fprintf(outFileMinutiae,"%d\n",minutiae[i].type);
    fprintf(outFileMinutiae,"%d\n",minutiae[i].x);
    fprintf(outFileMinutiae,"%d\n",minutiae[i].y);
}
fprintf(outFileMinutiae,"%d",count);
fclose(outFileMinutiae);
```

Figure 5.13 C code fragment of minutiae template.
5.1.5 Authentication Module – Matching of template

In this project, matching module was done based on distances to match the vein images. The similarity of two sets of point is compared using MHD algorithms.

![Behavioral Flow Chart and C Code Fragment of MHD](image)

```c
int calculateHD(const char *inFile1, const char *inFile2,
                unsigned int alignment)
{
    unsigned int totalPoint1;
    struct minutiaeRecord candidate1[lineOfData];
    unsigned int totalPoint2;
    struct minutiaeRecord candidate2[lineOfData];
    unsigned int hDirectedAB, hDirectedBA,
    hausdorffDistance;
    ...
    ...
    totalPoint1 = readMinutiaeRecord(inFile1, candidate1);
    unsigned int arrayRidgeX1[totalPoint1],
    arrayRidgeY1[totalPoint1];
    ...
    ...
    totalPoint2 = readMinutiaeRecord(inFile2, candidate2);
    unsigned int arrayRidgeX2[totalPoint2],
    arrayRidgeY2[totalPoint2];
    ...
    ...
    readRidgePoint2 (candidate1, totalPoint1, arrayRidgeX1,
    arrayRidgeY1);
    readRidgePoint2 (candidate2, totalPoint2, arrayRidgeX2,
    arrayRidgeY2);
    ...
    ...
    hDirectedAB = findDirectedMHD (arrayRidgeX1, arrayRidgeY1,
    totalPoint1, arrayRidgeX2, arrayRidgeY2, totalPoint2);
    hDirectedBA = findDirectedMHD (arrayRidgeX2, arrayRidgeY2,
    totalPoint2, arrayRidgeX1, arrayRidgeY1, totalPoint1);
    if (hDirectedAB<hDirectedBA) hausdorffDistance =
    hDirectedAB;
    else hausdorffDistance = hDirectedBA;
    return hausdorffDistance;
}
```

Figure 5.14 Behavioral flow chart and C code fragment of MHD.
5.1.6 Threshold and EER calculation

A threshold value is calculated for the templates in database for matching purpose. EER is calculated as well to determine the accuracy of the system. The formula for threshold and EER calculation are shown in the C code fragment below.

```c
void db_mhd(unsigned int alignment)
{
    ... ...
    meanClient=meanClient-valueClient[countClient];
    meanClient=meanClient/countClient;
    ... ...
    stdevClient += (meanClient - valueClient[i])*(meanClient - valueClient[i]);
    stdevClient = sqrt(stdevClient/(countClient-1));
    ... ...
    meanImposter=meanImposter-valueImposter[countImposter];
    meanImposter=meanImposter/countImposter;
    ... ...
    stdevImposter += (meanImposter - valueImposter[i])*(meanImposter - valueImposter[i]);
    stdevImposter = sqrt(stdevImposter/(countImposter-1));
    ... ...
    threshold =
                 (meanClient*stdevImposter)+(meanImposter*stdevClient) ) /
                 (stdevClient+stdevImposter);
    ... ...
    tempClient = (float(threshold)-
                  meanClient)/(stdevClient*1.4142);
    tempImposter = (float(threshold)-
                  meanImposter)/(stdevImposter*1.4142);
    ... ...
    erf = 1.128379167*(tempImposter - pow(tempImposter,3)/3 +
                  pow(tempImposter,5)/10 -pow(tempImposter,7)/42 +
                  pow(tempImposter,9)/216 - pow(tempImposter,11)/1320 +
                  pow(tempImposter,13)/9360 -pow(tempImposter,15)/75600);
    EER = (0.5 *(1+erf))*100;
    ... ...
```

Figure 5.15 Behavioral flow chart and C code fragment of threshold and EER calculation.
5.2 System demonstration

System prototype demonstration was done to show the functionality of the system. The system is run in terminal. There are 5 selections for user, which are conversion of image from jpeg to bmp, enrolment I (template generation), enrolment II (calculate threshold and EER), authentication and exit. When option 1 is chosen, the system will process the image of candidate from jpg to bmp format. Figure 5.15 below shows the menu and demonstration of selection 1.

![Figure 5.15 The main menu and demonstration of selection 1.](image)

Template generation is one of the procedures in enrolment module. When selection 2 is chosen, the image pre-processing and feature extraction processes will be implemented on the input images. Execution time for each process in second (ms) and clock cycle (cc) are shown. Figure 5.16 shows the demonstration of selection 2.
The third selection for user as shown in main menu is the enrolment II which is the calculation of threshold and EER for the system. This selection has to be run once after new user is added into the database. The new threshold value and EER for the system has to be calculated if there is new user added in the database. To get the threshold value of the system, mean and standard deviation of clients and imposters have to be calculated. The images are match with every other images in the database to get the client and imposter scores. Figure 5.16 shows the matching of the images. The calculation of threshold value and EER are shown in Figure 5.17.
Figure 5.18 Matching of images to get the client and imposter scores.

Figure 5.19 Calculation of threshold and EER.
For authentication module, the comparison of user image with the template in database is carried out to get the similarity of the image based on their distance. The matching score for user image is calculated and compared with threshold value, the user image is claimed to be matched with the database input image if the matching score is lower than the threshold. Examples of authentication process and the database list are shown in Figure 5.18 to 5.20.

Figure 5.20 Example of successful authentication.
Figure 5.21 Example of failed authentication.
Figure 5.22 Database list of the system.
CHAPTER 6

RESULTS AND DISCUSSION

This chapter presents the output image of image pre-processing module and feature extraction module. The performance analysis in terms of accuracy and speed are also discussed in this chapter.

6.1 Image Pre-processing Module

There were total of 8 processes done on the input image in image pre-processing module. Figure 6.1 shows the raw image captured by the low cost image capturing device. Every raw image was converted to bmp format before going through other image pre-processing processes. Figure 6.2 shows the image after jpg to bmp conversion. The image is then used as input image in image pre-processing module. The processes in image pre-processing module include median filtering for colour image, Canny edge detection, local thresholding, ROI detection, median filtering for binary image, opening, closing and thinning. The output images of each process of image pre-processing module are shown in Figure 6.3.
Figure 6.1 Captured raw image in jpeg format.

Figure 6.2 Input image in bmp format.
Figure 6.3 Output image for each process in image pre-processing module.
Table 6.1 Detailed techniques and output images in image pre-processing module.

<table>
<thead>
<tr>
<th>NO.</th>
<th>TECHNIQUES</th>
<th>OUTPUT IMAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>Median Filtering</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reduce the noise in image.</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td><strong>Canny Edge Detection</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Get the boundary of hand image so that ROI detection can be easily applied.</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td><strong>ROI detection</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Determination of the centroid coordinate of the image.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- X coordinate: 170 pixel positive offset from the X coordinate of the tip of middle finger.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Y coordinate: the Y value of the wrist of midpoint of wrist.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. A 70x70 pixel image was then cropped based on the centroid of image.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Operation</strong></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4</td>
<td><strong>Local Thresholding</strong></td>
<td>To outline the vein pattern</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><strong>Median filtering</strong></td>
<td>Reduce the noise in the vein image.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td><strong>Opening</strong></td>
<td>Enhance the vein pattern by clearing unwanted edge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td><strong>Closing</strong></td>
<td>Further enhance the vein image by filling up small hole in the vein pattern if any.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td><strong>Thinning</strong></td>
<td>Skeletonize the vein pattern to get one pixel wide texture.</td>
</tr>
</tbody>
</table>
6.2 Feature Extraction Module

The technique used in feature extraction module was minutiae extraction proposed by L.Y. Wang et. al. Figure 6.4 shows the output image of feature extraction module. Dot is denoted as ending point and cross is denoted as bifurcation point.

![Output image of minutiae extraction](image)

Figure 6.4 Output image of minutiae extraction

6.3 Template generation

A template was generated for each processed image which consisted of type of minutiae point and x and y location of the point. Figure 6.5 shows the sample of template.
Figure 6.5 Sample of template

6.4 Performance analysis

The performance of the implementation of hand vein biometric authentication on FPGA-based embedded system was evaluated in terms of accuracy and speed. There are a few performance measurement conducted as discussed in the following section.
6.4.1 Accuracy

Equal Error Rate (EER) is defined as the rate at which both accept and reject errors are equal. A biometric security system predetermines the threshold values for its false acceptance rate and its false rejection rate, and when the rates are equal, the common value is referred to as the equal error rate. The lower the equal error rate value, the higher the accuracy of the biometric system.

There are a total of 120 infrared hand vein images in the database. Four images each were taken from 30 individuals. The system experiment on 30 candidates show promising result of EER = 3.68%.

6.4.2 Speed

The hand vein biometric authentication embedded system was downloaded in Altera Nios II prototyping system with the use of Altera Stratix II Board. The system is running with 100MHz frequency and applied using RTOS which is Nios2-Linux. The execution time was calculated for each process done in the embedded system. Table 6.2 shows the execution time in second (s) and clock cycle (cc) for each image pre-processing processes and minutiae extraction process. The total execution times for image pre-processing and feature extraction are 13.31s. Canny edge detection process took the longest execution time which was 52.37% of total execution time. This was due to the large size of image and the algorithm had to run through every pixel in the image. Median filter process also took long execution time which was 39.29% of total execution time due to the same problem as canny edge detection process.
<table>
<thead>
<tr>
<th>Image Pre-processing</th>
<th>Time (s)</th>
<th>Clock Cycle (10^6 cc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Filter</td>
<td>5.23</td>
<td>524</td>
</tr>
<tr>
<td>Canny Edge Detection</td>
<td>6.97</td>
<td>697</td>
</tr>
<tr>
<td>ROI detection</td>
<td>0.5</td>
<td>51</td>
</tr>
<tr>
<td>Thresholding</td>
<td>0.11</td>
<td>11</td>
</tr>
<tr>
<td>Median Filter</td>
<td>0.17</td>
<td>17</td>
</tr>
<tr>
<td>Opening</td>
<td>0.07</td>
<td>7</td>
</tr>
<tr>
<td>Closing</td>
<td>0.13</td>
<td>13</td>
</tr>
<tr>
<td>Thinning</td>
<td>0.11</td>
<td>11</td>
</tr>
<tr>
<td>Minutiae Extraction</td>
<td>0.02</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>13.31</strong></td>
<td><strong>1333</strong></td>
</tr>
</tbody>
</table>

Table 6.2 Execution time for each image pre-processing blocks.
CHAPTER 7

CONCLUSION

This chapter discusses the contribution and limitation of this project. Recommendation for future work to improve the system is also discussed here.

7.1 Contribution

A hand vein biometric authentication system has been successfully implemented on FPGA-based embedded system. The system has been successfully run on RTOS Nios2-linux using Altera Stratix II board. This system can be applied for personal authentication or security clearance purpose. It can be implemented as attendance registration system for offices, schools, libraries, hotel rooms, apartments and other places. This system implements online recognition process. Therefore, it can be used to capture user’s image on the spot and register the user instantly.

This system can be integrated with other biometric system such as face and palm print recognition to increase the security level for higher security purpose. This
system can be further improved to provide the required level of security to allow access of only authorized personnel into places of utmost importance such as armory, nuclear missile control room and bank vaults.

7.2 Limitation

1. The peak wavelength of the infrared LEDs used in this project is 880nm. However, lighting should be performed under a very tight optical window namely 740nm to 760nm so that the deoxidized hemoglobin [Hb] in the vein vessels absorbs completely the radiation.

2. The system did not take into account rotation of hand position during image capturing process. It will cause inaccuracy in template generation and matching module.

3. ROI is too small to contain all the important information of the vein in the back of hand.

4. User has to fix their hand position during image capturing process.

5. The time taken to process the image on embedded system is long especially for median filtering and Canny edge detection. It will take very long time to process every image when the database is big.
7.3 Recommendation for future work

1. A better camera with higher pixel can be used to improve the quality of image.

2. The infrared LEDs can be replaced by infrared LEDs with peak wavelength at 740nm to 760nm.

3. A boundary tracking algorithm can be applied to trace along the hand contour and detect the valley points for ROI detection.

4. The image pre-processing algorithms can be written in hardware to increase the speed of image processing.
REFERENCES


This appendix contains the source code corresponding to Chapter 5. Due to space constraints, only simplified C programming codes are shown.

```c
#include "0handVein.h"

/***********************************************Canny Edge Detection ***********************************************
//INPUT: 'surfaceIn', 'lowT' low threshold, 'highT' high thresh, 'sig' sigma
//OUTPUT: 'surfaceOut'
void cannyEdgeDetector(SDL_Surface *surfaceIn, SDL_Surface *surfaceOut, int lowT, int highT, unsigned int sig)
{
    lowT=10000*lowT*lowT;
    highT=10000*highT*highT;

    //get the image size
    unsigned int width = surfaceIn->w;
    unsigned int height= surfaceIn->h;

    //looping parameters
    unsigned int i=0, x=0, y=0;

    //other parameter
    int dG[7]=-{-0133,-1080,-2420,0,2420,1080,0133};
    int gDiff1=0, gDiff2=0;
    unsigned int count=0;

    //declare array using malloc and free
    int **arrayOriPad;
    arrayOriPad= (int **) malloc ( (width+6)*sizeof(int) );
    for (i = 0; i < width+6; i++)
        arrayOriPad[i] = (int *) malloc ( (height+6) * sizeof(int) );
    long long int **sy;
```
long long int **sx;
long long int **sNorm; //norm of gradient
long long int sAngle; //direction of gradient
long long int sSlope;
unsigned int **sDiscreteAngles;
sy = (long long int **) malloc ( (width)*sizeof(long long int) );
sx = (long long int **) malloc ( (width)*sizeof(long long int) );
sNorm = (long long int **) malloc ( (width)*sizeof(long long int) );
sDiscreteAngles= (unsigned int **) malloc ( (width)*sizeof(unsigned int) );

for (i = 0; i < width; i++){
    sy[i] = (long long int *) malloc ( (height) * sizeof(long long int));
sx[i] = (long long int *) malloc ( (height) * sizeof(long long int));
sNorm[i] = (long long int *) malloc ( (height) * sizeof(long long int));
sDiscreteAngles[i]=(unsigned int*)malloc((height)*sizeof(unsigned int));
}

//initialize array
for (x=0; x<width+6; x++){  
    for (y=0; y<height+6; y++){
        arrayOriPad[x][y]=0;
    }
}

for (x=0; x<width; x++){
    for (y=0; y<height; y++){
        arrayOriPad[x][y]=0;
    }
}

//get pixel from surfaceIn
for (x=0; x<width; x++) {  
    for (y=0; y<height; y++) { 
        arrayOriPad[x+3][y+3]=getpixel(surfaceIn, x, y);
    }
}

for (x=0; x<width; x++) {
    for (y=0; y<height; y++) {
        sy[x][y]=(dG[0]*arrayOriPad[x-3][y+3]+dG[1]*arrayOriPad[x-2][y+3]+dG[2]*arrayOriPad[x-1][y+3]+dG[4]*arrayOriPad[x+1][y+3]+dG[5]*arrayOriPad[x+2][y+3]+dG[6]*arrayOriPad[x+3][y+3]);
        sx[x][y]=(dG[0]*arrayOriPad[x+3][y-3]+dG[1]*arrayOriPad[x+3][y-2]+dG[2]*arrayOriPad[x+3][y-1]+dG[4]*arrayOriPad[x+3][y+1]+dG[5]*arrayOriPad[x+3][y+2]+dG[6]*arrayOriPad[x+3][y+3]);
    }
}

//initialize array
for (x=0; x<width+6; x++){
    for (y=0; y<height+6; y++){
        arrayOriPad[x][y]=0;
    }
}
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
    {
        sNorm[x][y]= (sx[x][y]*sx[x][y] + sy[x][y]*sy[x][y]) / 10000;
        sAngle= ( arctan2( sy[x][y],sx[x][y] ) * 57) / (10000 * 10000);
        if (sAngle < 0)
            sAngle = sAngle + 180;

        // sDiscreteAngles
        if (sAngle < 45) sDiscreteAngles[x][y] = 1;
        if (sAngle >= 45 && (sAngle < 90)) sDiscreteAngles[x][y] = 2;
        if (sAngle >= 90 && (sAngle < 135)) sDiscreteAngles[x][y] = 3;
        if (sAngle > 135) sDiscreteAngles[x][y] = 4;

        // set the boundary pixels to 0, so we don't count them in analysis...
        if (x == 0) sDiscreteAngles[x][y] = 0;
        if (x == width - 1) sDiscreteAngles[x][y] = 0;
        if (y == 0) sDiscreteAngles[x][y] = 0;
        if (y == height - 1) sDiscreteAngles[x][y] = 0;
    }

for (x=0; x<width; x++)
    for (y=0; y<height; y++)
    {
        if (sDiscreteAngles[x][y] == 1)
        {
            if (sx[x][y] != 0) {
                sSlope = abs(sy[x][y] / sx[x][y]);
                gDiff1 = sSlope * (sNorm[x][y] - sNorm[x+1][y+1]) + (1 - sSlope) * (sNorm[x][y] - sNorm[x+1][y+1]);
                gDiff2 = sSlope * (sNorm[x][y] - sNorm[x-1][y-1]) + (1 - sSlope) * (sNorm[x][y] - (int)sNorm[x-1][y-1]);
                if ( (gDiff1 >= 0) && (gDiff2 >= 0) )
                    if (sNorm[x][y] >= lowT && sNorm[x][y] >= highT)
                        arrayOriPad[x+1][y+1] = 1;
                        count++;
                    else
                        arrayOriPad[x+1][y+1] = 2;
            }
        }

        else if (sDiscreteAngles[x][y] == 2)
        {
            if (sx[x][y] == 0) {
                gDiff1 = sNorm[x][y] - sNorm[x+1][y];
                gDiff2 = sNorm[x][y] - sNorm[x-1][y];
            }
            else {
                sSlope = abs(sy[x][y] * 100000 / sx[x][y]);
                gDiff1 = (100000 / sSlope) * (sNorm[x][y] - sNorm[x+1][y+1]) + (1 - (100000 / sSlope)) * (sNorm[x][y] - (int)sNorm[x+1][y]);
            }
        }
    }
gDiff2=(100000/sSlope)*(sNorm[x][y]-sNorm[x-1][y-1]) + (1-(100000/sSlope))*(sNorm[x][y]-(int)sNorm[x-1][y]);

if ((gDiff1>=0) && (gDiff2>=0))
{
    if (sNorm[x][y]>=lowT && sNorm[x][y]>=highT)
    {
        //putpixel(surfaceOut, x, y, 255);
        arrayOriPad[x+1][y+1]=1;
        count++;
    }
    else
    {
        arrayOriPad[x+1][y+1]=2;
        //putpixel(surfaceOut, x, y, 2);
    }
}
else if (sDiscreteAngles[x][y]==3)
{
    if (sx[x][y]==0) {
        gDiff1=sNorm[x][y]-sNorm[x+1][y];
        gDiff2=sNorm[x][y]-sNorm[x-1][y];
    }
    else{
        sSlope=abs(sy[x][y]*100000/sx[x][y]);
        gDiff1=(100000/sSlope)*(sNorm[x][y]-sNorm[x+1][y-1]) + (1-(100000/sSlope))*(sNorm[x][y]-(int)sNorm[x+1][y]);
        gDiff2=(100000/sSlope)*(sNorm[x][y]-sNorm[x-1][y+1]) + (1-(100000/sSlope))*(sNorm[x][y]-(int)sNorm[x-1][y]);
    }
    if ((gDiff1>=0) && (gDiff2>=0))
    {
        if (sNorm[x][y]>=lowT && sNorm[x][y]>=highT)
        {
            //putpixel(surfaceOut, x, y, 255);
            arrayOriPad[x+1][y+1]=1;
            count++;
        }
        else
        {
            arrayOriPad[x+1][y+1]=2;
            //putpixel(surfaceOut, x, y, 2);
        }
    }
}
else if (sDiscreteAngles[x][y]==4)
{
    if (sx[x][y]!=0) {
        sSlope=abs(sy[x][y]/sx[x][y]);
        gDiff1=(sSlope)*(sNorm[x][y]-sNorm[x+1][y-1]);
        gDiff2=(sSlope)*(sNorm[x][y]-sNorm[x-1][y+1]) + (1-(sSlope))*(sNorm[x][y]-sNorm[x][y-1]);
        if ((gDiff1>=0) && (gDiff2>=0))
        {
            
        }
}
if (sNorm[x][y] >= lowT && sNorm[x][y] >= highT) {
    //putpixel(surfaceOut, x, y, 255);
    arrayOriPad[x+1][y+1] = 1;
    count++;
} else {
    //putpixel(surfaceOut, x, y, 2);
    arrayOriPad[x+1][y+1] = 2;
}
}
}

while (count != 0) {
    count = 0;
    for (x = 0; x < width; x++) {
        for (y = 0; y < height; y++) {
            if (arrayOriPad[x+1][y+1] == 1) {
                if (arrayOriPad[x+1+1][y+1+1] == 2) {
                    //putpixel(surfaceOut, x+1, y+1, 255);
                    arrayOriPad[x+1+1][y+1+1] = 1;
                    count++;
                }
                if (arrayOriPad[x+1+1][y+1] == 2) {
                    //putpixel(surfaceOut, x+1, y, 255);
                    arrayOriPad[x+1+1][y+1] = 1;
                    count++;
                }
                if (arrayOriPad[x+1+1][y+1-1] == 2) {
                    //putpixel(surfaceOut, x+1, y-1, 255);
                    arrayOriPad[x+1+1][y+1-1] = 1;
                    count++;
                }
                if (arrayOriPad[x+1][y+1+1] == 2) {
                    //putpixel(surfaceOut, x, y+1, 255);
                    arrayOriPad[x+1][y+1+1] = 1;
                    count++;
                }
                if (arrayOriPad[x+1][y+1-1] == 2) {
                    //putpixel(surfaceOut, x, y-1, 255);
                    arrayOriPad[x+1][y+1-1] = 1;
                    count++;
                }
                if (arrayOriPad[x+1-1][y+1+1] == 2) {
                    //putpixel(surfaceOut, x-1, y+1, 255);
                    arrayOriPad[x+1-1][y+1+1] = 1;
                    count++;
                }
                if (arrayOriPad[x+1-1][y+1] == 2) {
                    //putpixel(surfaceOut, x-1, y, 255);
                    arrayOriPad[x+1-1][y+1] = 1;
                    count++;
                }
            }
        }
    }
}
count++;
}

if( arrayOriPad[x+1][y+1]==2 ) {
    //putpixel(surfaceOut, x-1, y-1, 255);
    arrayOriPad[x+1][y+1]=1;
    count++;
}
}
}

for (x=0; x<width; x++) {
    for (y=0; y<height;y++) {
        if( arrayOriPad[x+1][y+1]==1 ) putpixel(surfaceOut, x, y, 255);
        else putpixel(surfaceOut, x, y, 0);
    }
}

//free array
for (i = 0; i < width+6; i++)
    free(arrayOriPad[i]);
free(arrayOriPad);

for (i = 0; i < width; i++){
    free(sy[i]);
    free(sx[i]);
    free(sNorm[i]);
    //free(sAngle[i]);
    //free(sSlope[i]);
    free(sDiscreteAngles[i]);
    free(sy);
    free(sx);
    free(sNorm);
    //free(sAngle);
    //free(sSlope);
    free(sDiscreteAngles);
}

/****************************MEDIANFILTERRGB**************************/

//Median Filter - reduce noise
//INPUT: 'surfaceIn', 'wSize' size of nxn window, 'repeat' iteration for median filter
//OUTPUT: 'surfaceOut'

void medianFilterRGB (SDL_Surface *surfaceIn, SDL_Surface *surfaceOut, unsigned int wSize, unsigned int repeat) {

    //get the image size
    unsigned int width = surfaceIn->w;
    unsigned int height= surfaceIn->h;
    //looping parameters
    unsigned int i=0,j=0, x=0,y=0,loop=0;
    //unsigned in t k=0; //delete:2009.10.20

    //window size
    wSize = (wSize%2)?$size:wSize+1;    //make sure window size is odd
    unsigned int wArea = wSize*wSize;    //window area

    //padding

unsigned int padBoth = wSize - 1;
unsigned int padSingle = padBoth / 2;
unsigned int padW = width + padBoth;
unsigned int padH = height + padBoth;

arrayOriPad = (unsigned int **) malloc ((padW) * sizeof(unsigned int));
for (i = 0; i < padW; i++)
    arrayOriPad[i] = (unsigned int *) malloc ((padH) * sizeof(unsigned int));

//initialize array, to make sure padding is set to 0
for (x = 0; x < padW; x++)
    for (y = 0; y < padH; y++)
        arrayOriPad[x][y] = 0;

//store pixel in array
for (x = 0; x < width; x++)
    for (y = 0; y < height; y++)
        arrayOriPad[x + padSingle][y + padSingle] = getpixel(surfaceIn, x, y);

unsigned int count, out;
unsigned int *hist;
//array to keep value in the window
hist = (unsigned int *) malloc (256 * sizeof(unsigned int));

for (loop = 0; loop < repeat; loop++)
{
    for (x = 0; x < width; x++)
    {
        for (y = 0; y < height; y++)
        {
            //initialize
            if (y == 0)
            {
                for (i = 0; i < wSize; i++)
                    hist[i] = 0;
                for (i = 0; i < wSize; i++)
                    for (j = 0; j < wSize; j++)
                        hist[arrayOriPad[x + i][y + j]]++;
            }
            //add new data (row at y+wSize-1)
            else if (y != 0)
            {
                for (i = 0; i < wSize; i++)
                    hist[arrayOriPad[x + i][y + wSize - 1]]++;
            }
            count = 0;
            for (i = 0; i < 256; i++)
            {
                if (hist[i] > 0)
                    count = count + hist[i];
                if (count > ((wArea - 1) / 2))
                {
                    out = i;
                    goto finish;
                }
            }
            finish:
            putpixel(surfaceOut, x, y, out);
//delete old data (row at y)
for(i=0; i<wSize; i++)
    hist[arrayOriPad[x+i][y]]--;

}

if (repeat >1)
{
    //store new result into array, repeat median filter
    for (x=0; x<width; x++)
        for (y=0; y<height; y++)
            arrayOriPad[x+padSingle] [y+padSingle]= getpixel(surfaceOut, x, y);
}

// Free array
for (i = 0; i < padW; i++)
    free(arrayOriPad[i]);
free(arrayOriPad);

//free(arrayWindow);
free(hist);

//**********************************************************ROI DETECTION**********************************************************/
//surfaceIn-filled image (extracted finger region)
//surfaceOriGray-original grayscale image
//surfaceOut0-align filled image
//surfaceOut-align segmented grayscale image
void putBackAlignResize(SDL_Surface *surfaceIn, SDL_Surface *surfaceOriGray, SDL_Surface *surfaceOut0, SDL_Surface *surfaceOut, unsigned int refPointX, unsigned int refPointY)
{
    //size of original image (320*240)
    unsigned int width = surfaceIn->w;
    unsigned int height = surfaceIn->h;

    //size of resize image (320*150)
    unsigned int widthResize = surfaceOut->w;
    unsigned int heightResize = surfaceOut->h;

    //looping parameter
    unsigned int i=0, x=0, y=0;
    int x1=0,y1=0;    //negative, use int (cant reuse x & y)

    //find reference point for finger
    unsigned int fTipCoorX=0, coorY[2]={0,0},middleX=0, middleY=0;
    //delta x & y for alignment
    int tx, ty;

    unsigned int **arraySurfaceIn, **arraySegment;
    unsigned int **arraySurfaceInAligned, **arraySegmentAligned;

    arraySurfaceIn= (unsigned int **) malloc ( (width)*sizeof(unsigned int) );
    arraySegment= (unsigned int **) malloc ( (width)*sizeof(unsigned int) );
    arraySurfaceInAligned= (unsigned int **) malloc ( (width)*sizeof(unsigned int) );
    arraySegmentAligned= (unsigned int **) malloc ( (width)*sizeof(unsigned int) );
for (i = 0; i < width; i++)
{
    arraySurfaceIn[i] = (unsigned int *) malloc ( (height) * sizeof(unsigned int) );
    arraySegment[i] = (unsigned int *) malloc ( (height) * sizeof(unsigned int) );
    arraySurfaceInAligned[i] = (unsigned int *) malloc ( (height) * sizeof(unsigned int) );
    arraySegmentAligned[i] = (unsigned int *) malloc ( (height) * sizeof(unsigned int) );
}

//initialize array
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
    {
        arraySurfaceIn[x][y]=0;
        arraySegment[x][y]=255;
        arraySurfaceInAligned[x][y]=0;
        arraySegmentAligned[x][y]=255;
    }

//FIND tx & ty FOR ALIGNMENT
//finger tip coordinate X detection
for (x1=width-10; x1>0; x1--)
    for (y=5; y<height-10; y++)
        if (getpixel(surfaceIn, x1, y)==255)
        {
            fTipCoorX=x1;
            printf("finger tip: %d",fTipCoorX);
            goto DETECTED;
        }

DETECTED:

middleX=fTipCoorX-170;

for (y=0; y<height; y++)
    if (getpixel(surfaceIn, middleX-70, y)==255)
    {
        coorY[0]=y;        //top coordinate y
        break;
    }

for (y1=height-1; y1>=0; y1--)
    if (getpixel(surfaceIn, middleX-70, y1)==255)
    {
        coorY[1]=y1;       //bottom coordinate y
        break;
    }

middleY=(coorY[1]+coorY[0])/2;

tx=(int)refPointX-(int)middleX;
ty=(int)refPointY-(int)middleY;

//FIND tx & ty FOR ALIGNMENT END
//********************************************************************************************
//PUT BACK ORI IMAGE, surfaceOriGray
for (x=0; x<width; x++)
for (y=0; y<height; y++)
{
    arraySurfaceIn[x][y]=getpixel(surfaceOriGray, x, y);
    arraySegment[x][y]=getpixel(surfaceOriGray, x, y);
}

//=================================================
//===TRANSLATE IMAGE

//TRANSLATE IMAGE
if ( (tx<0) && (ty<0) )
{
    tx=abs(tx);
    ty=abs(ty);
    for (x=0; x<width-tx; x++)
        for (y=0; y<height-ty; y++)
            
                arraySurfaceInAligned[x][y]=arraySurfaceIn[x+tx][y+ty];
            arraySegmentAligned[x][y]=arraySegment[x+tx][y+ty];

}
else if (tx<0)
{
    tx=abs(tx);
    for (x=0; x<width-tx; x++)
        for (y=0; y<height-ty; y++)
            
                arraySurfaceInAligned[x][y+ty]=arraySurfaceIn[x+tx][y];
            arraySegmentAligned[x][y+ty]=arraySegment[x+tx][y];

}
else if (ty<0)
{
    ty=abs(ty);
    for (x=0; x<width-tx; x++)
        for (y=0; y<height-ty; y++)
            
                arraySurfaceInAligned[x+tx][y]=arraySurfaceIn[x][y+ty];
            arraySegmentAligned[x+tx][y]=arraySegment[x][y+ty];

}
else
{
    for (x=0; x<width-tx; x++)
        for (y=0; y<height-ty; y++)
            
                arraySurfaceInAligned[x+tx][y+ty]=arraySurfaceIn[x][y];
            arraySegmentAligned[x+tx][y+ty]=arraySegment[x][y];

}

//===TRANSLATE IMAGE END

//=================================================
//===resize image

//resize image
for (x=0; x<widthResize; x++)
    for (y=0; y<heightResize; y++)
        
            putpixel(surfaceOut0, x,y,arraySurfaceInAligned[x][y]);
        putpixel(surfaceOut, x,y,arraySegmentAligned[x][y]);
free array
for (i = 0; i < width; i++)
{
    free(arraySurfaceIn[i]);
    free(arraySegment[i]);
    free(arraySurfaceInAligned[i]);
    free(arraySegmentAligned[i]);
}
free(arraySurfaceIn);
free(arraySegment);
free(arraySurfaceInAligned);
free(arraySegmentAligned);

} ****************************LOCALTHRESHOLDING ****************************/
//INPUT: 'surfaceIn', 'wSize' size of nxn window, 'k' constant
//OUTPUT: 'surfaceOut'
void localThresholding2 (SDL_Surface *surfaceIn, SDL_Surface *surfaceResize, SDL_Surface *surfaceOut, unsigned int wSize, unsigned int k)
{
    //get image size
    unsigned int width = surfaceIn->w;
    unsigned int height= surfaceIn->h;
    //looping parameter
    unsigned int i=0,j=0,x=0,y=0;

    unsigned int sum, mean, pixelOut;

    //window size
    wSize = (wSize%2)?wSize:wSize+1;       //make sure window size is odd
    unsigned int wArea = wSize*wSize;      //window area

    //add padding
    unsigned int padBoth     = wSize-1;
    unsigned int padSingle   = padBoth/2;
    unsigned int padW        = width + padBoth;
    unsigned int padH        = height + padBoth;

    //declare array using malloc & free
    unsigned int **arrayOriPad;        //array with pad to keep original image
    arrayOriPad= (unsigned int **) malloc ( (padW)*sizeof(unsigned int) );
    for (i = 0; i < padW; i++)
        arrayOriPad[i] = (unsigned int *) malloc ( (padH) * sizeof(unsigned int) );

    //initialize array, to make sure padding is set to 0
    for (x=0; x<padW; x++)
        for (y=0; y<padH; y++)
            arrayOriPad[x][y]=0;
        //arrayOriPad[x][y]=255;

    //store pixel in array
    for (x=0; x<width; x++)
        for (y=0; y<height; y++)
            arrayOriPad[x+padSingle][y+padSingle]=getpixel(surfaceIn, x, y);

    for (x=0; x<width; x++)
for (y=0; y<height; y++)
{
    //initialize
    if (y==0){
        sum=0;
        for(i=0; i<wSize; i++)
            for(j=0; j<wSize; j++)
                sum+=arrayOriPad [x+i][y+j];
    }
    //add new data (row at y+wSize-1)
    else if (y!=0)
        for(i=0; i<wSize; i++)
            sum+=arrayOriPad [x+i][y+wSize-1];
    mean=sum/wArea;
    pixelOut = (arrayOriPad [x+padSingle][y+padSingle]<=mean)? 0:255;
    putpixel(surfaceOut, x, y, pixelOut);
    //delete old data (row at y)
    for(i=0; i<wSize; i++)
        sum=sum-
            arrayOriPad [x+i][y];
}
signed int pixel;
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
    {
        pixel=getpixel(surfaceResize, x, y);
        if(pixel==0)
            //if(pixel==255)
            putpixel(surfaceOut, x, y, 255);
    }
//free array
for (i=0; i<padW; i++)
    free(arrayOriPad[i]);
free(arrayOriPad);

void medianFilter (SDL_Surface *surfaceIn, SDL_Surface *surfaceResize, SDL_Surface *surfaceOut,
        unsigned int wSize, unsigned int repeat)
{
    //get image size
    unsigned int width = surfaceIn->w;
    unsigned int height= surfaceIn->h;
    //looping parameter
    unsigned int i=0, j=0, k=0, x=0, y=0, loop=0;
    //other (result)
    unsigned int Out=0;
    //window size
    wSize = (wSize%2)?wSize:wSize+1;   //make sure window size is odd
    unsigned int wArea = wSize*wSize;   //window area

    //********************************MEDIAN FILTER********************************//
    //INPUT: 'surfaceIn', 'wSize' size of nxn window, 'repeat' iteration for median filter
    //OUTPUT: 'surfaceOut'

    unsigned int pixel;
//add padding
unsigned int padBoth = wSize-1;
unsigned int padSingle = padBoth/2;
unsigned int padW = width + padBoth;
unsigned int padH = height + padBoth;

//declare array using malloc & free
unsigned int **arrayOriPad;  //array with pad to keep original image
arrayOriPad = (unsigned int **) malloc ((padW)*sizeof(unsigned int));
for (i = 0; i < padW; i++)
    arrayOriPad[i] = (unsigned int *) malloc ((padH) * sizeof(unsigned int));

//initialize array, to make sure padding is set to 255
for (x=0; x<padW; x++)
    for (y=0; y<padH; y++)
        arrayOriPad[x][y]=255;

//store pixel in array
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
    {
        if(getpixel(surfaceResize, x, y)==0) //black
            arrayOriPad[x+padSingle][y+padSingle]=255;  //white
        else
            arrayOriPad[x+padSingle][y+padSingle]=getpixel(surfaceIn, x, y);
    }

for (loop=0; loop<repeat; loop++)
    {  //initialize
        for (y=0; y<height; y++)
            {
                if (y==0) {
                    k=0;
                    for(i=0; i<wSize; i++)
                        for(j=0; j<wSize; j++)
                            if( !(arrayOriPad[x+i][y+j]))
                                k++;
                }
               //add new data (row at y+wSize-1)
               else if (y!=0)
                    for(i=0; i<wSize; i++)
                        if( !(arrayOriPad[x+i][y+wSize-1]))
                            k++;

        Out = (k>=((wArea-1)/2) +1)?0:255;
        //Out = (k>=((wArea-1)/2))?0:255;
        putpixel(surfaceOut, x, y, Out);

        //delete old data (row at y)
        for(i=0; i<wSize; i++)
            if( !(arrayOriPad [x+i][y]))  k=k-1;
    }
unsigned int pixel;
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
    {
        pixel=getpixel(surfaceResize, x, y);
        if(pixel==0)
            //if(pixel==255)
            putpixel(surfaceOut, x, y, 255);
    }

    //free array
    for (i = 0; i < padW; i++)
        free(arrayOriPad[i]);
    free(arrayOriPad);

} //**********************************************************OPENING***************************************************

void opening(SDL_Surface *surfaceIn, SDL_Surface *surfaceOut, unsigned int wSize)
{
    wSize= (wSize%2)?wSize:wSize+1;         //to make sure is odd number
    unsigned int width = surfaceIn->w;
    unsigned int height = surfaceIn->h;
    unsigned int i,j,x,y;
    //add padding
    unsigned int padBoth = wSize-1;
    unsigned int padSingle = padBoth/2;
    unsigned int padW = width + padBoth;
    unsigned int padH = height + padBoth;
    unsigned int arrayOriPad [padW][padH];   //array with pad to keep original image with padding
    unsigned int arrayDilate [padW][padH];   //array with pad to keep image after dilation

    //result
    unsigned int oriPixel,zeroOnePixel, outPixel; //, out;

    //initialize array, to make sure padding is set to 0
    for (x=0; x<padW; x++)
        for (y=0; y<padH; y++)
            {
                arrayOriPad [x][y]=0;
                arrayDilate [x][y]=1;
            }
    //store pixel in array
    for (x=0; x<width; x++)
        for (y=0; y<height; y++)
Uint8 *pIn=(Uint8 *)surfaceIn->pixels + y * surfaceIn->pitch + x;
oriPixel=*pIn;
zeroOnePixel = oriPixel&1;
arrayOriPad [x+padSingle][y+padSingle]=zeroOnePixel;
arrayDilate [x+padSingle][y+padSingle]=zeroOnePixel;
}

//Dilation
//loop every pixel
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
    {
        if (arrayOriPad[x+padSingle][y+padSingle]==0)
        {
            outPixel=0;
            for (i=0; i<padBoth; i++)
                for (j=0; j<padBoth; j++)
                    outPixel=outPixel||arrayOriPad[x+i][y+j];
            arrayDilate
            [x+padSingle][y+padSingle]=outPixel;
        }
    }

//Erosion
//loop every pixel
for (x=0; x<width; x++)
    {for (y=0; y<height; y++)
    {
        if (arrayDilate[x+padSingle][y+padSingle]==1)
        {
            outPixel=1;
            for (i=0; i<padBoth; i++)
                for (j=0; j<padBoth; j++)
                    outPixel=outPixel&&arrayDilate[x+i][y+j];
            Uint8 *pOut = (Uint8 *)surfaceOut->pixels + y * surfaceOut->pitch + x;
            if (outPixel==1)
                *pOut = 255;
            else
                *pOut = 0;
        }
    }
}

void closing(SDL_Surface *surfaceIn, SDL_Surface *surfaceOut, unsigned int wSize)
{
    wSize = (wSize%2)?wSize:wSize+1;
    unsigned int width = surfaceIn->w;
    unsigned int height= surfaceIn->h;
    unsigned int i,j,x,y;
    //add padding

    /**************************************************CLOSING**************************************************/

unsigned int padBoth = wSize - 1;
unsigned int padSingle = padBoth / 2;
unsigned int padW = width + padBoth;
unsigned int padH = height + padBoth;

unsigned int arrayOriPad[padW][padH]; //array with pad to keep original image with padding
unsigned int arrayErode[padW][padH]; //array with pad to keep image after dilation

//initialize array, to make sure padding is set to 0
for (x=0; x<padW; x++)
    for (y=0; y<padH; y++)
        {
            arrayOriPad[x][y]=1;
            arrayErode[x][y]=0;
        }

//store pixel in array
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
        {
            Uint8 *pIn=(Uint8 *)surfaceIn ->pixels + y * surfaceIn ->pitch + x ;
            oriPixel=*pIn;
            zeroOnePixel = oriPixel&1;
            arrayOriPad[x+padSingle][y+padSingle]=zeroOnePixel;
            arrayErode[x+padSingle][y+padSingle]=zeroOnePixel;
        }

//Erosion
//loop every pixel
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
        {
            if (arrayOriPad[x+padSingle][y+padSingle]==1)
                {
                    outPixel=1;
                    for (i=0; i<padBoth; i++)
                        for (j=0; j<padBoth; j++)
                            outPixel=outPixel&&arrayOriPad[x+i][y+j];
                    arrayErode[x+padSingle][y+padSingle]=outPixel;
                }
        }

//Erosion
//loop every pixel
for (x=0; x<width; x++)
    for (y=0; y<height; y++)
        {
            if (arrayErode[x+padSingle][y+padSingle]==0)
                {
                    outPixel=0;
                    for (i=0; i<padBoth; i++)
                        for (j=0; j<padBoth; j++)
outPixel = outPixel || arrayErode[x+i][y+j];
Uint8 *pOut = (Uint8 *)surfaceOut->pixels + y * surfaceOut->pitch + x;

if (outPixel==1)
  *pOut = 255;
else
  *pOut = 0;
}
else {
  Uint8 *pOut = (Uint8 *)surfaceOut->pixels + y * surfaceOut->pitch + x;
  *pOut = 255;
}
}

/****************************THINNING CONDITION***************************/

//Thinning Condition - check thining condition (2 condition here)
//INPUT: array 'P' in 3x3 window
//RETURN: return 1 satisfy for thinning condition, pixel will be removed

unsigned int thinningConditionI(unsigned int P[9])
{
  //thinning condition I: to check nonzero neigbour of P[0]
  unsigned int B = 0;
  if ((B >= 2) && (B <= 6))
    return 1;
  else
    return 0;
}

//Thinning Condition - to check transition 0->1
//INPUT: array 'P' in 3x3 window
//RETURN: return 1 satisfy for thinning condition, pixel will be removed

unsigned int thinningConditionII(unsigned int P[9])
{
  unsigned int A = 0;
  if (!P[1] && (P[2]) A++;
  if (!P[2] && (P[3]) A++;
  if (!P[3] && (P[4]) A++;
  if (!P[4] && (P[5]) A++;
  if (!P[5] && (P[6]) A++;
  if (!P[6] && (P[7]) A++;
  if (!P[7] && (P[8]) A++;
  if (!P[8] && (P[1]) A++;

  if (A == 1)
    return 1;
  else
    return 0;
}
void thinning (SDL_Surface *surfaceIn, SDL_Surface *surfaceOut)
{
    unsigned int width = surfaceIn->w;
    unsigned int height= surfaceIn->h;
    unsigned int x,y;

    //padding
    unsigned int padW = width+2;
    unsigned int padH = height+2;

    //array to keep original pixel, and pixel to be deleted
    unsigned int arrayThinningOri [padW][padH];
    unsigned int arrayThinningDel [padW][padH];

    unsigned int count = 0;
    unsigned int P[9];//window 3x3

    unsigned int oriPixel, zeroOnePixel, xorPixel;
    int deleted;

    //initialise
    for (x=0; x<padW; x++)
        for (y=0; y<padH; y++)
            {
                arrayThinningOri[x][y]=0;
                arrayThinningDel[x][y]=0;
            }

    //in my image, 0=black, 255=white
    //in algorithm, 0=white, 1=black
    //1. change all pixel to 0 and 1 by divided by 255
    //2. invert all bit using xor
    for (x=1; x<padW-1; x++)
        for (y=1; y<padH-1; y++)
            {
                Uint8 *pIn=(Uint8 *)surfaceIn->pixels + (y-1) * surfaceIn->pitch + (x-1) ;
                oriPixel=*pIn;
                zeroOnePixel = oriPixel&1;
                xorPixel = zeroOnePixel ^ 1;
                arrayThinningOri[x][y]=xorPixel;
            }

    do
    {
        //1st iteration, loop to search M for every pixel
        for (x=1; x<padW-1; x++)
            for (y=1; y<padH-1; y++)
                {
                    //do if pixel is 1 (need to be deleted)
                    if (arrayThinningOri[x][y]==1)
                        {
                            P[0] = arrayThinningOri[x][y ];
                            P[1] = arrayThinningOri[x ][y-1];
                            P[2] = arrayThinningOri[x+1][y-1];
                        }
                }
    } while (count < padW*padH);
for (x=1; x<padW-1; x++)
for (y=1; y<padH-1; y++)
{
    deleted = arrayThinningOri[x][y] - arrayThinningDel[x][y];
    deleted = (deleted<0)? 0:deleted;
    arrayThinningOri[x][y] = deleted;
}

//2nd iteration, loop to search M for every pixel
if (!(count == 0))
{
    count =0;
    for (x=1; x<padW-1; x++)
for (y=1; y<padH-1; y++)
{
    if (arrayThinningOri[x][y]==1)
    {
        P[0] = arrayThinningOri[x][y];
P[1] = arrayThinningOri[x][y-1];
P[2] = arrayThinningOri[x+1][y-1];
P[3] = arrayThinningOri[x+1][y ];
P[4] = arrayThinningOri[x+1][y+1];
P[5] = arrayThinningOri[x ][y+1];
P[6] = arrayThinningOri[x-1][y+1];
P[7] = arrayThinningOri[x-1][y ];
P[8] = arrayThinningOri[x-1][y-1];

        arrayThinningDel[x][y]=0;
        if (thinningConditionI(P))
          if (thinningConditionII(P))
            if (!(P[1]&&P[3]&&P[5]))
              if(!(P[3]&&P[5]&&P[7]))
              {
                  arrayThinningDel[x][y]=1;
                  count++;
              }
        }
    }
    for (x=1; x<padW-1; x++)
for (y=1; y<padH-1; y++)
{
for (x=1; x<padW-1; x++)
    for (y=1; y<padH-1; y++)
    {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels + (y-1) * surfaceOut->pitch + (x-1);
      if(arrayThinningOri[x][y]==1)
        *pOut = 0;
      else
        *pOut = 255;
    }

//******************************MINUTIAE POINT EXTRACTION*******************************/
//INPUT: 'surfaceIn'
//OUTPUT: 'surfaceOut', array minutiaeRecord 'minutiae'
//RETURN: Total minutiae point 'count'
void minutiaeExtraction (SDL_Surface *surfaceIn, SDL_Surface *surfaceOut, const char *outFileInfo)
{
    struct minutiaeRecord minutiae[200];
    unsigned int width = surfaceIn->w;
    unsigned int height= surfaceIn->h;
    unsigned int i,x,y;
    //padding
    unsigned int padW = width+2;
    unsigned int padH = height+2;
    //array to keep original pixel, and pixel to be deleted
    unsigned int arrayMinutiaeOri [padW][padH];
    //Count for minutiae point
    unsigned int count = 0;
    //window 3x3.int because abs function
    int P[9];
    //result
    unsigned int oriPixel, zeroOnePixel;
    unsigned int Cn;
    //initialise
    for (x=0; x<padW; x++)
        for (y=0; y<padH; y++)
        {
            arrayMinutiaeOri[x][y]=0;
        }
// Change all pixels to 0 and 1 by divided by 255
for (x=1; x<padW-1; x++)
    for (y=1; y<padH-1; y++)
    {
        Uint8 *pIn=(Uint8 *)surfaceIn->pixels + (y-1) * surfaceIn->pitch + (x-1);
        oriPixel=*pIn;
        zeroOnePixel = oriPixel&1;
        arrayMinutiaeOri[x][y]=zeroOnePixel;
        Uint8 *pOut = (Uint8 *)surfaceOut->pixels + (y-1) * surfaceOut->pitch + (x-1);
        *pOut = 0;
    }

for (x=1; x<padW-1; x++)
    for (y=1; y<padH-1; y++)
    { // do if pixel is 0 (black pixel)
        if (!arrayMinutiaeOri[x][y])
            {
                P[0] = arrayMinutiaeOri[x][y-1];
                P[1] = arrayMinutiaeOri[x][y-1];
                P[2] = arrayMinutiaeOri[x+1][y-1];
                P[3] = arrayMinutiaeOri[x+1][y];
                P[4] = arrayMinutiaeOri[x+1][y+1];
                P[5] = arrayMinutiaeOri[x][y+1];
                P[6] = arrayMinutiaeOri[x-1][y+1];
                P[7] = arrayMinutiaeOri[x-1][y];
                P[8] = arrayMinutiaeOri[x-1][y-1];

                // cross number implementation
                Cn=0;
                for(i=0; i<8; i++)
                {
                    Cn += abs(P[i]-P[i+1]);
                }
                Cn += abs(P[8]-P[1]);

                // Store end ridge
                if (Cn == 2 || Cn == 3)
                    {
                        minutiae[count].type = endRidge;
                        minutiae[count].x = x-1;
                        minutiae[count].y = y-1;
                        count++;
                        Uint8 *pOut = (Uint8 *)surfaceOut->pixels + (y-1) * surfaceOut->pitch + (x-1);
                        *pOut = 255;
                    }

                // Store bifurcation ridge
                if (Cn == 6 || Cn == 7)
                    {
                        minutiae[count].type = biRidge;
                        minutiae[count].x = x-1;
                        minutiae[count].y = y-1;
                        count++;
                    }
            }
        }
 Uint8 *pOut = (Uint8 *)surfaceOut->pixels + (y-1) * surfaceOut->pitch + (x-1);

  *pOut = 255;

  if ( (x-3>=0) && (x-3<width) )
  {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels +
      (y-1) * surfaceOut->pitch + (x-3);
      *pOut = 255;
  }
  if ( (x-2>=0) && (x-2<width) )
  {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels +
      (y-1) * surfaceOut->pitch + (x-2);
      *pOut = 255;
  }
  if ( (y-3>=0) && (y-3<height) )
  {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels +
      (y-3) * surfaceOut->pitch + (x-1);
      *pOut = 255;
  }
  if ( (y-2>=0) && (y-2<height) )
  {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels +
      (y-2) * surfaceOut->pitch + (x-1);
      *pOut = 255;
  }
  if ( (y>=0) && (y<height) )
  {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels +
      * surfaceOut->pitch + (x-1);
      *pOut = 255;
  }
  if ( (y+1>=0) && (y+1<height) )
  {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels +
      (y+1) * surfaceOut->pitch + (x-1);
      *pOut = 255;
  }
  if ( (x>=0) && (x<width) )
  {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels +
      (y-1) * surfaceOut->pitch + (x);
      *pOut = 255;
  }
  if ( (x+1>=0) && (x+1<width) )
  {
      Uint8 *pOut = (Uint8 *)surfaceOut->pixels +
      (y-1) * surfaceOut->pitch + (x+1);
      *pOut = 255;
  }

FILE *outFileMinutiae;
outFileMinutiae = fopen (outFileInfo,"w+");

for (i=0; i<count; i++)
{
    //fprintf(outFileMinutiae, "%d\n",i);
    fprintf(outFileMinutiae, "%d\n",minutiae[i].type);
    fprintf(outFileMinutiae, "%d\n",minutiae[i].x);
    fprintf(outFileMinutiae, "%d\n",minutiae[i].y);
}
fprintf(outFileMinutiae, "%d",count);
fclose (outFileMinutiae);

int imageProcessing(const char *inFile,
    const char *outFileBMP1, const char *outFileBMP2, const char *outFileBMP3,
    const char *outFileBMP4, const char *outFileBMP5, const char *outFileBMP6,
    const char *outFileBMP7, const char *outFileBMP8, const char *outFileBMP9,
    const char *outFileInfo)
{
    SDL_Surface *image, *imageHist, *imageMedianFilt, *imageThres,
    *imMdnFilt2, *imageFilt, *imageClosing, *imageOpening, *imageThinning,
    *imageMinutiae, *imageMinutiae1, *imageEdge, *imageResize0, *imageResize;

    unsigned long long int clkCycle=0;
    clock_t t1, t2;
    //Load in bmp 'inFile'
    image = IMG_Load(inFile);

    //****************IMAGEPROCESSING**********************
    FILE *timeProfile;
    timeProfile = fopen("./timeProfile.csv","a+");
fclose (timeProfile);
imageMedianFilt= SDL_CreateRGBSurface(SDL_SWSURFACE,wOriIm,hOriIm,8,0xff,0xff,0xff,0);
printf (" \n Median filter (RGB)...
");
if (embedded==1)
{
    showHEX0(1);
    IOWR(COUNTER_BASE,0,0); //reset counter
    IOWR(COUNTER_BASE,0,1); //start counter
}
t1 = clock();

medianFilterRGB (image, imageMedianFilt, 3, 1);
if (embedded==1)
clkCycle = IORD(COUNTER_BASE,1); //read counter

t2 = clock();

printf ("\n\t\t\t (Time : %lu ms, %llu CC)n",(t2-t1)/1000,clkCycle);
timeProfile = fopen("./timeProfile.csv","a+");
fprintf(timeProfile, ",%lu,%llu",(t2-t1)/1000,clkCycle);
fclose (timeProfile);

SDL_SaveBMP(imageMedianFilt,outFileBMP1);

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

imageEdge = SDL_CreateRGBSurface(SDL_SWSURFACE,wOriIm,hOriIm,8,0xff,0xff,0xff,0);

printf ("\n\t\t\t - Canny Edge Detector...\n");
if (embedded==1)
{
    showHEX0(2);
    IOWR(COUNTER_BASE,0,0); //reset counter
    IOWR(COUNTER_BASE,0,1); //start counter
}
t1 = clock();

cannyEdgeDetector(imageMedianFilt, imageEdge, 2.5,4.0, 2.4); //try 3,35
if (embedded==1)
    clkCycle = IORD(COUNTER_BASE,1); //read counter
t2 = clock();

printf ("\n\t\t\t (Time : %lu ms, %llu CC)n",(t2-t1)/1000,clkCycle);
timeProfile = fopen("./timeProfile.csv","a+");
fprintf(timeProfile, ",%lu,%llu",(t2-t1)/1000,clkCycle);
fclose (timeProfile);

SDL_SaveBMP(imageEdge,outFileBMP2);

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

imageResize0 = SDL_CreateRGBSurface(SDL_SWSURFACE,wResizeIm,hResizeIm,8,0xff,0xff,0xff,0);
imageResize = SDL_CreateRGBSurface(SDL_SWSURFACE,wResizeIm,hResizeIm,8,0xff,0xff,0xff,0);

printf ("\n\t\t\t - Align, Resize...\n");
if (embedded==1)
{
    showHEX0(3);
    IOWR(COUNTER_BASE,0,0); //reset counter
    IOWR(COUNTER_BASE,0,1); //start counter
}
t1 = clock();

putBackAlignResize(imageEdge, imageMedianFilt, imageResize0,imageResize, 40, 35);
if (embedded==1)
    clkCycle = IORD(COUNTER_BASE,1); //read counter
t2 = clock();

timeProfile = fopen("./timeProfile.csv","a+");
fprintf(timeProfile, ",%lu,%llu",(t2-t1)/1000,clkCycle);
fclose (timeProfile);

SDL_SaveBMP(imageResize0,outFileBMP3);
//***************************************
//***************************************
imageThres = SDL_CreateRGBSurface(SDL_SWSURFACE, wResizeIm, hResizeIm, 8, 0xff, 0xff, 0xff, 0); printf ("\t\t-> Local Thresholding (Binarization)...\n"); if (embedded==1)
{
    showHEX0(4);
    IOWR(COUNTER_BASE,0,0); //reset counter
    IOWR(COUNTER_BASE,0,1); //start counter
}
t1 = clock();
localThresholding2 (imageResize0, imageResize0, imageThres, 13, 1);
if (embedded==1)
    clkCycle = IORD(COUNTER_BASE,1); //read counter
    t2 = clock();
    printf ("\t\t\t(Time : %lu ms, %llu CC)\n",(t2-t1)/1000,clkCycle);
    timeProfile = fopen("./timeProfile.csv","a+);
    fprintf(timeProfile, "%lu,%llu,(t2-t1)/1000,clkCycle);
    fclose (timeProfile);
    SDL_SaveBMP(imageThres,outFileBMP4);

//***************************************
//***************************************
imMdnFilt2 = SDL_CreateRGBSurface(SDL_SWSURFACE, wResizeIm, hResizeIm, 8, 0xff, 0xff, 0xff, 0);
printf ("\t\t-> Median Filter...\n");
if (embedded==1)
{
    showHEX0(5);
    IOWR(COUNTER_BASE,0,0); //reset counter
    IOWR(COUNTER_BASE,0,1); //start counter
}
t1 = clock();
medianFilter (imageThres, imageResize0, imMdnFilt2, 7, 3);
//medianFilter (imThres, imMdnFilt2, 5, 3);
if (embedded==1)
    clkCycle = IORD(COUNTER_BASE,1); //read counter
    t2 = clock();
    printf ("\t\t\t(Time : %lu ms, %llu CC)\n",(t2-t1)/1000,clkCycle);
    timeProfile = fopen("./timeProfile.csv","a+);
    fprintf(timeProfile, "%lu,%llu,(t2-t1)/1000,clkCycle);
    fclose (timeProfile);
    SDL_SaveBMP(imMdnFilt2,outFileBMP5);

//***************************************
//***************************************
imageOpening = SDL_CreateRGBSurface(SDL_SWSURFACE, wResizeIm, hResizeIm, 8, 0xff, 0xff, 0xff, 0);
printf ("\t\t-> Opening...\n");
if (embedded==1)
{
    showHEX0(6);
IOWR(COUNTER_BASE,0,0);  //reset counter
IOWR(COUNTER_BASE,0,1);  //start counter
}
t1 = clock();

//2009.06.30: swap closing and opening
//opening(imageClosing, imageOpening,11);
opening(imMdnFilt2, imageOpening,5);
//SDL_SaveBMP(imageOpening,outFileBMP6);
if (embedded==1)
kclCycle = IORD(COUNTER_BASE,1);  //read counter
t2 = clock();
printf ("t   t   t (Time : %lu ms, %llu CC)\n",(t2-t1)/1000,clkCycle);
timeProfile = fopen("./timeProfile.csv","a+");
fprintf(timeProfile, "%lu,%llu,",(t2-t1)/1000,clkCycle);
close (timeProfile);

SDL_SaveBMP(imageOpening,outFileBMP6);

*********************************************************************************/
timeProfile = fopen("./timeProfile.csv","a+");
close (timeProfile);
imageClosing = SDL_CreateRGBSurface(SDL_SWSURFACE,wResizeIm,hResizeIm,8,0xff,0xff,0xff,0);
printf ("t   t   t - > Closing...\n");
if (embedded==1)
{
    showHEX0(7);
    IOWR(COUNTER_BASE,0,0);  //reset counter
    IOWR(COUNTER_BASE,0,1);  //start counter
}
t1 = clock();

closing(imageOpening, imageClosing, 7);
if (embedded==1)
kclCycle = IORD(COUNTER_BASE,1);  //read counter
t2 = clock();
printf ("t   t   t (Time : %lu ms, %llu CC)\n",(t2-t1)/1000,clkCycle);
timeProfile = fopen("./timeProfile.csv","a+");
fprintf(timeProfile, "%lu,%llu,",(t2-t1)/1000,clkCycle);
close (timeProfile);

SDL_SaveBMP(imageClosing,outFileBMP7);
//http://www.songho.ca/misc/timer/timer.html

*********************************************************************************/
imageThinning = SDL_CreateRGBSurface(SDL_SWSURFACE,wResizeIm,hResizeIm,8,0xff,0xff,0xff,0);
printf ("t   t   t - > Thinning...\n");
if (embedded==1)
{
    showHEX0(8);
    IOWR(COUNTER_BASE,0,0);  //reset counter
    IOWR(COUNTER_BASE,0,1);  //start counter
}
t1 = clock();

//thinning (imageOpening, imageThinning);  //2009.06.30 delete
thinning (imageClosing, imageThinning);
//SDL_SaveBMP(imageThinning,outFileBMP7);
if (embedded==1)
    clkCycle = IORD(COUNTER_BASE,1); //read counter
t2 = clock();

printf ("\t\t(Time : %lu ms, %llu CC)\n",(t2-t1)/1000,clkCycle);
timeProfile = fopen("./timeProfile.csv","a+" gord);
fprintf(timeProfile, "%lu,%llu,\n,(t2-t1)/1000,clkCycle);
fclose (timeProfile);
SDL_SaveBMP(imageThinning,outFileBMP8);

//*****************************************************************************
imageMinutiae = SDL_CreateRGBSurface(SDL_SWSURFACE,wResizeIm,hResizeIm,8,0xff,0xff,0xff,0);
printf ("\tMinutiae Extraction...\n");
if (embedded==1)
{
    showHEX0('A');
    IOWR(COUNTER_BASE,0,0); //reset counter
    IOWR(COUNTER_BASE,0,1); //start counter
}
t1 = clock();

minutiaeExtraction (imageThinning, imageMinutiae, outFileInfo);
//SDL_SaveBMP(imageMinutiae,outFileBMP8);
if (embedded==1)
    clkCycle = IORD(COUNTER_BASE,1); //read counter
t2 = clock();
printf ("\t(Time : %lu ms, %llu CC)\n",(t2-t1)/1000,clkCycle);
timeProfile = fopen("./timeProfile.csv","a+" görd);
fprintf(timeProfile, "%lu,%llu,\n,(t2-t1)/1000,clkCycle);
fclose (timeProfile);
SDL_SaveBMP(imageMinutiae,outFileBMP9);
if (embedded==1)
    showHEX0(0);
SDL_FreeSurface (image);

return 0;
}

//********************************************************THRESHOLD AND EER********************************************************/
//Main function for Database MHD Calculation
//Calculate threshold, EER
//void db_mhd() void db_mhd(unsigned int alignment)
{
    //array to keep client and imposter score
    unsigned int valueClient[clientScore], valueImposter[impScore];

    //count for total client and imposter score
    unsigned int countClient=0, countImposter=0;

    //result directory and file name
    char sResultClient[80];
char sResultImposter[80];
char sResultThreshold[80];
char sResultEER[80];

//results
float meanClient, stdevClient;
float meanImposter, stdevImposter;
float threshold;
float tempClient, tempImposter;
float erf,EER;

//for looping
unsigned int i;

//call function to calculate client and imposter score
system("echo -n \e[4;37m""); //underline, white
printf("Calculating Client & Imposter Score:\n");
system("echo -n \e[0m");

calMatchAndRejectScore (alignment);

printf("...Done\n\n\n\n");

system("echo -n \e[4;37m""); //underline, white
printf("Calculating Threshold and EER:\n");
system("echo -n \e[0m");

//using global declaration
strcpy(sResultClient, outResultDir);
strcat(sResultClient, outScoreClient);
strcpy(sResultImposter, outResultDir);
strcat(sResultImposter, outScoreImposter);
strcpy(sResultThreshold, outResultDir);
strcat(sResultThreshold, outValueThreshold);
strcpy(sResultEER, outResultDir);
strcat(sResultEER, outValueEER);

FILE *outClient;
FILE *outImposter;
FILE *outThreshold;
FILE *outEER;

outClient  = fopen (sResultClient,"r");
outImposter = fopen (sResultImposter,"r");
outThreshold = fopen (sResultThreshold,"w+");
outEER = fopen (sResultEER,"w+");

meanClient=0;
//processing for client’s score
if (outClient == NULL) perror ("Error opening file rClient");
else
    while (!feof(outClient))
    {
        if (fscanf(outClient, "%u", &valueClient[countClient] )==1) //avoid to
          read eof twice
            [ 


meanClient+=valueClient[countClient];
countClient++;
}
}
fclose (outClient);
countClient=countClient-1;        //last line is total line
//calculate mean and stdDev
meanClient=meanClient-valueClient[countClient];
meanClient=meanClient/countClient;
stdevClient=0;
for (i=0; i<countClient; i++)
    stdevClient += (meanClient - valueClient[i])*(meanClient - valueClient[i]);
stdevClient = sqrt(stdevClient/(countClient-1));

meanImposter=0;
//processing for imposter's score
if (outImposter == NULL) perror ("Error opening file rImposter");
else
    while (!feof(outImposter))
    {
        if(fscanf(outImposter, "%u", &valueImposter[countImposter] )==1)
            //avoid to read eof twice
            {
                meanImposter+=valueImposter[countImposter];
countImposter++;
            }
    }
fclose (outImposter);
countImposter=countImposter-1;
//calculate mean and stdDev
meanImposter=meanImposter-valueImposter[countImposter];
meanImposter=meanImposter/countImposter;
stdevImposter=0;
for (i=0; i<countImposter; i++)
    stdevImposter += (meanImposter - valueImposter[i])*(meanImposter - valueImposter[i]);
stdevImposter = sqrt(stdevImposter/(countImposter-1));

//calculate thresholding
threshold = ( (meanClient*stdevImposer)+(meanImposter*stdevClient) ) /
            (stdevClient+stdevImposter);
//keep the value in text file
fprintf(outThreshold, "%f",threshold);
fclose (outThreshold);
tempClient = (float(threshold)-meanClient)/(stdevClient*1.4142);
tempImposter = (float(threshold)-meanImposter)/(stdevImposter*1.4142);

erf = 1.128379167*(tempImposter - pow(tempImposter,3)/3 + pow(tempImposter,5)/10 -
pow(tempImposter,7)/42 + pow(tempImposter,9)/216 - pow(tempImposter,11)/1320 +
pow(tempImposter,13)/9360 -pow(tempImposter,15)/75600);
EER = (0.5 * (1+erf)) * 100;
fprintf(outEER, "%f", EER);
fclose(outEER);

printf("countClient (%u)\t", countClient);
printf("meanClient (%f)\t", meanClient);
printf("stdevClient (%f)\t", stdevClient);

printf("countImposter (%u)\t", countImposter);
printf("meanImposter (%f)\t", meanImposter);
printf("stdevImposter (%f)\n\t", stdevImposter);

printf("threshold (%f)\t", threshold);
printf("tempClient (%f)\t", tempClient);
printf("tempImposter (%f)\n\t", tempImposter);

printf("EER (%f percent)\n", EER);
printf("\n\n");
}
//**************************THRESHOLD AND EER*************
*************//
//Main function for Database MHD Calculation
//Calculate threshold, EER
void db_mhd(unsigned int alignment)
{
    //array to keep client and imposter score
    unsigned int valueClient[clientScore], valueImposter[impScore];

    //count for total client and imposter score
    unsigned int countClient=0, countImposter=0;

    //result directory and file name
    char sResultClient[80];
    char sResultImposter[80];
    char sResultThreshold[80];
    char sResultEER[80];

    //results
    float meanClient, stdevClient;
    float meanImposter, stdevImposter;
    float threshold;
    float tempClient, tempImposter;
    float erf,EER;

    //for looping
    unsigned int i;

    //call function to calculate client and imposter score
    system("echo -n \"e[4;37m\"e[1;37m\"");  //underline, white
    printf("Calculating Client & Imposter Score:\n");
    system("echo -n \"e[0m\"");

    system("echo -n \"e[4;37m\"e[1;37m\"");  //underline, white
    printf("Calculating Threshold and EER:\n");
system("echo -n 'e[0m'");

// using global declaration
strcpy(sResultClient, outResultDir);
strcat(sResultClient, outScoreClient);
strcpy(sResultImposter, outResultDir);
strcat(sResultImposter, outScoreImposter);
strcpy(sResultThreshold, outResultDir);
strcat(sResultThreshold, outValueThreshold);
strcpy(sResultEER, outResultDir);
strcat(sResultEER, outValueEER);

FILE *outClient;
FILE *outImposter;
FILE *outThreshold;
FILE *outEER;

outClient = fopen (sResultClient,"r");
outImposter = fopen (sResultImposter,"r");
outThreshold = fopen (sResultThreshold,"w+");  
outEER = fopen (sResultEER,"w+");

meanClient=0;
// processing for client's score
if (outClient == NULL) perror ("Error opening file rClient");
else
    while (!feof(outClient))
    {
        if(fscanf(outClient, "%u", &valueClient[countClient] )==1)
            {
                meanClient+=valueClient[countClient];
                countClient++;
            }
    }
fclose (outClient);

countClient=countClient-1;   //last line is total line

// calculate mean and stdDev
meanClient=meanClient-valueClient[countClient];
meanClient=meanClient/countClient;

stdevClient=0;
for (i=0; i<countClient; i++)
    stdevClient += (meanClient - valueClient[i])*(meanClient - valueClient[i]);

stdevClient = sqrt(stdevClient/(countClient-1));

meanImposter=0;
// processing for imposter's score
if (outImposter == NULL) perror ("Error opening file rImposter");
else
    while (!feof(outImposter))
    {
        if(fscanf(outImposter, "%u", &valueImposter[countImposter] )==1)
            {
                meanImposter+=valueImposter[countImposter];
                countImposter++;
            }
    }

fclose (outImposter);
countImposter=countImposter-1;

//calculate mean and stdDev
meanImposter=meanImposter-valueImposter[countImposter];
meanImposter=meanImposter/countImposter;

stdevImposter=0;
for (i=0; i<countImposter; i++)
    stdevImposter += (meanImposter - valueImposter[i])*(meanImposter - valueImposter[i]);

stdevImposter = sqrt(stdevImposter/(countImposter-1));

//calculate thresholding
threshold = ( (meanClient*stdevImposter)+(meanImposter*stdevClient) ) /
(stdevClient+stdevImposter);

//keep the value in text file
fprintf(outThreshold, "%f", threshold);
close (outThreshold);

tempClient = (float(threshold)-meanClient)/(stdevClient*1.4142);
tempImposter = (float(threshold)-meanImposter)/(stdevImposter*1.4142);

erf = 1.128379167*(tempImposter - pow(tempImposter,3)/3 + pow(tempImposter,5)/10
-pow(tempImposter,7)/42 + pow(tempImposter,9)/216 -pow(tempImposter,11)/1320 +
pow(tempImposter,13)/9360 -pow(tempImposter,15)/75600);
EER = (0.5 *(1+erf))*100;
fprintf(outEER, "%lf", EER);
close (outEER);

printf("countClient (%u)\n",countClient);
printf("meanClient (%f)\n",meanClient);
printf("stdevClient (%f)\n",stdevClient);

printf("countImposter (%u)\n",countImposter);
printf("meanImposter (%f)\n",meanImposter);
printf("stdevImposter (%f)\n",stdevImposter);

printf("threshold (%f)\n",threshold);
printf("tempClient (%f)\n",tempClient);
printf("tempImposter (%f)\n",tempImposter);

printf("EER (%f percent)\n",EER);
printf("\n");}