Fingerprint Authentication System

T. G. Vikram Rao

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela-769 008, Odisha, India

Fingerprint Authentication System

Thesis submitted in

May 2014

to the department of

Computer Science and Engineering

of

National Institute of Technology Rourkela

in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

T. G. Vikram Rao

Roll No. 110CS0143

under the guidance of

Prof. Banshidhar Majhi

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela-769 008, Odisha, India

Department of Computer Science & Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India. www.nitrkl.ac.in

Dr. Banshidhar Majhi
Professor

May 07, 2014

Certificate

This is to certify that the work in the thesis entitled ***Fingerprint Authentication System*** by ***T. G. Vikram Rao***, bearing Roll No. **110CS0143**, is a record of an original work carried out by him under my guidance in partial fulfilment of the requirements for the award of the degree of *Bachelor of Technology* in *Computer Science and Engineering*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Prof. Banshidhar Majhi

Acknowledgments

I would like to express my earnest gratitude to my thesis guide, Prof. Banshidhar Majhi for believing in my ability to work on the challenging domain of biometrics.

My heartfelt thanks to several honourable people involved in this project for consistently showing me directions in carrying out the work. I am indebted to all the professors, batch mates and friends at National Institute of Technology Rourkela for their immense support.

T. G. Vikram Rao

Abstract

Fingerprint is one of the most widely used biometric modality for recognition due to its reliability, non-invasive characteristic, speed and performance. The patterns remain stable throughout the lifetime of an individual. Attributable to these advantages,the application of fingerprint biometric is increasingly encouraged by various commercial as well as government organizations. Fingerprint feature detection is to automatically and reliably extract minutiae from the input fingerprint images. However, the performance of a minutiae extraction algorithm relies heavily on the quality of the input fingerprint images. In order to ensure that the performance of an fingerprint authentication system to be robust, it is essential to preprocess fingerprint image. This thesis describes steps involved during fingerprint preprocessing, which improves the clarity of ridge and bifurcation structures of input fingerprint images. After preprocessing minutiae are extracted and stored in database. Further an online fingerprint authentication system is implemented in which elementary indexing strategy is used. Indexing fingerprint data is done to identify and retrieve a small subset of candidate data from the database of fingerprint data of individuals. Experimental work show that incorporating the online system, preprocessing algorithm, matching algorithm improves the overall response time.

Keywords: Biometrics, FingerPrint recognition, Indexing, kd-trees, authentication.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The term Biometrics refers to the field of development of statistical and mathematical methods applicable to data analysis problems existing in the biological sciences.Biometrics is the science of establishing the identity of an individual based on physiological and behavioural characteristics of the individual.The objective of Biometrics is to promote the use of statistical and mathematical theory towards the development of novel biometrical techniques and their application to new and ongoing subject matter challenges. Biometric authentication has evolved from the disadvantages of traditional means of authentication. It is more reliable and capable compared to traditional approaches. The problem with token based systems is that the possession could be lost, stolen, forgotten or misplaced. The drawbacks of knowledge based approaches is that it is tough for a person to remember difficult passwords/PINs; while keeping in mind secuirty against attacks.The combination of knowledge and token based system, e.g. automated teller machine (ATM) also cannot satisfy the security requirements. The primary advantage of biometrics over token based and knowledge based approaches is that, it cannot be misplaced, forgotten or stolen. Also it is very difficult to spoof biometric traits of an individual. A generic biometric system operates by taking an input from the user, preprocessing the signal to denoise it to find the region of interest, extracting features, and authenticating an individual based on the result of comparison [2]. Depending upon the application context a biometric system operates in the following modes: enrolment mode, verification mode, identification mode. In enrolment mode, the feature from a subject is extracted and stored in the database. In verification mode, a subject is authenticated by comparing, one on one, live query biometric template with the database template of the individual whom the subject claims himself to be. In identification mode, the system takes live query template from the subject and searches the entire database to find the best-match template to identify the subject and thereby making it a one-to-many process. Several biometric traits such as face, iris, fingerprint, voice, face-thermograph, signature are of key research area due to enormous need of security in automated systems.

Observing underlying modalities, two basic categories can be identified as: Physiological (or passive) and Behavioral (or active) biometrics [2]. Physiological biometrics are based on measurements or data derived from direct measurement of a human body part. Fingerprint, iris, retina, hand geometry, and face recognition are leading physiological biometrics. Behavioral characteristics, on the other hand, are based on an action taken by a person. Behavioral biometrics are based on measurements of data derived from an action, and thereby indirectly measure characteristics of the human body.A good biometric trait is characterised by use of features that are highly unique, stable, easy to capture,acceptable, collectable and prevents circumvention.

1.1 Fingerprint Biometrics

Fingerprint based recognition method because of its relatively outstanding features of universality, permanence, uniqueness, accuracy and low cost has made it most popular and a reliable technique and is currently the leading biometric technology [3]. Fingerprint identification is one of the most important biometric technologies which has drawn a substantial amount of attention recently [22, 24]. A fingerprint is the pattern of ridges and furrows on the surface of a fingertip. The uniqueness of a fingerprint is exclusively determined by the local ridge characteristics

These local ridge characteristics are not evenly distributed. Most of them depend heavily on the impression conditions and quality of fingerprints and are rarely observed in fingerprints. The two most prominent ridge characteristics, called minutiae, are ridge ending and ridge bifurcation. A ridge ending is defined as the point where a ridge ends abruptly. A ridge bifurcation is defined as the point where a ridge forks or diverges into branch ridges. A good quality fingerprint typically contains about 40-100 minutiae. However, in practice, due to variations in impression conditions, ridge configuration, skin conditions (aberrant formations of epidermal ridges of fingerprints, postnatal marks, occupational marks), acquisition devices, and non-cooperative attitude of subjects, etc. a significant percentage of acquired fingerprint images (approximately 10 percent according to our experience) is of poor quality. The ridge structures in poor-quality fingerprint images are not always well-defined and hence they can not be correctly detected. This leads to following problems like a significant number of spurious minutiae may be created, a large percent of genuine minutiae may be ignored, and larger errors in their localization (position and orientation) may be introduced. So because of these problems preprocessing of input image is done and then feature is extracted. Preprocessing is done in fingerprint biometrics to enhances image contrast, genuine minutiae and reduces errors.

1.2 Thesis Outline

This thesis is organized as follows, In abstract the problem statements and its approach to solve is mentioned in brief, and in Chapter 2 Literature review is done about fingerprint biometrics and several indexing approaches. In Chapter 3 Introduction to KD-Tree and operations involved, are explained in detail with algorithms. Chapter 4 explains about each and every step involved in fingerprint preprocessing from minutiae extraction to fingerprint matching, it also describes about the challenges encountered during minutiae extraction with algorithms and figures. All these modules constitute fingerprint authentication system. In Chapter 5 the details of Chapter 4 are implemented and experimental results are demonstrated. In Chapter 6 conclusion of project is described. All important references mentioned are appended after Chapter 6

Chapter 2

Literature Review

Current fingerprint recognition techniques can be broadly classified as Minutiae-based, Ridge feature-based, Correlation-based [1] and Gradient based [2]. Most fingerprint identification systems employ techniques based on minutiae points [3]. Although the minutiae pattern of each finger is quite unique, noise and distortion during the acquisition of the fingerprint and errors in the minutiae extraction process result in a number of missing and spurious minutiae [4].

The smooth flow pattern of ridges and valleys in a fingerprint can be also viewed as an oriented texture [3]. [7] describes a global texture descriptor called Finger Code that utilizes both global and local ridge descriptions for an oriented texture such as fingerprints. A variation to this method is used by [4] that use localized texture features of minutiae and another one by [8] that uses texture correlation matching.

In a fingerprint identification system as explained in [9], a person is identified only by his fingerprint. [9] provides solution to this problem by reducing the number of fingerprints that have to be matched. This is achieved by extracting features from the fingerprints and first matching the fingerprints that have the smallest feature distance to the query fingerprint. the registered directional field esti-mate, FingerCode and minutiae triplets. It is shown that combining these features results in better performance

Further, [2] proposed gradient based approach to capture textural information by dividing each minutiae neighbourhood locations into several local regions of which histograms of oriented gradients are then computed to characterize textural information around each minutiae location.[5] proposed that Texture feature of Energy of a fingerprint can be used for effecting fingerprint verification.

An efficient fingerprint indexing algorithm as proposed in [6] retrieves the top best matches from a huge database. It considers minutia features based on 9-dimensional index space comprised of transformation invariant information and a stable triangulation algorithm i.e 1-order Delaunay triangulation, both of which are insensitive to fingerprint distortion. It uses indexing technique based on kd-tree to reduce the search space to 15 percent of the large database.

Chapter 3

KD-Tree

3.1 Introduction

A k-d tree is a space-partitioning data structure for organizing points in a k-dimensional space. k-d trees are a useful data structure for several applications, such as searches involving a multidimensional search key. Figure 3-1 illustrates example of KD-Tree.

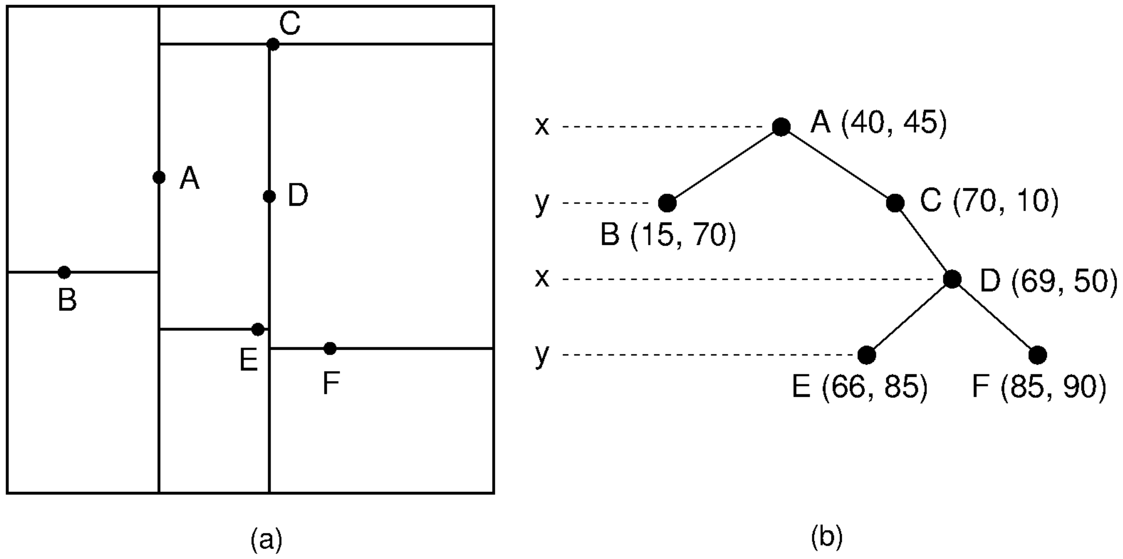


Figure 3-1: KDTree Example

The k-d tree is a binary tree in which every node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces. Points to the left of this hyperplane are represented by the left subtree of that node and points right of the hyperplane are represented by the right subtree. The hyperplane direction is chosen in the following way: every node in the tree is associated with one of the k-dimensions, with the hyperplane perpendicular to that dimension's axis. So, for example, if for a particular split the ”x” axis is chosen, all points in the subtree with a smaller ”x” value than the node will appear in the left subtree and all points with larger ”x” value will be in the right subtree. In such a case, the hyperplane would be set by the x-value of the point, and its normal would be the unit x-axis.

Operations on k-d trees are Insert, Delete, FindMin, Search and Nearest Neighbor described below.

3.2 Operations

3.2.1 Insert

Insertion is performed with the algorithm-1 as given in [6]. Insertions in an kD-tree are similar to any other search tree. The kD-tree is traversed to locate an appropriate leaf starting from the root depending on whether the point to be inserted is on the ”left” or ”right” side to accommodate the new entry [6]. The entry is inserted as either the left or right child of the leaf node again depending on which side of the node's splitting plane contains the new node. In case of duplicate entry, it is not inserted giving a duplicate entry message.

Algorithm 1 Insert

Input: T: Tree node, P: Point to be inserted, l: level, D: Tree Dimension **Output:** T: root node

```
1: procedure INSERT($T, P, l$)
2:     if $T$ is $null$ then
3:         Create new node including point P
4:     else if $T.data$ equals $P$ then
5:         Notify duplicate entry
6:     else if $T.data[l] > P[l]$ then
7:         $T.left \leftarrow$Insert($T.left, P, (l+1) \bmod D$)
8:     else
9:         $T.right \leftarrow$ Insert($T.right, P, (l+1) \bmod D$)
10:    end if
11:    return $T$
12: end procedure
```

3.2.2 Delete

To delete a point from an existing k-d tree is to form the set of all nodes and leaves from the children of the target node, and recreate that part of the tree. Deletion is performed with the algorithm-2 as given in [6].

Algorithm 2 Delete

Input: T: Tree node, P: Point to be Deleted, l: level, D: Tree Dimension **Output:** T: root node

```
1: procedure DELETE($T, P, l$)
2:     if $T$ is $null$ then
3:         Notify point not found.
4:         return $T$
5:     end if
6:     if $T.data$ equals $P$ then
7:         if $T.right \neq null$ then
8:             $T.data \leftarrow$FindMin( $T.right, l, (l+1)$ mod $D$)
9:             $T.right \leftarrow$Delete($T.right, T.data, (l+1)$ mod $D$)
10:        else if $T.left \neq null$ then
11:            $T.data \leftarrow$FindMin($T.left, l, (l+1)$ mod $D$)
12:            $T.left \leftarrow$Delete($T.left, T.data, (l+1)$ mod $D$)
13:        else
14:            $T \leftarrow null$
15:        end if
16:    else if $T.data[l] > P[l]$ then
17:        $T.left \leftarrow$Delete($T.left, P, (l+1)$ mod $D$)
18:    else
19:        $T.right \leftarrow$Delete($T.right, P, (l+1)$ mod $D$)
20:    end if
21:    return $T$
22: end procedure
```

3.2.3 FindMin

FindMin Routine helps to find a point with the smallest value in the dth dimension. FindMin is performed with the algorithm-3 as given in [6].

Algorithm 3 FindMin

Input: T: Tree node, l: level, D: Tree Dimension

Output: R: Tree node

```
1: procedure FindMin($T, l$)
2:     if $T$ is $null$ then
3:         Notify point not found.
4:         return $null$
5:     end if
6:     if $l$ equals $D$ then                ▷ T splits on the dimension were searching
7:         $l \leftarrow (l + 1) \bmod D$
8:         if $T.left == null$ then
9:             $R \leftarrow T$
10:            return $R$
11:        else
12:            return FindMin($T.left, l$)
13:        end if
14:    else
15:        $R \leftarrow$Minimum(FindMin($T.left, l$), FindMin($T.right, l$))
16:    end if
17:    return $R$
18: end procedure
```

3.2.4 Search

Search is performed with the algorithm-4 as given in [6].

Algorithm 4 Search

Input: T: Tree node, P: Point of interest

Output: T: Tree node

```
1: procedure SEARCH($T, P$)
2:     for all $l$ such that $T! = null$ do
3:         if $T.data$ equals $P$ then
4:             return $null$
5:         else if $T.data[l] > P[l]$ then
6:             $T \leftarrow T.left$
7:         else
8:             $T \leftarrow T.right$
9:         end if
10:        $l \leftarrow (l + 1) \bmod D$
11:    end for
12:    return $null$
13: end procedure
```

3.2.5 RangeSearch

RangeSearch is performed with the algorithm 5.

Algorithm 5 Range Search

Input: T: Tree node, P_1: Point Low , P_2: Point High, l: level, D: Tree Dimension

Output: R: List of Node(s)

```
1: procedure RangeSearch($T, P_1, P_2, l$)
2:     if $T$ is $null$ then
3:         return $R$
4:     end if
5:     if $P_1[l] <= T.data[l]$ then
6:         RangeSearch($T.left, P_1, P_2, (l + 1) \bmod D$)
7:     end if
8:     $j \leftarrow 0$
9:     while $P_1[l] <= T.data[l] <= P_2[l]$ do
10:        increment $j$
11:    end while
12:    if $j == l$ then
13:        Add $T$ to $R$
14:    end if
15:    if $P_2[l] > T.data[l]$ then
16:        RangeSearch($T.right, P_1, P_2, (l + 1) \bmod D$)
17:    end if
18: end procedure
```

3.2.6 Nearest Neighbor

The nearest neighbour search (NN) algorithm aims to find the point in the tree that is nearest to a given point. This search can be done efficiently by using the tree properties to quickly eliminate large portions of the search space.

Algorithm 6 Nearest Neighbor

Input: T: Tree node, P: Point to be inserted, l: level, BB: Bounding Box Rectangle, D: Tree Dimension

Output: R: Nearest Neighbor node

```
1: procedure NearestNeighbor($T, P, l, BB$)
2:     if $T$ is $null$ or $distance(P, BB) > BestDistance$ then
3:         $R \leftarrow T$
4:         return $R$                    ▷ if this bounding box is too far, then do nothing
5:     end if
6:     $dist \leftarrow distance(Q, T.data)$                    ▷ If this point is better than the best
7:     if $dist < BestDistance$ then
8:         $BestDistance \leftarrow dist$
9:     end if
10:    $l \leftarrow (l + 1) \bmod D$
11:    if $P[l] < T.data[l]$ then
12:        $R \leftarrow$ NearestNeighbor($T.left, P, l, BB.TrimLeft(l, T.data)$)
13:        $R \leftarrow$NearestNeighbor$T.right, P, l, BB.TrimRight(l, T.data)$)
14:    else
15:        $R \leftarrow$NearestNeighbor($T.right, P, l, BB.TrimRight(l, T.data)$)
16:        $R \leftarrow$NearestNeighbor($T.left, P, l, BB.TrimLeft(l, T.data)$)
17:    end if
18: end procedure
```

Chapter 4

Proposed Approach: Fingerprint Preprocessing

4.1 Minutiae Detection

This section first describes what fingerprint minutiae are,

4.1.1 Definition of Minutiae

Traditionally, two fingerprints have been compared using discrete features called minutiae. These features include points in a finger's friction skin where ridges end (called a ridge ending) or split (called a ridge bifurcation). Typically, there are on the order of 100 minutiae on a tenprint. In order to search and match fingerprints, the coordinate location and the orientation of the ridge at each minutia point are recorded. Figure 4-1 shows an example of the two types of minutiae. The minutiae are marked in the right image, and the tails on the markers point in the direction of the minutia's orientation.

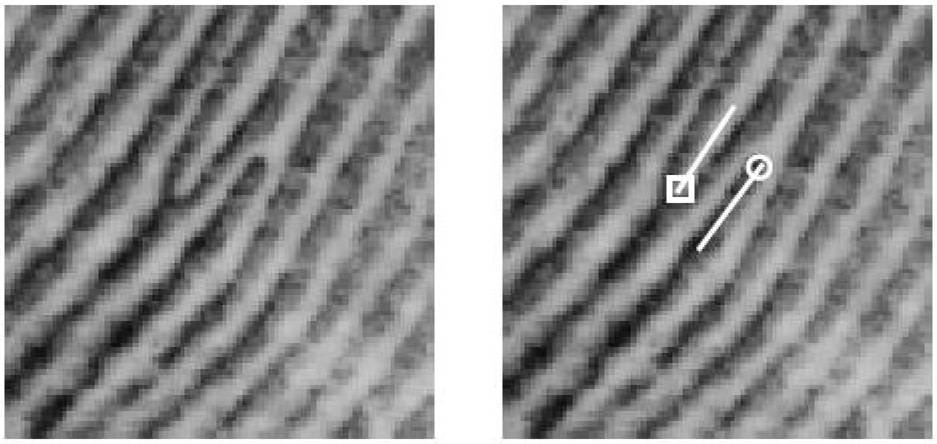


Figure 4-1: Minutiae

The location of each minutia is represented by a coordinate location within the fingerprint's image. Different AFIS systems represent this location differently. The ANSI/NIST standard specifies units of distance in terms of 0.01 mm from an origin in the bottom left corner of the image. For example, a 500 x 600 pixel image scanned at 19.69 pixels per millimeter (ppmm) has dimensions 25.39 x 30.47 mm which in standard units of 0.01 mm is Thus, the pixel coordinate (200, 192) will be represented in standard units at where the Y-coordinate is measured from the bottom of the image upward. Minutiae orientation is represented in degrees, with zero degrees pointing horizontal and to the right, and increasing degrees proceeding counter-clockwise. The orientation of a ridge ending is determined by measuring the angle between the horizontal axis and the line starting at the minutia point and running through the middle of the ridge. The orientation of a bifurcation is determined by measuring the angle between the horizontal axis and the line starting at the minutia point and running through the middle of the intervening valley between the bifurcating ridges.

Each minutia symbol is comprised of a circle or square, marking the location of the minutia point, and the line or tail proceeding from the circle or square is projected along either the ridge endings ridge, or the bifurcations valley. The angle of orientation as specified by the ANSI/NIST standard is marked as angle A in the illustration.

4.1.2 Latent Fingerprints

In addition to tenprints, there is a smaller population of fingerprints also important. These are fingerprints captured at crime scenes that can be used as evidence in solving criminal cases. Unlike tenprints, which have been captured in a relatively controlled environment for the expressed purpose of identification, crime scene fingerprints are by nature incidentally left behind. They are often invisible to the eye without some type of chemical processing or dusting. It is for this reason that they have been traditionally called latent fingerprints. As one would expect, the composition and quality of latent fingerprints are significantly different from tenprints. Typically, only a portion of the finger is present in the latent, the surface on which the latent was imprinted is unpredictable, and the clarity of friction skin details are often blurred or occluded. All this leads to fingerprints of significantly lesser quality than typical tenprints. While there are 100 minutiae on a tenprint, there may be only a dozen on a latent. Figure 4-2 shows a "good" quality latent on the left and its matching tenprint on the right.

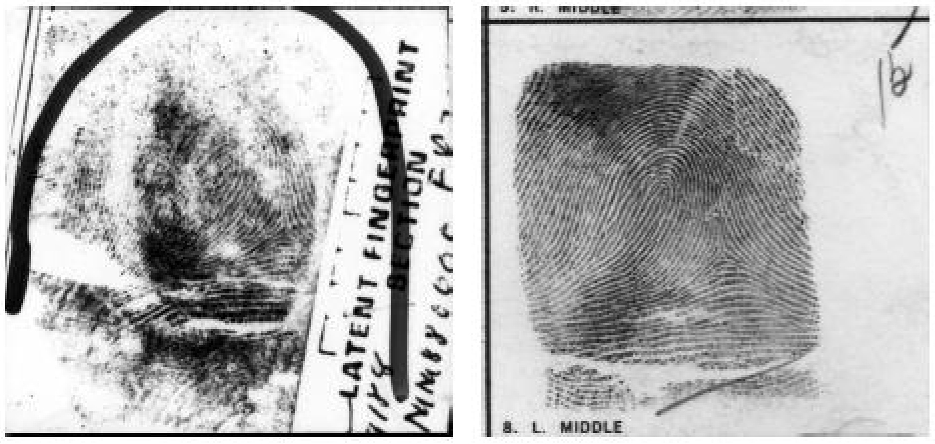


Figure 4-2: Latent Fingerprints

Due to the poor conditions of latent fingerprints, today's AFIS technology operates poorly when presented a latent fingerprint image. It is extremely difficult for the automated system to accurately classify latent fingerprints and reliably locate the minutiae in the image. Consequently, human fingerprint experts, called latent examiners, must analyze and manually mark up each latent fingerprint in preparation for matching. This is a tedious and labor intensive task. To support the processing of latent fingerprints, the FBI and NIST collaboratively developed a specialized workstation called the Universal Latent Workstation (ULW). This workstation has been designed to aid the latent examiner in preparing a latent fingerprint for search. In addition, the workstation provides for interoperability between different AFIS systems by functioning as a vendor-independent front-end interface. These two aspects of the ULW contribute significantly to the advancement of the state-of-the-art in latent fingerprint identification and law enforcement

4.1.3 Minutiae Detection Process

It should be noted that in minutae detection process algorithms and software parameters have been designed and set to optimally process images scanned at 500 pixels per inch and quantized to 256 levels of gray.

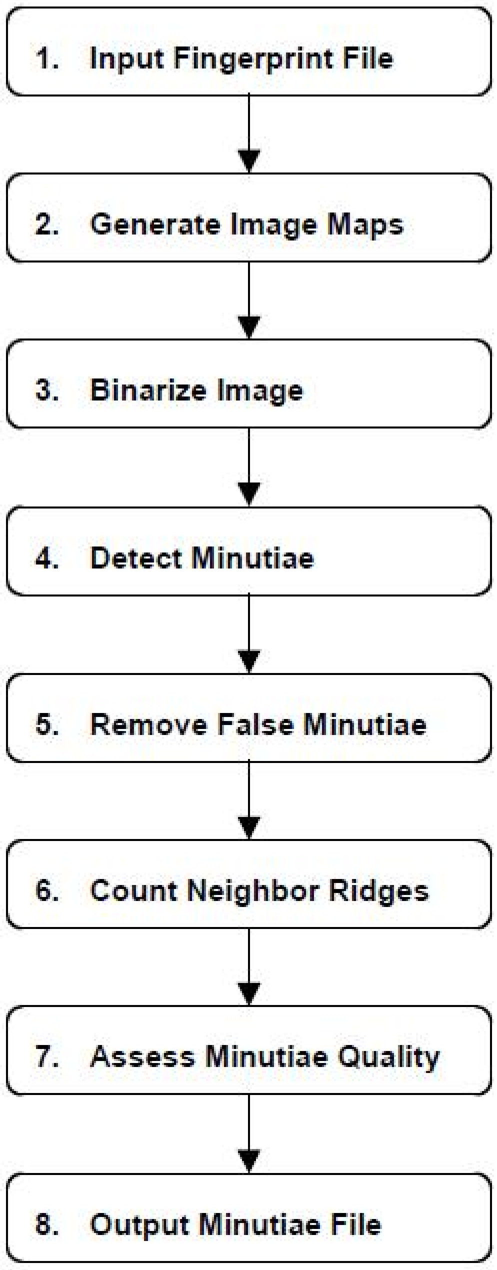


Figure 4-3: Minutiae Detection Process

the program for detecting minutiae in a fingerprint image. minutiae detection process. Figure 4-3 lists the functional steps executed. The software has been designed in a modular fashion so that each of the steps listed in Figure 4-3 is primarily executed by a single subroutine. This permits other alternative approaches to be implemented and substituted into the process, and the overall impact on performance can be evaluated. To support the many required operating parameters, a single global control structure is used to record sizes, tolerances, and thresholds.

4.1.3.1 Input Fingerprint Image File

Mindtct inputs a fingerprint image and automatically detects minutiae on the fingerprint. The algorithms and parameters have been developed and set for images scanned at 19.69 ppmm and quantized to 256 levels of gray. The application can read files in ANSI/NIST, WSQ, JPEGB, JPEGL, and IHEAD formats. In ANSI/NIST formatted files it searches the file structure for a grayscale fingerprint record. Once found, the fingerprint image in this record is processed. Currently, only the first grayscale fingerprint record in the ANSI/NIST file is processed, but the application could be changed to process all grayscale fingerprints in the ANSI/NIST file. Mindtct has an option that will allow it to enhance very low contrast images. If the option is selected, mindtct will evaluate the histogram of the input image. If the image is a very low contrast image, it is enhanced to improve the contrast otherwise it is not modified.

4.1.3.2 Generate Image Quality Maps

Because the image quality of a fingerprint may vary, especially in the case of latent fingerprints, it is critical to be able to analyze the image and determine areas that are degraded and likely to cause problems. Several characteristics can be measured that are designed to convey information regarding the quality of localized regions in the image. These include determining the directional flow of ridges in the image and detecting regions of low contrast, low ridge flow, and high curvature. These last three conditions represent unstable areas in the image where minutiae detection is unreliable, and together they can be used to represent levels of quality in the image.

4.1.3.3 Direction Map

One of the fundamental steps in this minutiae detection process is deriving a directional ridge flow map, or direction map. The purpose of this map is to represent areas of the image with sufficient ridge structure. Well-formed and clearly visible ridges are essential to reliably detecting points of ridge ending and bifurcation. In addition, the direction map records the general orientation of the ridges as they flow across the image.

To locally analyze the fingerprint, the image is divided into a grid of blocks. All the pixels within a block are assigned the same results.Therefore, in the case of the direction map, all the pixels in a block will be assigned the same ridge flow direction. Several considerations must be made when using a block-based approach. First, it must be determined how much local information is required to reliably derive the desired characteristic. This area is referred to as the window. The characteristic measured within the window is then assigned to each pixel in the block. It is typically desirable to share data used to compute the results assigned to neighboring blocks. This way some of the image that contributed to one blocks results is included in the neighboring blocks results as well. This helps minimize the discontinuity in block values as you cross the boundary from one block to its neighbor. This smoothing can be implemented using a system where a block is smaller than its surrounding window, and windows overlap from one block to the next. This is illustrated in Figure 4-4.

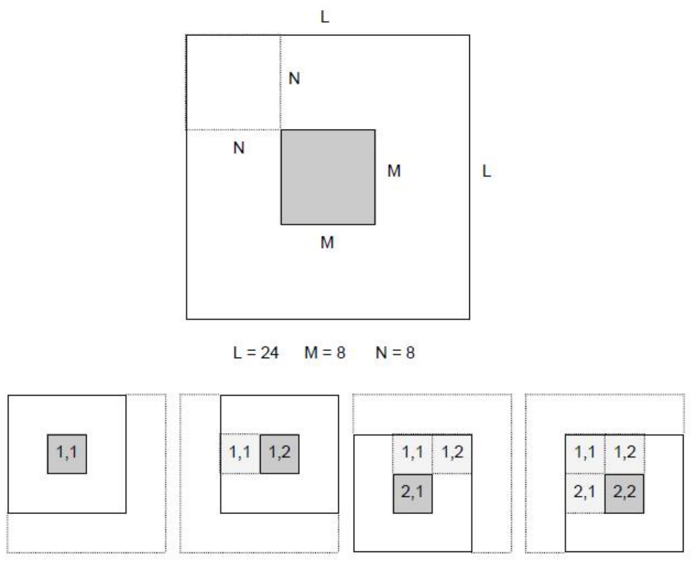


Figure 4-4: Direction Map

The large frame at the top of the figure depicts a window (in white) surrounding a smaller block (in gray). Assuming that neighboring blocks are adjacent and non-overlapping, this scenario is defined by three parameters: the window size L, the block size M and the offset of the block from the windows origin N. In the global control structure, lfsparmsV2, these parameters are defined as MAPWINDOWSIZEV2=24, MAPBLOCKSIZEV2=8, and APWINDOWOFFSETV2=8 respectively. As a result, the image is divided up into a grid of 8x8 pixel blocks with each block being assigned a result from a larger surrounding 24x24 pixel window, and the area for windows of neighboring blocks overlap by up to 2/3. The bottom row of frames in the Figure 4-4 illustrates how this works in practice. Designating the address of a block by its (row, column) indices, the left frame shows the first block (1,1) being computed. The next frame advances to the next adjacent block to the right, block (1,2). Correspondingly, its window is shifted 8 pixels, and the new block receives its results. Note that there are two copies of the image being used. Each window operates on the original image data, while block results are written to a separate output image. The third frame in the illustration depicts the window configuration for block (2,1), and the fourth frame shows its right neighborbeing computed.

One additional consideration must be made when using blocks. It must be determined how to handle the edges of the image. The dimensions of the image will likely not be an even multiple of blocks, and the windows surrounding blocks along the perimeter of the image may extend off the image. In this software, the image is padded by a margin of medium gray pixels (set to intensity 128). This margin is sufficiently large to contain the perimeter windows in the image. The processing of partial blocks is also accounted for at the right and bottom of the image. Given the above approach for computing block results with an overlapping window, the technique used for determining ridge flow direction in the image can be described. For each block in the image, the surrounding window is rotated incrementally and a Discrete Fourier Transform (DFT) analysis is conducted at each orientation. The top left box in the figure depicts a window with its rows rotated 90deg. counterclockwise so that they are aligned vertically. This is considered orientation 0 in the software.

The parameter NUMDIRECTIONS in the global control structure, lfsparsV2, specifies the number of orientations to be analyzed in a semicircle. This parameter is set to 16, creating an increment in angle of 11.25deg. between each orientation. These orientations are depicted on the circle in the figure. The bottom row in the figure illustrates the incremental rotation of the windows rows at each defined orientation.

When determining the direction of ridge flow for a block, each of its window orientations is analyzed. Within an orientation, the pixels along each rotated row of the window are summed together, forming a vector of 24 pixel row sums. The 16 orientations produce 16 vectors of row sums. Each vector of row sums is convolved with 4 waveforms of increasing frequency. The top waveform in the figure has a single period extending across the length of the entire vector. The second waveforms frequency is doubled from the first; the third is doubled from the second, and so forth. Discrete values for the sine and cosine functions at the 4 different frequencies are computed for each unit along the vector. The row sums in a vector are then multiplied to their corresponding discrete sine values, and the results are accumulated and squared. The same computation is done between the row sums in the vector and their corresponding discrete cosine values. The squared sine component is then added to the squared cosine component, producing a resonance coefficient that represents how well the vector fits the specific waveform frequency.

4.1.3.4 High Curve Map

Another part of fingerprint image that is problematic when it comes to detecting minutiae reliably is in areas of high curvature. This is especially true of the core and delta regions of a fingerprint. The high curve map marks blocks that are in high-curvature areas of the fingerprint. Two different measures are used. The first, called vorticity, measures the cumulative change in ridge flow direction around all the neighbors of a block. The second called, curvature, measures the largest change in direction between a blocks ridge flow and the ridge flow of each of its neighbors. The details are in the source code. In the event that minutiae are detected in these blocks, their assigned quality is reduced because they have been detected within a less reliable part of the image. The white cross marks in the fingerprint image in Figure 4-5 label blocks with high-curvature ridges.

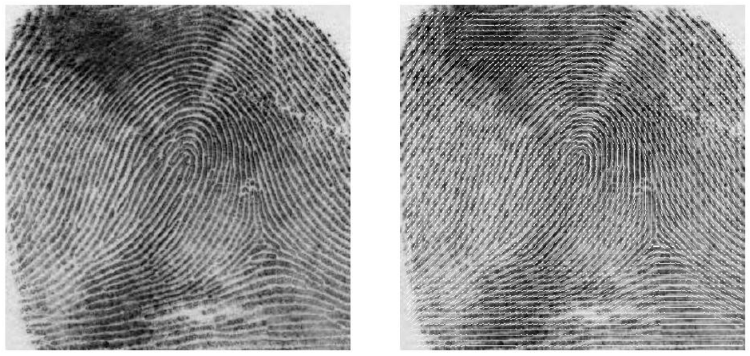


Figure 4-5: High Curve Map

4.1.3.5 Binarize Image

The minutiae detection algorithm in this system is designed to operate on a bi-level (or binary) image where black pixels represent ridges and white pixels represent valleys in a finger's friction skin. To create this binary image, every pixel in the grayscale input image must be analyzed to determine if it should be assigned a black or white pixel. This process is referred to as image binarization. A pixel is assigned a binary value based on the ridge flow direction associated with the block the pixel is within. If there was no detectable ridge flow for the current pixel's block, then the pixel is set to white. If there is detected ridge flow, then the pixel intensities surrounding the current pixel are analyzed within a rotated grid as illustrated in Figure 4-6.

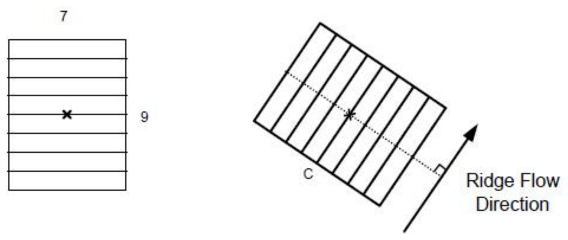


Figure 4-6: Rotation grid

Rotated grid used to binarize the fingerprint image. This grid is defined in the global control structure, lfsparmsV2, with column width (DIRBINGRIDW) set to 7 pixels and row height (DIRBINGRIDH) set to 9 pixels. With the pixel of interest in the center, the grid is rotated so that its rows are parallel to the local ridge flow direction. Grayscale pixel intensities are accumulated along each rotated row in the grid, forming a vector of row sums. The binary value to be assigned to the center pixel is determined by multiplying the center row sum by the number of rows in the grid and comparing this value to the accumulated grayscale intensities within the entire grid. If the multiplied center row sum is less than the grid's total intensity, then the center pixel is set to black; otherwise, it is set to white.

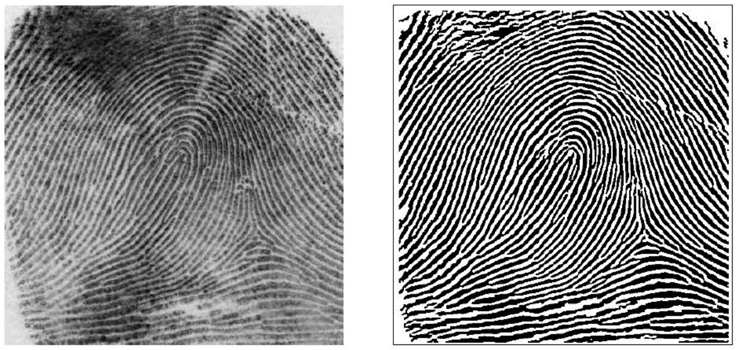


Figure 4-7: Binarize Image

The results of binarization are shown in the Figure 4-7. The original grayscale image is on the left, and its binarization results are on the right. It should be noted that the binarization step is critical to the successful detection of minutiae in this approach. The binarization results need to be robust in terms of effectively dealing with varying degrees of image quality and reliable in terms of rendering ridge and valley structures accurately. These are challenging, and at times conflicting goals. It is desirable to preserve as much image information and ridge/valley structure as possible so that minutiae are not missed, and yet it is undesirable to accentuate degraded areas in the image to the point of introducing false minutiae. Significant effort has been invested to promote both robust and reliable binary images, and yet the current system tends to produce a considerable number of false minutiae. This is particularly troublesome when processing latent fingerprint images.

4.1.3.6 Detect Minutiae

This step methodically scans the binary image of a fingerprint, identifying localized pixel patterns that indicate the ending or splitting of a ridge. The patterns searched for are very compact as illustrated in Figure 4-8. The left-most pattern contains six binary pixels in a 2x3 configuration. This pattern may represent the end of a black ridge protruding into the pattern from the right. The same is true for the next 2x4 pattern. The only difference between this pattern and the first one is that the middle pixel pair is repeated. In fact, this is true for all the patterns depicted. This ”family” of ridge ending patterns can be represented by the right-most pattern, where the middle pair of pixels (signified by *) may repeat one or more times.

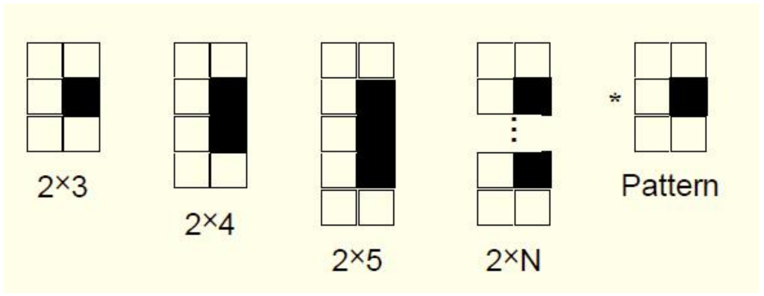


Figure 4-8: Localized pixel patterns

Candidate ridge endings are detected in the binary image by scanning consecutive pairs of pixels in the image looking for sequences that match this pattern. Pattern scanning is conducted both vertically and horizontally. The pattern as illustrated is configured for vertical scanning as the pixel pairs are stacked on top of each other. To conduct the horizontal scan, the pixel pairs are unstacked, rotated 90deg clockwise, and placed back in sequence left to right. Using the representation above, a series of minutiae patterns are used to detect candidate minutia points in the binary fingerprint image. These patterns are illustrated in Figure 4-9. There are two patterns representing candidate ridge endings, the rest represent various ridge bifurcations. A secondary attribute of appearing/disappearing is assigned to each pattern. This designates the direction from which a ridge or valley is protruding into the pattern. All pixel pair sequences matching these patterns, as the image is scanned both vertically and horizontally, form a list of candidate minutia points.

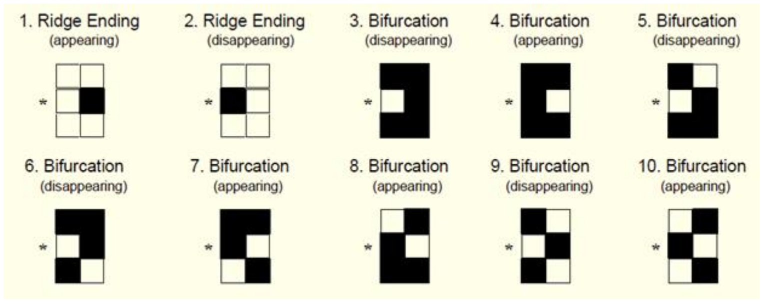

1. Ridge Ending (appearing)
2. Ridge Ending (disappearing)
3. Bifurcation (disappearing)
4. Bifurcation (appearing)
5. Bifurcation (disappearing)
6. Bifurcation (disappearing)
7. Bifurcation (appearing)
8. Bifurcation (appearing)
9. Bifurcation (disappearing)
10. Bifurcation (appearing)

Figure 4-9: Ridge bifurcation patterns

4.1.3.7 Remove False Minutiae

Using the patterns in Figure 4-9, candidate minutiae points are detected with as few as six pixels. This facilitates a particularly greedy detection scheme that minimizes the chance of missing true minutiae; however, many false minutiae are included in the candidate list. Because of this, much effort is spent on removing the false minutiae. These steps include removing islands, lakes, holes, minutiae in regions of poor image quality, side minutiae, hooks, overlaps, minutiae that are too wide, and minutiae that are too narrow (pores). A short description of each of these steps is provided in the order in which they are executed.

4.1.3.7.1 Remove Islands and Lakes

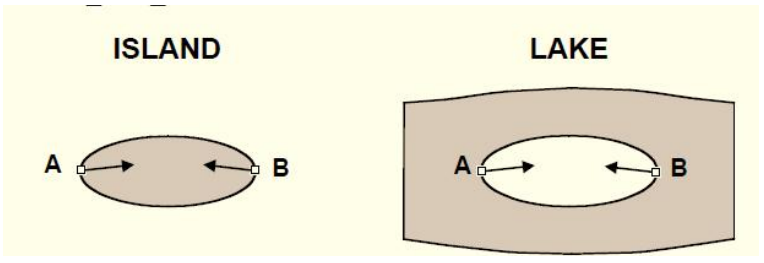


Figure 4-10: Remove Islands and Lakes

Algorithm 7 Remove Islands and Lakes

```
1: procedure REMOVE ISLANDS AND LAKES ($A, B$)
2:     if $distance(A, B) <= 18pixels$ then
3:         if $directionAngle(A, B) >= 123.75deg$ then
4:             if $onLoop(A)$ AND $onLoop(B)$ then
5:                 if $loopLength <= 80pixels$ then
6:                     remove($A, B$)
7:                 end if
8:             end if
9:         end if
10:    end if
11:    fillLoop()
12: end procedure
```

In this step, ridge ending fragments and spurious ink marks (islands) along with interior voids in ridges (lakes) are identified and removed. These features are somewhat larger than the size of pores in the friction skin and they are often elliptical in shape; therefore, they typically will have a pair of candidate minutia points detected at opposite ends. An illustration of these types of features is shown in Figure 4-10. Included at the bottom of the figure are the criteria used to detect islands and lakes. A pair of minutia must be within 16 pixels (MAXRMTESTDISTV2) of each other. If so, then the directions of the two minutiae must be nearly opposite (123.75deg) each other. Next, both minutiae must lie on the edge of the same loop, and the perimeter of the loop must be 60 pixels (MAXHALFLOOPV2). If all these criteria are true, then the pair of candidate minutiae are removed for the list and the binary image is altered so that the island

4.1.3.7.2 Remove Holes

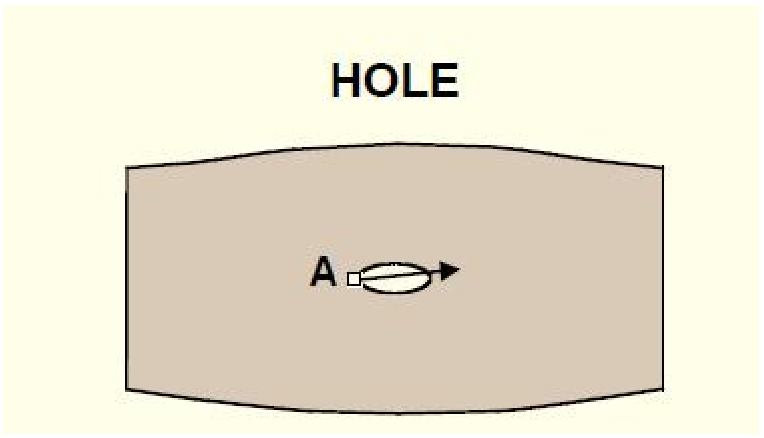


Figure 4-11: Remove Holes

Algorithm 8 Remove Holes

```
1: procedure REMOVE HOLES $(A, B)$
2:     if $onLoop(A)$ AND $onLoop(B)$ then
3:         if $loopLength <= 15pixels$ then
4:             remove$(A)$
5:         end if
6:     end if
7: end procedure
```

Here a hole is defined similarly to an island or lake, only smaller, and the loop need only have one minutia point on it. The criteria for removing a hole are illustrated in Figure 4-11. If a candidate minutia point lies on the edge of a loop with perimeter length 15 pixels(SMALLLOOPLEN), then the point is removed from the candidate list.

4.1.3.7.3 Remove Pointing to Invalid Block

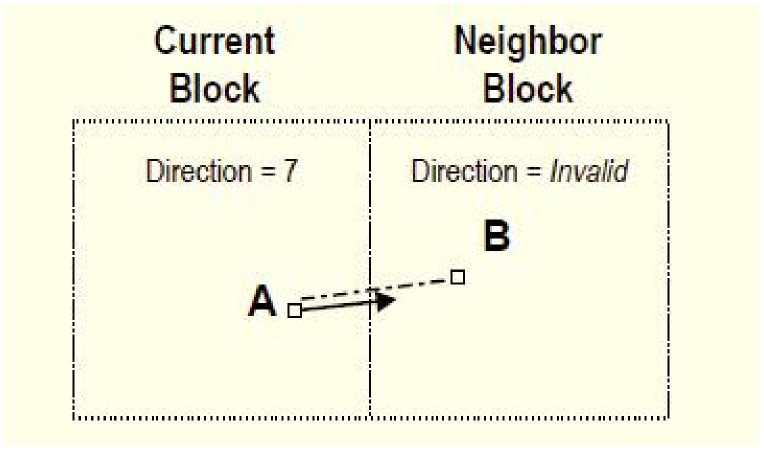


Figure 4-12: Remove Pointing to Invalid Block

Algorithm 9 Remove Pointing to Invalid Block

```
1: procedure REMOVE POINTING TO INVALID BLOCK $(A, B)$
2:     $B = translate(A, 4pixels, direction(A))$
3:     $D = direction(block(B))$
4:     if $D$ is invalid then
5:         remove($A$)
6:     end if
7: end procedure
```

This step and the next identify and remove candidate minutiae that are located near blocks that contain no detectable ridge flow. These blocks are referred to as containing invalid ridge flow direction and represent low-quality areas in the fingerprint image.

This step is illustrated in Figure 4-12. A minutia point is translated 4 pixels(TRANSDIRPIXV2) in the direction the minutia is pointing. If the translated point lies within a block with invalid ridge flow direction, then the original minutia point is remove from the list.

4.1.3.7.4 Remove Near Invalid Blocks

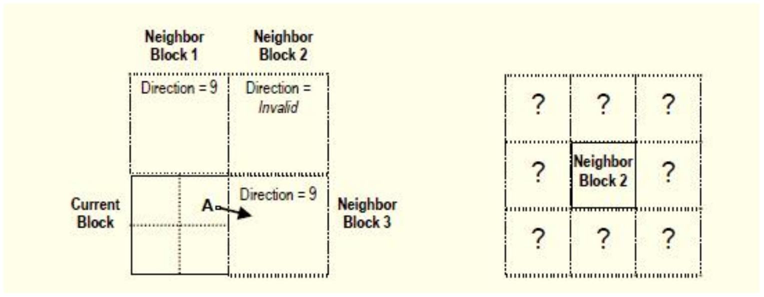


Figure 4-13: Remove Near Invalid Blocks

Algorithm 10 Remove Near Invalid Blocks

```
1: procedure REMOVE NEAR INVALID BLOCKS ($A$)
2:     $Nbrs = blockNeighbors(A)$
3:     $InvNbrs = invalidDirections(Nbrs)$
4:     while $Ni$ in $InvNbrs$ do
5:         $NiNbs = neighbors(Ni)$
6:         $Ci = countValidDirections(NiNbrs)$
7:     end while
8:     if $Ci < 7$ then
9:         remove($A$)
10:    end if
11: end procedure
```

Here, the proximity of a candidate minutia to a number of surrounding blocks with invalid ridge flow direction is evaluated. Given a minutia point, the blocks sufficiently close to the minutia (details left to the source code), and immediately neighboring the block in which the minutia resides, are tested in turn. If one of these neighboring blocks has invalid ridge flow direction, then its surrounding 8 neighbors are tested. The number of surrounding blocks with valid ridge flow direction are counted, and if the number of valid blocks is less than 7 (RMVALIDNBRMIN), then the original minutia point is removed from the candidate list. Figure 4-13 illustrated this step.

4.1.3.7.5 Remove or Adjust Side Minutiae

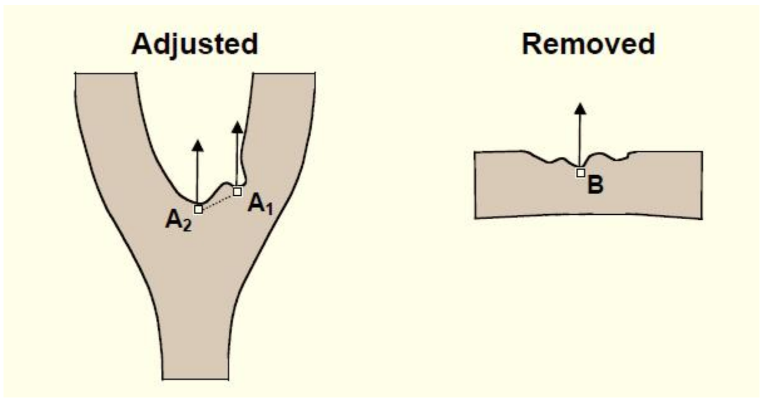


Figure 4-14: Remove or Adjust Side Minutiae

Algorithm 11 Remove or Adjust Side Minutiae

```
1: procedure REMOVE OR ADJUST SIDE MINUTIAE ($A$)
2:     $Pts = traceContours(A, 7pixels)$
3:     $RPts = rotatePointsVertical(Pts, direction(A))$
4:     $(MinYs, MaxYs) = minMaxYs(RPts)$
5:     if $MinYs == 1$ then
6:         $Adjust(A, Pts[MinY1])$
7:     else if $(MinYs, MaxYs) == (MinY1, MaxY1)$ then
8:         $MinY = pointAtMinY(RPts, MinYs)$
9:         $Adjust(A, Pts[MinY])$
10:    else
11:        revove(A)
12:    end if
13: end procedure
```

This step accomplishes two purposes. The first is to fine-tune the position of a minutia point so that it is more symmetrically placed on a ridge or valley ending. In the process, it may be determined that there is no clear symmetrical shape to the contour on which the candidate minutia lies. This is often the case with points detected along the side of a ridge or valley instead of the ridge or valley's ending. In this case, the misplaced minutia point is removed. In Figure 4-14, the illustration on the left depicts the adjustment of a minutia point from point A1 to A2. The illustration on the right depicts the removal of a side point, B. To accomplish this, starting at the candidate minutia point, the edge of either the ridge or valley is traced to the right and to the left 7 pixels (SIDEHALFCONTOUR) , producing a list of 15 contour points. The coordinates of these contour points are rotated so that the direction of the candidate minutia is pointing vertical. The rotated coordinates are then analyzed to determine the number and sequence of relative maxima and minima in the rotated y-coordinates. If there is only one y-coordinate minima, then the point of the minimum is assumed to lie at the bottom of a bowl-shaped rotated contour, and the candidate minutia is moved to correspond to this position in the original image. If there are more than one y-coordinate minima, then a specific sequence of minima-maxima-minima must exist, in which case the candidate minutia is moved to the point. in the original image corresponding to the lowest y-coordinate minima. Again, this is assumed to be the bottom of a relatively bowl-shaped rotated contour. If there is more than one ycoordinate minima and there is not an exact minima-maxima-minima sequence along the rotated contour, then the minutia point is determined to lie along the side of a ridge or valley, and it isremoved from the candidate list

4.1.3.7.6 Remove Hooks

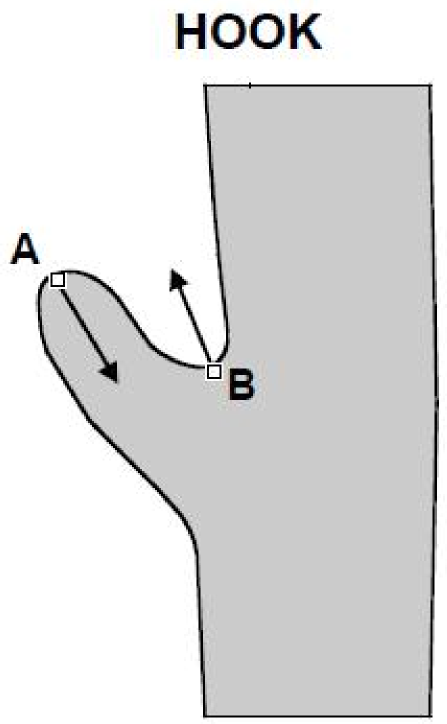


Figure 4-15: Remove Hooks

Algorithm 12 Remove Hooks

```
1: procedure REMOVE HOOKS $(A, B)$
2:     if $distance(A, B) <= 16 pixels$ then
3:         if $directionAngle(A, B) >= 123.75deg$ then
4:             if $type(A)! = type(B)$ then
5:                 Pts=traceCountours(A, 30 pixels)
6:                 if inPoints(Pts,B) then
7:                     remove$(A, B)$
8:                 end if
9:             end if
10:        end if
11:    end if
12: end procedure
```

A hook is a spike or spur that protrudes off the side of a ridge or valley. An example is illustrated in Figure 4-15. This feature typically has two minutiae of opposite type, one on a small piece of ridge and the other in a small valley, that are relatively close to each other. The two points must be within 16 pixels (MAXRMTESTDISTV2) of each other, their directions must be nearly opposite (123.75deg), they must be of opposite type, and they must lie on the same ridge/valley edge within 30 contour pixels (MAXHOOKLENV2) from each other. If all these are true, then the two minutia points are removed from the candidate list.

4.1.3.7.7 Remove Overlaps

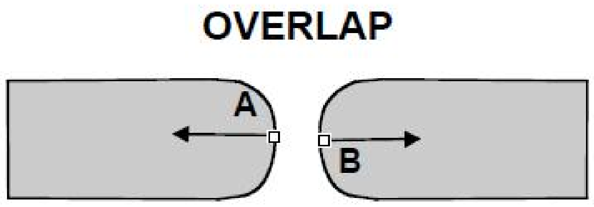


Figure 4-16: Remove Overlaps

Algorithm 13 Remove Overlaps

```
1: procedure REMOVE OVERLAPS $(A, B)$
2:     if $distance(A, B) <= 8pixels$ then
3:         if $directionAngle(A, B) >= 123.75deg$ then
4:             if $type(A) == type(B)$ then
5:                 J=joinDirections(A, B)
6:                 if $directionAngle(180deg - A, J) <= 90deg$ then
7:                     remove$(A, B)$
8:                 else if $distance(A, B) <= 18pixels$ AND $freePath(A, B)$ then
9:                     remove$(A, B)$
10:                end if
11:            end if
12:        end if
13:    end if
14: end procedure
```

In this step, an overlap is a discontinuity in a ridge or valley. These artifacts are typically introduced by the fingerprint impression process. A break in a ridge causes 2 false ridge endings to be detected, while a break in a valley causes 2 false bifurcations. The criteria for detecting an overlap are illustrated in Figure 4-16. Two minutia points must be within 8 pixels (MAXOVERLAPDIST) of each other, and their directions must be nearly opposite (123.75deg). If so, then the direction of the line joining the two minutia points is calculated. If the difference between the direction of first minutia and the joining line is (90deg), then the two minutiae are removed from the cadidate list. Otherwise, if the minutiae are within 6 pixels (MAXOVERLAPJOINDIST) of each other, and there are no pixel value transitions along the joining line, then the points are removed from the candidate list.

4.1.3.7.8 Remove Too Wide Minutiae

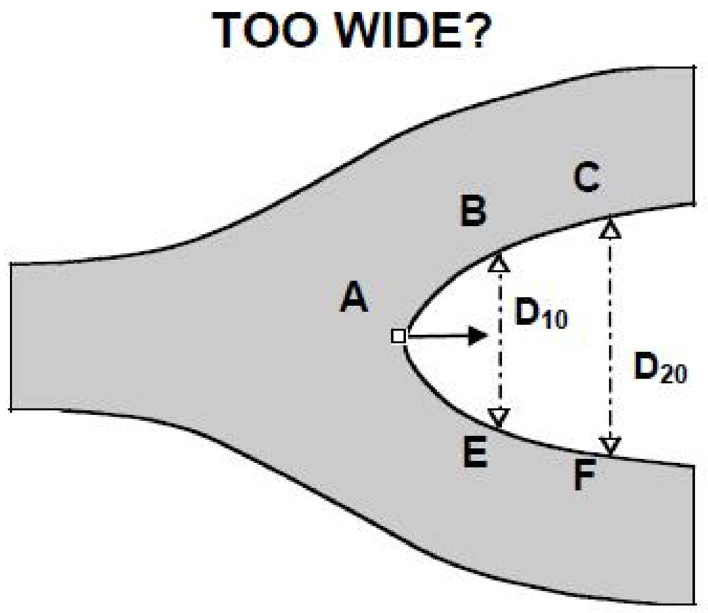


Figure 4-17: Remove Too Wide Minutiae

Algorithm 14 Removal of too wide minutiae

```
1: procedure Removal of too wide minutiae $(A, B)$
2:     Pts1 = traceContour(A, 20 pixels)
3:     Pts2 = traceContour(A, -20 pixels)
4:     B= Pts1(10); C = Pts1(20)
5:     E = Pts2(10); F = Pts2(20)
6:     D10 = distance(B, E)
7:     D20 = distance(C, F)
8:     if $D20/D10 > 2.0$ then
9:         remove($A$)
10:    end if
11: end procedure
```

The next two steps identify false minutiae that lie on malformed ridge and valley structures. A generalized ridge ending is comprised of a Y-shaped valley enveloping a black rod. The inverse is true for a generalized bifurcation. Simple tests are applied to evaluate the quality of this Yshape. This step evaluates whether the structure enveloping a ridge or valley ending is relatively Yshaped and not too wide. Figure 4-17 illustrates the criteria applied. The edge of the ridge or valley is traced to the left and to the right 20 pixels (MALFORMATIONSTEPS2) producing 2 lists of contour points. On each contour,coordinates at pixel index 10 (B,E) and at pixel index 20 (C,F) are stored. The distance between pixels at index 10 (MALFORMATION-STEPS1) is computed as is the distance between pixels at index 20. The ratio of these two distances is then calculated (D20/D10), and if the ratio is larger than 2.0 (MINMALFORMATIONRATIO), then the minutia point is removed from the candidate list. It should be noted that based on these criteria the bifurcation in the illustration would not be removed.

4.1.3.7.9 Remove Too Narrow Minutiae

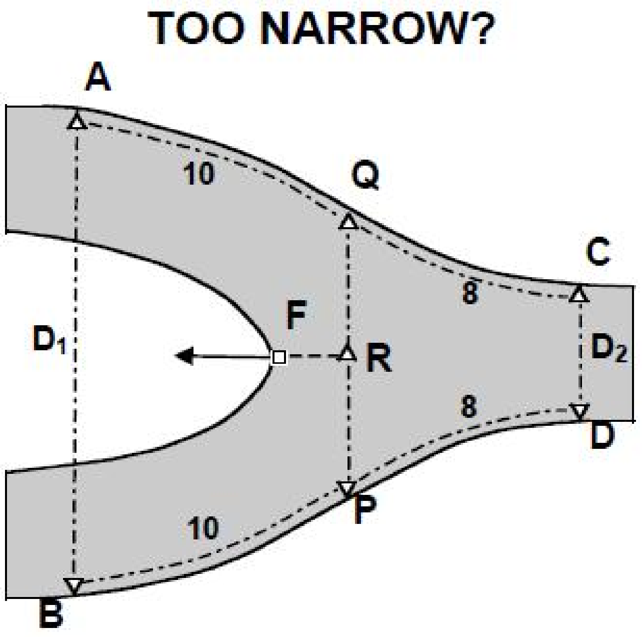


Figure 4-18: Remove Too Narrow Minutiae

Algorithm 15 Removal of too narrow minutiae

```
1: procedure REMOVAL OF TOO NARROW MINUTIAE ($F$)
2:     T = 180 - direction(F)
3:     R = translate(F, 3 pixels, T)
4:     Q = findEdge(R, Up, 12 pixels)
5:     P = findEdge(R, Down, 12 pixels)
6:     Pts = traceContour(Q, 10 pixels)
7:     A = Pts(10)
8:     Pts = traceContour(Q, -8 pixels)
9:     C = Pts(8)
10:    Pts = traceContour(P, 10 pixels)
11:    B = Pts(10)
12:    Pts = traceContour(P, -8 pixels)
13:    D = Pts(8)
14:    D1 = distance(A, B)
15:    D2 = distance(C, D)
16:    if $D1/D2 <= 2.25$ then
17:        remove($F$)
18:    end if
19: end procedure
```

The previous step tests for candidate minutiae that are too wide. This step tests for points that are on structures that are too narrow. This is typical, for example, of pores in the friction skin. Figure 4-18 illustrates this test. Starting with the candidate minutia point, F, its coordinates are translated 3 pixels (PORESTRANSR) opposite the minutia's direction. The top edge and bottom edges of the enveloping structure are then located at (Q, P). From these two points, the edge is traced to the left 10 pixels (PORESSTEPFWD) and to the right 8 pixels (PORESSTEPBWD). The points at the end of the 10 pixel contours are stored (A, B), and the points at the end of the 8-pixel contours are stored (C, D). Next, distances are computed between these pairs of points, and the ratio (D1/D2) is computed. If the ratio is 2.25 (PORESMAXRATIO), then the minutia point is removed from the candidate list. In fact, if the process fails to find any of the points in the illustration, then the candidate minutia is removed. It should be noted that, mindtct, only searches for minutiae that are too narrow within high-curvature regions or regions where ridge flow direction is non-determinable.

4.1.3.8 Count Neighbor Ridges

Fingerprint minutiae matchers often use information in addition to just the points themselves. Ancillary information usually includes the minutia's direction, its type, and it may include information pertaining to minutiae neighbors. Beyond a minutia's position, direction, and type, there are no standard neighbor schemes. Different AFIS systems use different neighbor topologies and attributes. One common attribute is the number of intervening ridges (called ridge crossings) between a minutia and each of its neighbors. For example, the FBI's IAFIS uses ridge crossings between a minutia and its 8 nearest neighbors, where each neighbor is the closest within a specified octant. Up to 5 nearest neighbors (MAXNBRS) are reported. Given a minutia point, the closest neighbors below (in the same pixel column), and to the right (within entire pixel columns) in the image are selected. These nearest neighbors are sorted in order of their direction, starting with vertical and working clockwise. Using this topology, ridge counts are computed and recorded between a minutia point and each of its nearest neighbors.

4.1.3.9 Assess Minutia Quality

One of the goals of developing this software package was to compute a quality/reliability to be associated with each detected minutia point. Even with the lengthy list of removal steps above, false minutiae potentially remain in the candidate list. A robust quality measure can help manage this in that false minutiae should be assigned a lower quality than true minutiae. Through dynamic thresholding, a trade off between retaining false minutiae and throwing away true minutiae may be determined.

To this end, mindtct, computes and reports minutiae qualities. Two factors are combined to produce a quality measure for each detected minutia point. The first factor, L, is taken directly from the location of the minutia point within the quality map described in Section 4.1.3.3. One of five quality levels is initially assigned, with 4 being the highest quality and 0 being the lowest. The second factor is based on simple pixel intensity statistics (mean and standard deviation) within the immediate neighborhood of the minutia point. The size of the neighborhood is set to 11 pixels (RADIUSMM). This is sufficiently large to contain generous portions of an average ridge and valley. A high quality region within a fingerprint image will have significant contrast that will cover the full grayscale spectrum. Consequently, the mean pixel intensity of the neighbor hood will be very close to 127. For similar reasons, the pixel intensities of an ideal neighborhood will have a standard deviation greater than or equal to 64. Using this logic, the following reliability measure, R, is calculated given neighborhood mean and standard deviation This results in a quality value on the range .01 to .99. A low quality value represents a minutia detected in a lower quality region of the image, whereas a high quality value represents a minutia detected in a higher quality region.

4.1.3.10 Output Minutiae

Upon completion, resulting minutiae outputs to a file. If the input file was an ANSI/NIST formatted file, mindtct adds two new records and writes a new ANSI/NIST formatted file to oroot.mdt, where oroot is passed as a parameter to mindtct. The new records are a Type-9 record, holding the detected minutiae, is constructed and inserted along with a Type-13 or Type-14 record, holding the image binarization results. If the input image is of a latent fingerprint, then the binarization results are stored in a Type-13 record; otherwise, the image results are stored in a Type-14 record. It should be noted that the minutiae in the Type-9 record are formatted in the NIST-assigned fields according to the ANSI/NIST standard.[17] The utilities, an2k7toiaf and iaf2an2k7, as described in the Reference Manual of this document may be used to convert between these fields and the FBI/IAFISassigned fields 13-23.[17]

If the input file is not in ANSI/NIST format, the resulting minutiae can be accessed in the text file oroot.min and there is no ANSI/NIST output file created but a raw pixel file is created that has the image binarization results. For all input types the detected minutiae are also written to a text file oroot.xyt that is formatted for use with the bozorth3 matcher. This file has one space delimited line per minutiae containing its x and y coordinate, direction angle theta, and the minutiae quality. The output minutiae are in the ANSI/NIST format which has the origin at the bottom left of the image and directions pointing out and away from the ridge ending or bifurcation valley. There is an option to output the minutiae in the M1 (ANSI INCITS 378-72 3004) representation which has the pixel origin at the top left of the image and directions pointing up the ridge ending or bifurcation valley. If this option (-m1) is used when running mindtct it should also be used by bozorth3 when matching the minutiae files. The last text output file, oroot.min, contains a formatted listing of attributes associated with each detected minutiae in the fingerprint image. Among these attributes are the minutia's pixel coordinate location, its direction, and type.

4.2 Fingerprint Matching

An algorithm and utility for taking features extracted from two fingerprints (minutiae detection) and matching them together for either the purpose of one-to-one verification or one-to-many identification. This type of algorithm is commonly referred to as a fingerprint matcher.

The BOZORTH3 matching algorithm computes a match score between the minutiae from any two fingerprints to help determine if they are from the same finger. It's a modified version of a fingerprint matcher written by Allan S. Bozorth while at the FBI. The early version of the matching algorithm that NIST has used internally was named bozorth98.[5][7][8] The BOZORTH3 matcher is functionally the same as the bozorth98 matcher, improvements have been made to remove bugs in the code (specifically memory leaks in statically defined variables) and improve the speed of the matcher. The BOZORTH3 matcher using only the location (x,y) and orientation (theta) of the minutia points to match the fingerprints. The matcher is rotation and translation invariant. The matcher builds separate tables for the fingerprints being matched that define distance and orientation between minutia in each fingerprint. These two tables are then compared for compatibility and a new table is constructed that stores information showing the inter-fingerprint compatibility. The inter-finger compatibility table is used to create a match score by looking at the size and number of compatible minutia clusters. A detailed description of the BOZORTH3 matching algorithm is described below.

4.2.1 Background

An FBI employee by the name of Allan S. Bozorth, had set off on an effort to investigate the notion of a translation and rotation invariant algorithm for matching two fingerprints to each other. It was around this time that the construction of the demonstration display began. In the demonstration, the Home Office algorithm for minutiae detection (the algorithm on which MINDTCT is based) was used; and when the need for a fingerprint matcher arose, Allans algorithm was selected and integrated into both the IAFIS and NCIC portions of the display. The matcher performed at an adequate level to support the demonstration where a guest could be enrolled and searched against a very small background. Much to Allans surprise and credit, this algorithm has since been extensively used as a technology benchmark by NIST to support work under the U.S. Patriot Act.[5] The algorithm has been shown to perform respectably well for both verification and identification applications. In honor of Allans hard work and accomplishments, NIST has chosen to name this matcher the Bozorth Matcher, or in short Bozorth. A description of the algorithm follows.

4.2.2 Bozorth Algorithm

Two key things are important to note regarding this fingerprint matcher:

1. Minutia features are exclusively used and limited to location (x,y) and orientation t,represented as x,y,t.

2. The algorithm is designed to be rotation and translation invariant.

The algorithm is comprised of three major steps:

1. Construct Intra-Fingerprint Minutia Comparison Table.
   a. One table for the probe fingerprint and one table for each gallery fingerprint to be matched against

2. Construct an Inter-Fingerprint Compatibility Table.
   a. Compare a probe prints minutia comparison table to a gallery prints minutia comparison table and construct a new compatibility table

3. Traverse the Inter-Fingerprint Compatibility Table
   a. Traverse and link table entries into clusters
   b. Combine compatible clusters and accumulate a match score

4.2.2.1 Construct Intra-Fingerprint Minutia Comparison Tables

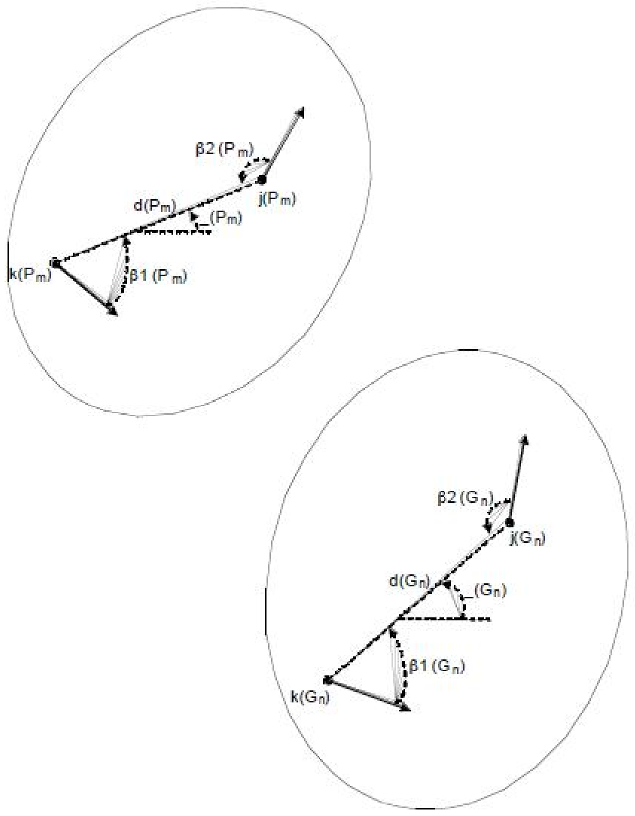


Figure 4-19: inter-minutia measurements

The first step in the Bozorth Matcher is to compute relative measurements from each minutia in a fingerprint to all other minutia in the same fingerprint. These relative measurements are stored in a minutia comparison table and are what provide the algorithms rotation and translation invariance. Figure 4-19 illustrates the inter-minutia measurements that are used. There are two minutiae shown in this example. Minutia k is in the lower left of the fingerprint and is depicted by the dot representing location (xk,yk) and the arrowed line pointing down and to the right representing orientation tk. A second minutia j is in the upper right with orientation pointing up and to the right. To account for relative translational position, the distance dkj is computed between the two minutia locations. This distance will remain relatively constant between corresponding points on two different finger impressions regardless of how much shifting and rotating may exist between the two prints. To make relative rotational measurements is a bit more involved. The objective for each of the minutiae in the pair-wise comparison is to compute the angle between each minutias orientation and the intervening line between both minutiae. This way, these angles remain relatively constant to the intervening line regardless of how much the fingerprint is rotated. In the illustration above, the angle kj of the intervening line between minutia k and j is computed by taking the arctangent of the slope of the intervening line. Angles k and j are computed relative to the intervening line as shown by incorporating kj and each minutias orientation t. It should be noted that the point-wise comparison is conducted on minutia positions sorted first on xcoordinate, then on y-coordinate, and that all orientations are limited to the period (-180deg, 180deg) with 0deg pointing horizontal to the right and increasing degrees proceeding counter clockwise. For each pair-wise minutia comparison, an entry is made into a comparison table Entries are stored in the comparison table in order of increasing distance and the table is trimmed at the point in which a maximum distance threshold is reached. Making these measurements between pairs of minutiae, a comparison table must be constructed for each and every fingerprint you wish to match with or against.

4.2.2.2 Construct Inter-Fingerprint Minutia Comparison Tables

The next step in the Bozorth matching algorithm is to take the minutia comparison tables from two separate fingerprints and look for compatible entries between the two tables. Figure 4-20 depicts two impressions of the same fingerprint with slight differences in both rotation and scale. Two corresponding minutia points are shown in each fingerprint. The upper left print represents a probe print in which all its minutiae have been pair-wised compared with relative measurements stored in minutia comparison table P. The measurements computed from the particular pair of minutia in this example have been stored as the mth entry in table P, denoted Pm. The notation of individual values stored in the table are represented as lookup functions on a given table entry. For example, the index of the lower left minutia is stored in table entry Pm and is referenced as k(Pm), while the distance between the two minutiae is also stored in table entry Pm and is referenced as d(Pm). The lower right fingerprint represents a gallery print, and uses similar notation, except that all its pair-wise minutia comparisons have been stored in table G, and the measurements made on the two corresponding minutia in the gallery print have been stored in table entry Gn. The following three tests are conducted to determine if table entries Pm and Gn are compatible. The first test checks to see if the corresponding distances are within a specified tolerance Td. The last two tests check to see if the relative minutia angles are within a specified tolerance T.

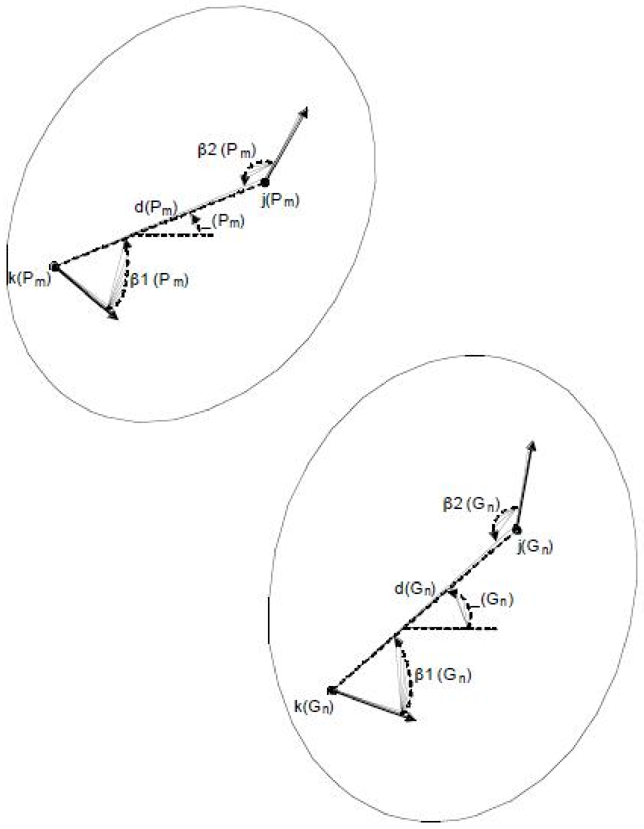


Figure 4-20: intra-minutia measurements

If the relative distance and minutia angles between the two comparison table entries are within acceptable tolerance, then the following entry is entered into a compatibility table: A compatibility table entry therefore incorporates two pairs of minutia, one pair from the probe fingerprint (k(Pm), j(Pm)) and the other from the gallery fingerprint (k(Gn), j(Gn)). The entry into the compatibility table then indicates that k(Pm) corresponds with k(Gn) and j(Pm) corresponds with j(Gn).

4.2.2.3 Traverse the Inter-Fingerprint Compatibility Table

At this point in the process, we have constructed a compatibility table which consists of a list of compatibility association between two pairs of potentially corresponding minutiae. These associations represent single links in a compatibility graph. To determine how well the two fingerprints match each other, a simple goal would be to traverse the compatibility graph finding the longest path of linked compatibility associations. The match score would then be the length of the longest path. There are some serious challenges to such a simple approach. These include:

1. The compatibility table is not a coherent graph, but rather a disjoint collection of single links within a graph.
2. Each node in the graph is potentially linked to many other nodes.
3. This leads to the potential for circuits.
4. There is no obvious root node in the graph that can be predicted to lead to the maximum path.
5. Occlusions and/or voids within either of the two fingerprints being matched will cause discontinuities in the graph.

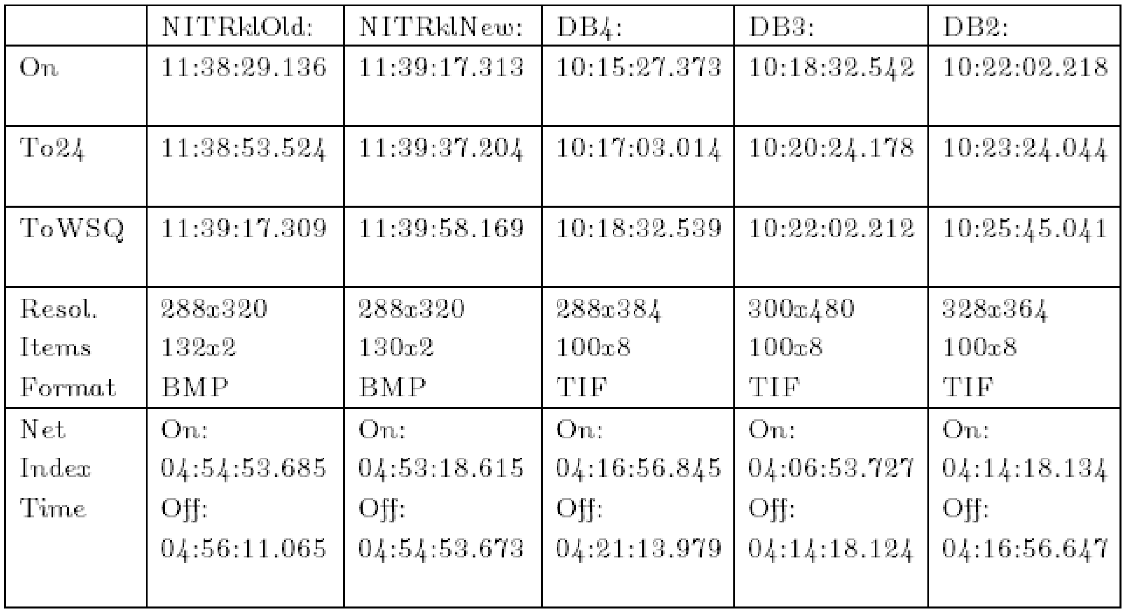
To account for these issues, Allan Bozorth implemented an algorithm that processes the compatibility table so that traversals are initiated from various staring points. As traversals are conducted, portions or clusters of the compatibility graph are created by linking entries in the table. Once the traversals are complete, compatible clusters are combined and the number of linked table entries across the combined clusters is accumulated to form the match score. The larger the number of linked compatibility associations, the larger the match score.

Chapter 5

Experimental Results

5.1 Performance

The following result in Figure 5-1 shows the total time required to convert entire database (NIST DB2, DB3, DB4 [27]) of images from one format to another and total time to Index entire databases.

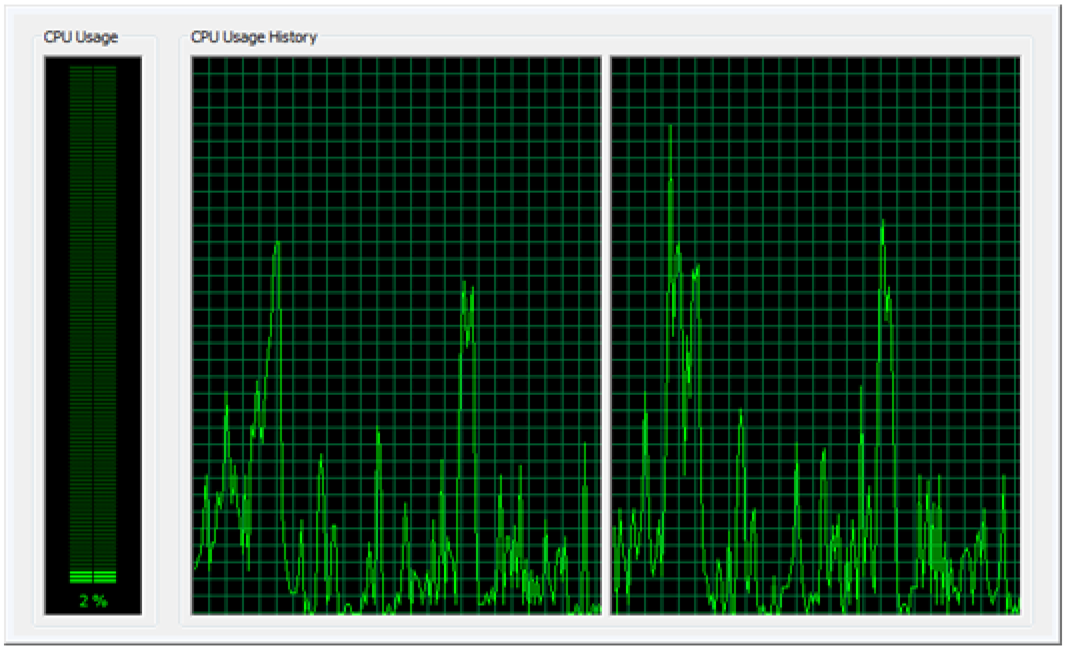


	NITRklOld:	NITRklNew:	DB4:	DB3:	DB2:
On	11:38:29.136	11:39:17.313	10:15:27.373	10:18:32.542	10:22:02.218
To24	11:38:53.524	11:39:37.204	10:17:03.014	10:20:24.178	10:23:24.044
ToWSQ	11:39:17.309	11:39:58.169	10:18:32.539	10:22:02.212	10:25:45.041
Resol. Items Format	288x320 132x2 BMP	288x320 130x2 BMP	288x384 100x8 TIF	300x480 100x8 TIF	328x364 100x8 TIF
Net Index Time	On: 04:54:53.685 Off: 04:56:11.065	On: 04:53:18.615 Off: 04:54:53.673	On: 04:16:56.845 Off: 04:21:13.979	On: 04:06:53.727 Off: 04:14:18.124	On: 04:14:18.134 Off: 04:16:56.647

Figure 5-1: Full database DB4, DB3, DB2 indexing speed test

The following result in Figure 5-2 shows the CPU Usage and Memory consumption during Image conversion and Indexing of entire databases.

CPU Usage during indexing: (Not Completely Utilized)



RAM Usage

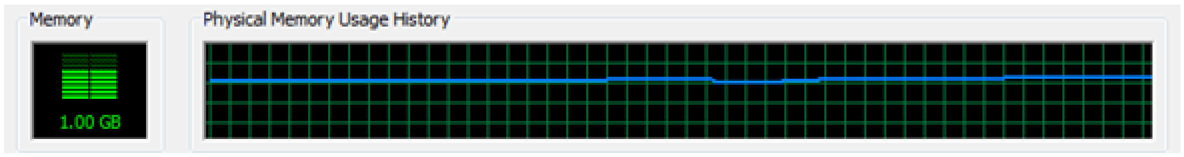
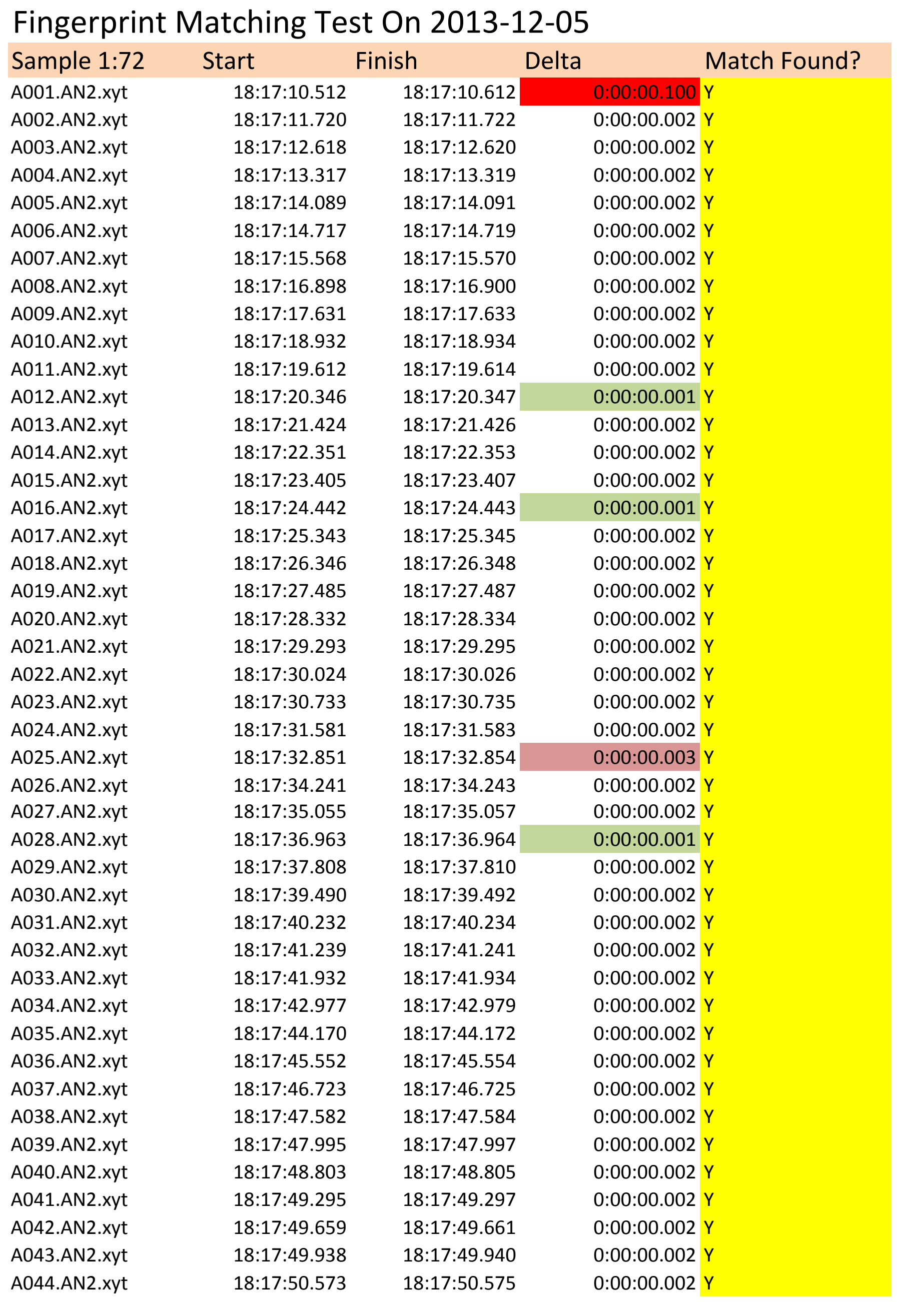


Figure 5-2: CPU state during Full database DB4, DB3, DB2 indexing

The following result in Figure 5-3 shows the time taken from start to finish of matching 1 fingerprint in a group of 72 fingerprints. Here all 44 samples were tested.

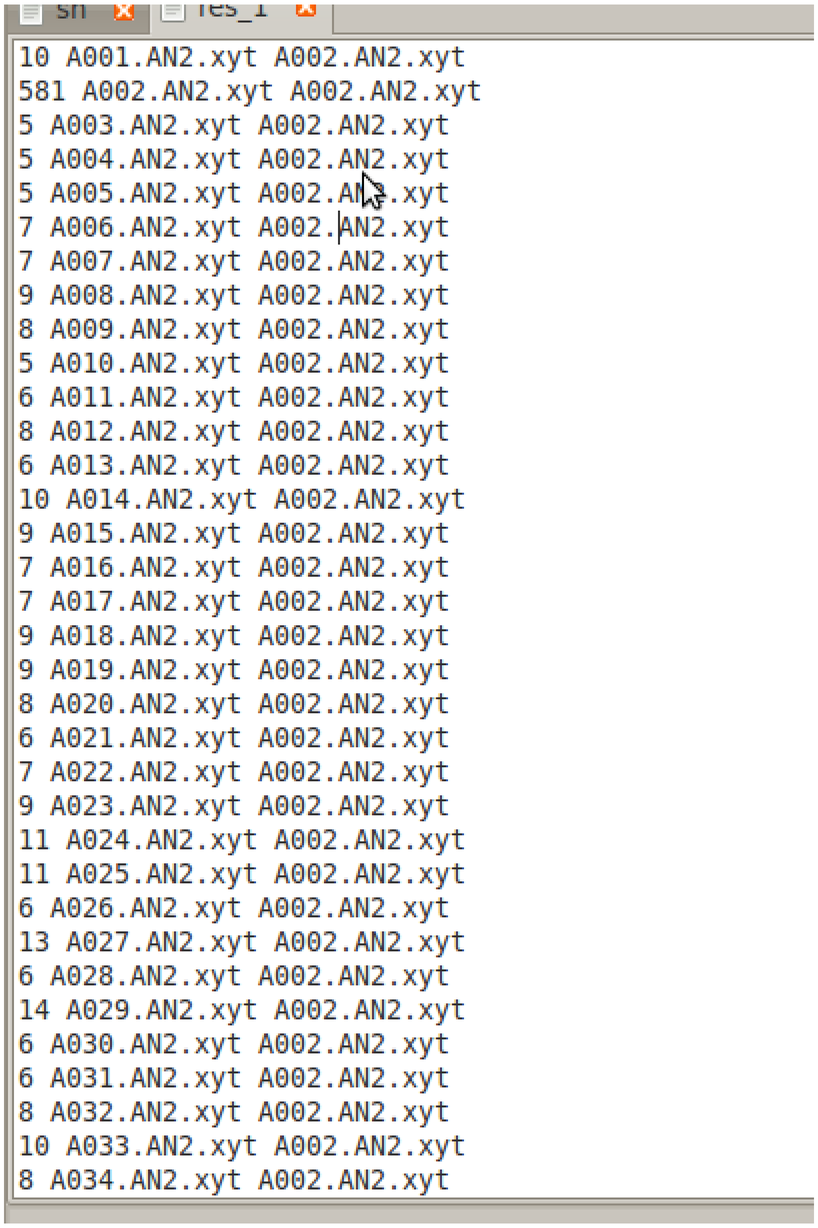


Fingerprint Matching Test On 2013-12-05

Sample 1:72	Start	Finish	Delta	Match Found?
A001.AN2.xyt	18:17:10.512	18:17:10.612	0:00:00.100	Y
A002.AN2.xyt	18:17:11.720	18:17:11.722	0:00:00.002	Y
A003.AN2.xyt	18:17:12.618	18:17:12.620	0:00:00.002	Y
A004.AN2.xyt	18:17:13.317	18:17:13.319	0:00:00.002	Y
A005.AN2.xyt	18:17:14.089	18:17:14.091	0:00:00.002	Y
A006.AN2.xyt	18:17:14.717	18:17:14.719	0:00:00.002	Y
A007.AN2.xyt	18:17:15.568	18:17:15.570	0:00:00.002	Y
A008.AN2.xyt	18:17:16.898	18:17:16.900	0:00:00.002	Y
A009.AN2.xyt	18:17:17.631	18:17:17.633	0:00:00.002	Y
A010.AN2.xyt	18:17:18.932	18:17:18.934	0:00:00.002	Y
A011.AN2.xyt	18:17:19.612	18:17:19.614	0:00:00.002	Y
A012.AN2.xyt	18:17:20.346	18:17:20.347	0:00:00.001	Y
A013.AN2.xyt	18:17:21.424	18:17:21.426	0:00:00.002	Y
A014.AN2.xyt	18:17:22.351	18:17:22.353	0:00:00.002	Y
A015.AN2.xyt	18:17:23.405	18:17:23.407	0:00:00.002	Y
A016.AN2.xyt	18:17:24.442	18:17:24.443	0:00:00.001	Y
A017.AN2.xyt	18:17:25.343	18:17:25.345	0:00:00.002	Y
A018.AN2.xyt	18:17:26.346	18:17:26.348	0:00:00.002	Y
A019.AN2.xyt	18:17:27.485	18:17:27.487	0:00:00.002	Y
A020.AN2.xyt	18:17:28.332	18:17:28.334	0:00:00.002	Y
A021.AN2.xyt	18:17:29.293	18:17:29.295	0:00:00.002	Y
A022.AN2.xyt	18:17:30.024	18:17:30.026	0:00:00.002	Y
A023.AN2.xyt	18:17:30.733	18:17:30.735	0:00:00.002	Y
A024.AN2.xyt	18:17:31.581	18:17:31.583	0:00:00.002	Y
A025.AN2.xyt	18:17:32.851	18:17:32.854	0:00:00.003	Y
A026.AN2.xyt	18:17:34.241	18:17:34.243	0:00:00.002	Y
A027.AN2.xyt	18:17:35.055	18:17:35.057	0:00:00.002	Y
A028.AN2.xyt	18:17:36.963	18:17:36.964	0:00:00.001	Y
A029.AN2.xyt	18:17:37.808	18:17:37.810	0:00:00.002	Y
A030.AN2.xyt	18:17:39.490	18:17:39.492	0:00:00.002	Y
A031.AN2.xyt	18:17:40.232	18:17:40.234	0:00:00.002	Y
A032.AN2.xyt	18:17:41.239	18:17:41.241	0:00:00.002	Y
A033.AN2.xyt	18:17:41.932	18:17:41.934	0:00:00.002	Y
A034.AN2.xyt	18:17:42.977	18:17:42.979	0:00:00.002	Y
A035.AN2.xyt	18:17:44.170	18:17:44.172	0:00:00.002	Y
A036.AN2.xyt	18:17:45.552	18:17:45.554	0:00:00.002	Y
A037.AN2.xyt	18:17:46.723	18:17:46.725	0:00:00.002	Y
A038.AN2.xyt	18:17:47.582	18:17:47.584	0:00:00.002	Y
A039.AN2.xyt	18:17:47.995	18:17:47.997	0:00:00.002	Y
A040.AN2.xyt	18:17:48.803	18:17:48.805	0:00:00.002	Y
A041.AN2.xyt	18:17:49.295	18:17:49.297	0:00:00.002	Y
A042.AN2.xyt	18:17:49.659	18:17:49.661	0:00:00.002	Y
A043.AN2.xyt	18:17:49.938	18:17:49.940	0:00:00.002	Y
A044.AN2.xyt	18:17:50.573	18:17:50.575	0:00:00.002	Y

Figure 5-3: Fingerprint matching test on DB2

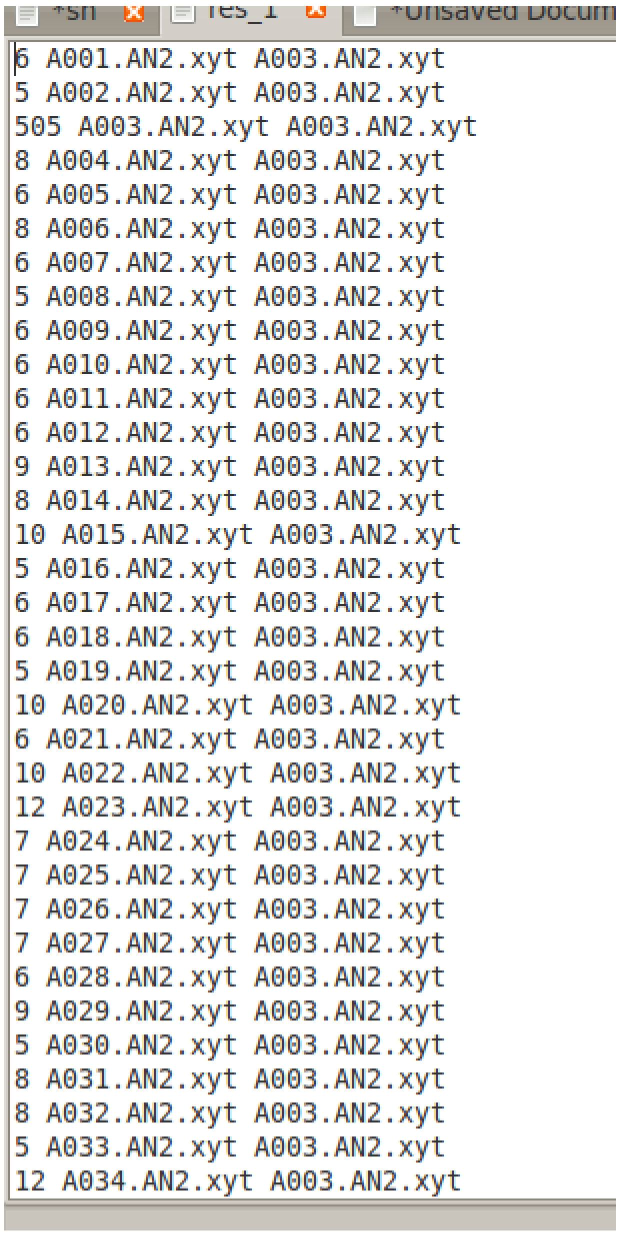
The following result in Figure 5-4 shows the score of matching 1 fingerprint in a group of 72 fingerprints. Here as expected A002 fingerprint on match with A002 produces highest score.



```
10 A001.AN2.xyt A002.AN2.xyt
581 A002.AN2.xyt A002.AN2.xyt
5 A003.AN2.xyt A002.AN2.xyt
5 A004.AN2.xyt A002.AN2.xyt
5 A005.AN2.xyt A002.AN2.xyt
7 A006.AN2.xyt A002.AN2.xyt
7 A007.AN2.xyt A002.AN2.xyt
9 A008.AN2.xyt A002.AN2.xyt
8 A009.AN2.xyt A002.AN2.xyt
5 A010.AN2.xyt A002.AN2.xyt
6 A011.AN2.xyt A002.AN2.xyt
8 A012.AN2.xyt A002.AN2.xyt
6 A013.AN2.xyt A002.AN2.xyt
10 A014.AN2.xyt A002.AN2.xyt
9 A015.AN2.xyt A002.AN2.xyt
7 A016.AN2.xyt A002.AN2.xyt
7 A017.AN2.xyt A002.AN2.xyt
9 A018.AN2.xyt A002.AN2.xyt
9 A019.AN2.xyt A002.AN2.xyt
8 A020.AN2.xyt A002.AN2.xyt
6 A021.AN2.xyt A002.AN2.xyt
7 A022.AN2.xyt A002.AN2.xyt
9 A023.AN2.xyt A002.AN2.xyt
11 A024.AN2.xyt A002.AN2.xyt
11 A025.AN2.xyt A002.AN2.xyt
6 A026.AN2.xyt A002.AN2.xyt
13 A027.AN2.xyt A002.AN2.xyt
6 A028.AN2.xyt A002.AN2.xyt
14 A029.AN2.xyt A002.AN2.xyt
6 A030.AN2.xyt A002.AN2.xyt
6 A031.AN2.xyt A002.AN2.xyt
8 A032.AN2.xyt A002.AN2.xyt
10 A033.AN2.xyt A002.AN2.xyt
8 A034.AN2.xyt A002.AN2.xyt
```

Figure 5-4: Another Fingerprint matching test result with scores on DB2

The following result in Figure 5-5 shows the score of matching 1 fingerprint in a group of 72 fingerprints. Here as expected A003 fingerprint on match with A003 produces highest score.



```
6 A001.AN2.xyt A003.AN2.xyt
5 A002.AN2.xyt A003.AN2.xyt
505 A003.AN2.xyt A003.AN2.xyt
8 A004.AN2.xyt A003.AN2.xyt
6 A005.AN2.xyt A003.AN2.xyt
8 A006.AN2.xyt A003.AN2.xyt
6 A007.AN2.xyt A003.AN2.xyt
5 A008.AN2.xyt A003.AN2.xyt
6 A009.AN2.xyt A003.AN2.xyt
6 A010.AN2.xyt A003.AN2.xyt
6 A011.AN2.xyt A003.AN2.xyt
6 A012.AN2.xyt A003.AN2.xyt
9 A013.AN2.xyt A003.AN2.xyt
8 A014.AN2.xyt A003.AN2.xyt
10 A015.AN2.xyt A003.AN2.xyt
5 A016.AN2.xyt A003.AN2.xyt
6 A017.AN2.xyt A003.AN2.xyt
6 A018.AN2.xyt A003.AN2.xyt
5 A019.AN2.xyt A003.AN2.xyt
10 A020.AN2.xyt A003.AN2.xyt
6 A021.AN2.xyt A003.AN2.xyt
10 A022.AN2.xyt A003.AN2.xyt
12 A023.AN2.xyt A003.AN2.xyt
7 A024.AN2.xyt A003.AN2.xyt
7 A025.AN2.xyt A003.AN2.xyt
7 A026.AN2.xyt A003.AN2.xyt
7 A027.AN2.xyt A003.AN2.xyt
6 A028.AN2.xyt A003.AN2.xyt
9 A029.AN2.xyt A003.AN2.xyt
5 A030.AN2.xyt A003.AN2.xyt
8 A031.AN2.xyt A003.AN2.xyt
8 A032.AN2.xyt A003.AN2.xyt
5 A033.AN2.xyt A003.AN2.xyt
12 A034.AN2.xyt A003.AN2.xyt
```

Figure 5-5: Another Fingerprint matching test result with scores on DB2

5.2 Real-time Experiment

Outline of fingerprint enrollmnt module is shown in Figure 5-6

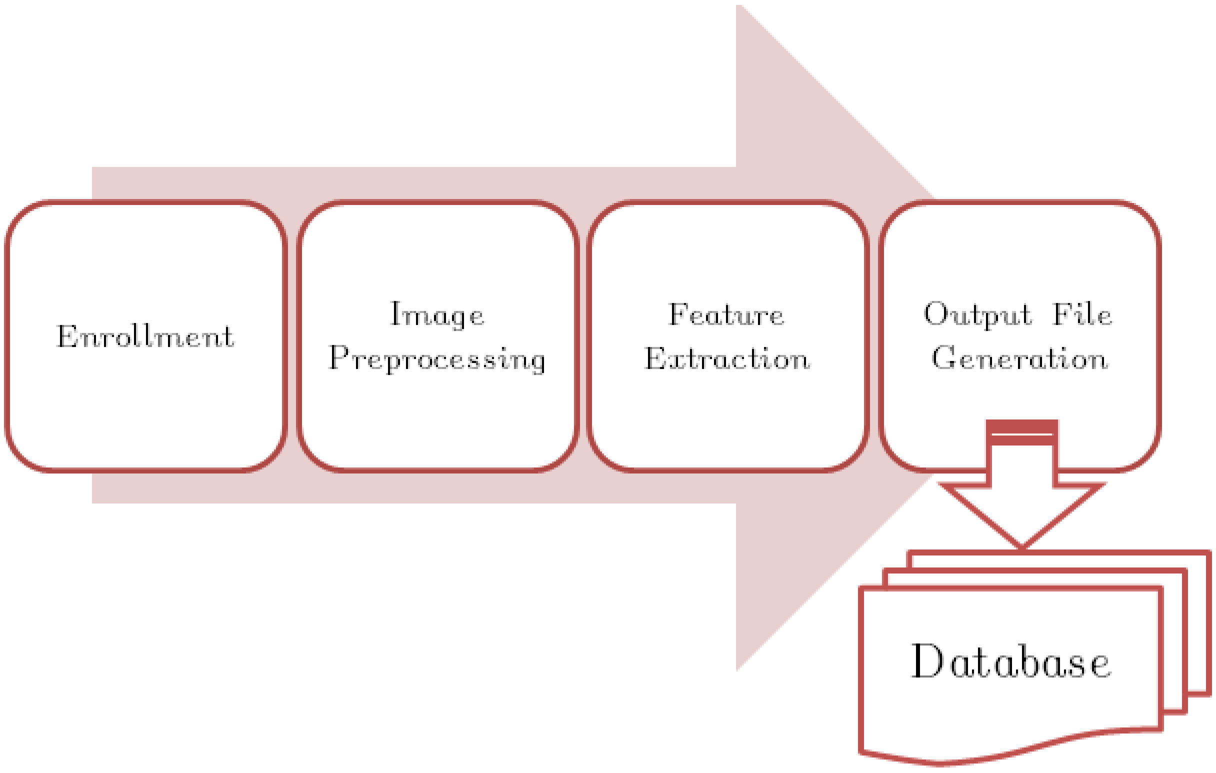


Figure 5-6: Fingerprint enrollment outline

In real time first step is to capture image, we use a better hamster as shown in Figure 5-7 to capture Fingerprint Image

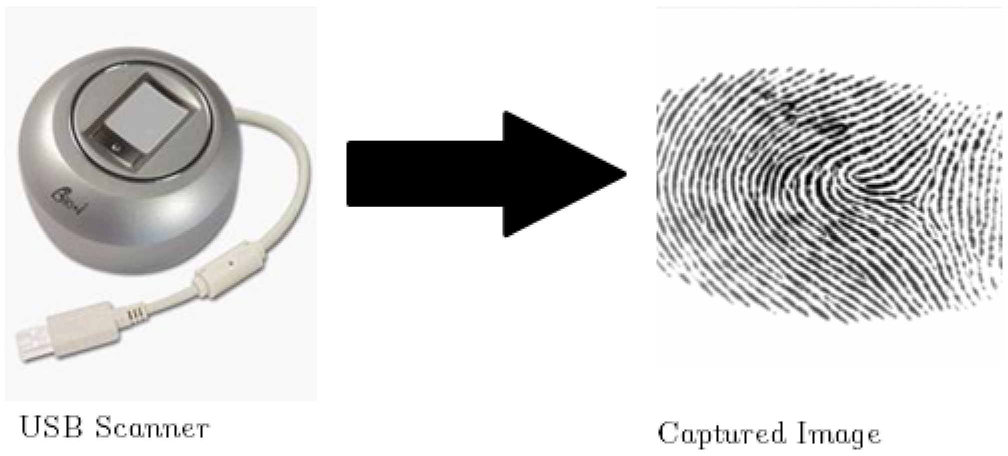


Figure 5-7: Hamster fingurprint capture

Here are the visual output generated from simulation of Fingerprint Preprocessor

Module. In Figure 5-8 one can see visual output of preprocessing procedures as described in Chapter 4.

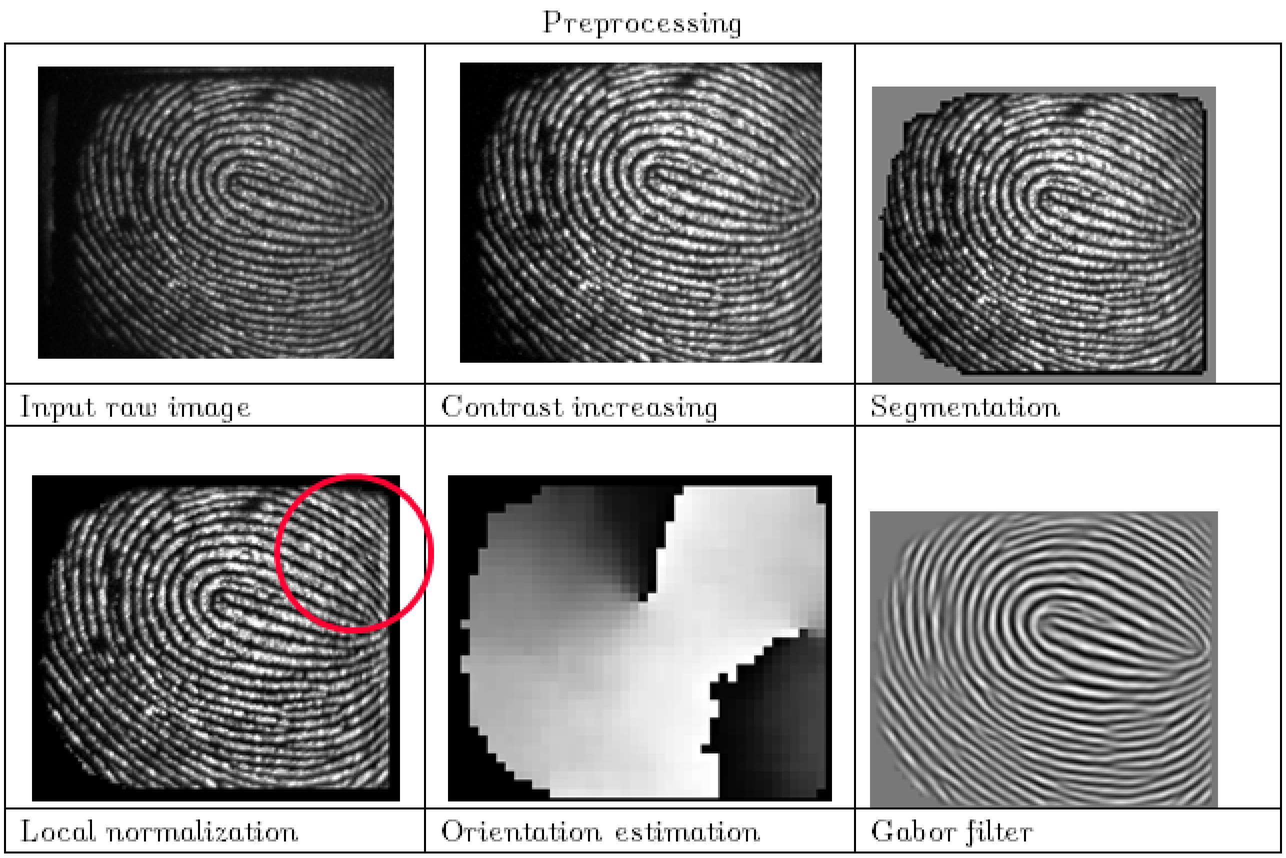


Figure 5-8: Preprocessing input image

Here are the visual output generated from simulation of Minutiae Detector Module. In Figure 5-9 one can see visual output of how minutiae are extracted as described in Chapter 4.

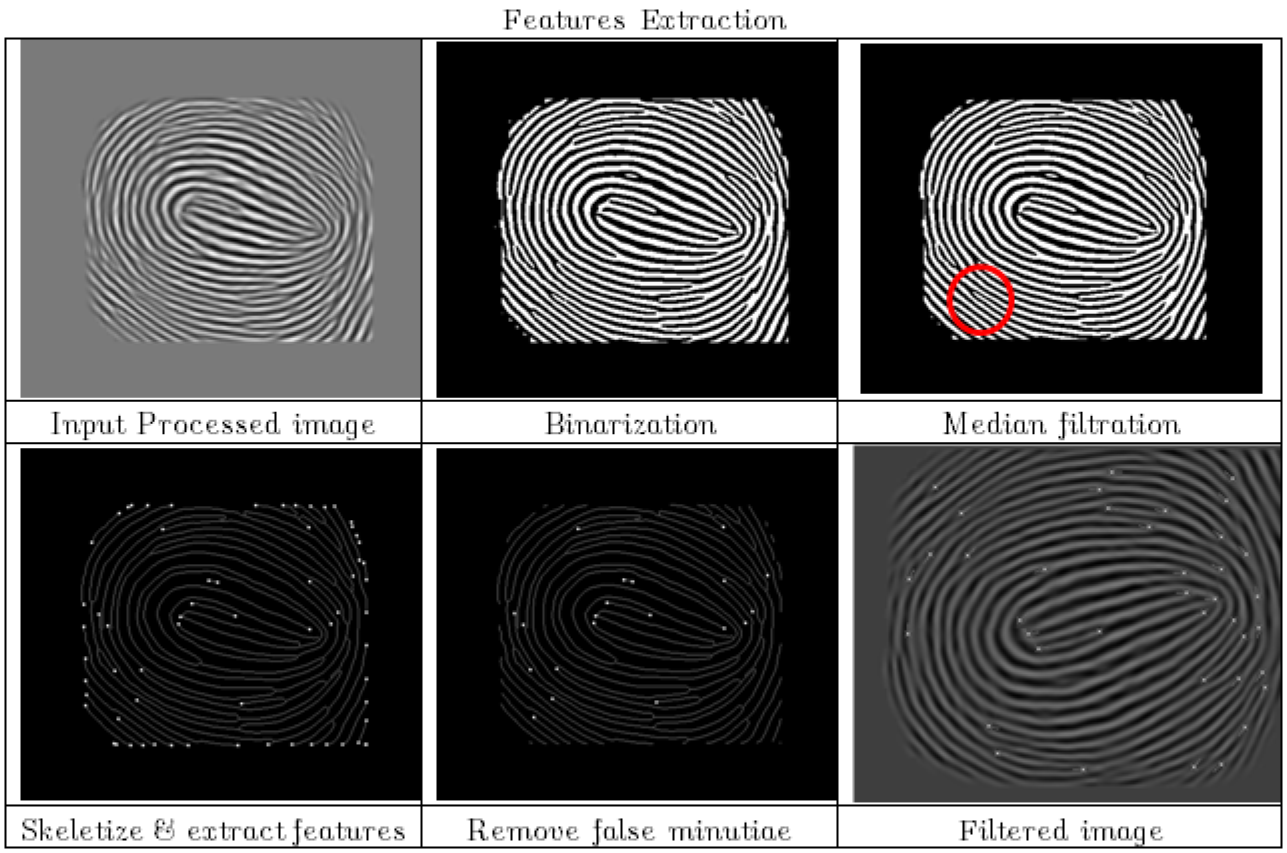
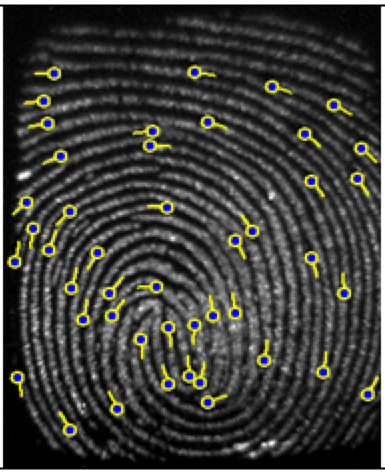


Figure 5-9: Process involved in feature extraction

Fingerprint with features and Figure 5-10 describes what output file will store from selected minutiae.

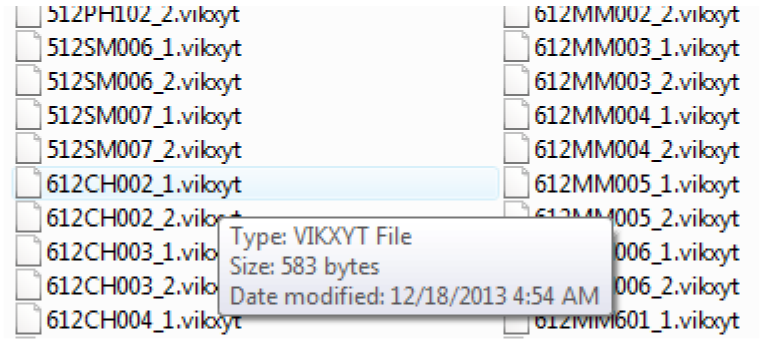


Feature vector includes:

- Minutiaes position
- Minutiaes orientation
- Information about minutiaes interposition
- Information about ridge pattern structure

Figure 5-10: Storing feature vector in Output file

Output fingerprint data is stored in speacially defined vikxyt vector file, which is encoded and compressed for security reasons and collection of such file is shown in Figure 5-11



```
512PH102_2.vikxyt          612MM002_2.vikxyt
512SM006_1.vikxyt          612MM003_1.vikxyt
512SM006_2.vikxyt          612MM003_2.vikxyt
512SM007_1.vikxyt          612MM004_1.vikxyt
512SM007_2.vikxyt          612MM004_2.vikxyt
612CH002_1.vikxyt          612MM005_1.vikxyt
612CH002_2.vik             005_2.vikxyt
612CH003_1.vik             006_1.vikxyt
612CH003_2.vik             006_2.vikxyt
612CH004_1.vikxyt          601_1.vikxyt

Type: VIKXYT File
Size: 583 bytes
Date modified: 12/18/2013 4:54 AM
```

Figure 5-11: Vectors stored in file

Chapter 6

Conclusion

In this thesis, elementary indexing approach is proposed and implemented. Also kD-tree was implemented. Complex systems like fingerprint authentication systems constitute many modules. Major modules like Fingerprint preprocessor, Minutiae extractor and Fingerprint matching were explored, implemented and simulation of realtime fingerprint authentication system was performed. The preliminary results on testing with 40+ volunteers were obtained as expected. This project opened new doors and gave a insight that real time fingerprint authentication systems are professional, accurate and complicated. elementary indexing approach of grouping people from a large group was incorporated. The model was successfully implemented and simulated in scheduled time.

Bibliography

[1] JAIN A. K. AND PRABHKAR S. 2001. *Fingerprint Matching Using Minutiae and Texture Features. Proceeding of International Conference on Image Processing (ICIP)*, pp. 282-285.

[2] AGGARWAL G., RATHA N. K., TSAI-YANG J., AND BOLLE R. M. 2008. *Gradient based textural characterization of fingerprints. In proceedings of IEEE International conference on Biometrics: Theory, Applications and Systems.*

[3] JAIN A. K. AND PRABHKAR S. 2001. *Fingerprint Matching Using Minutiae and Texture Features. Proceeding of International Conference on Image Processing (ICIP)*, pp. 282-285.

[4] CHIKKERUR S., PANKANTI S., JEA A., AND BOLLE R. 2006. *Fingerprint Representation using Localized Texture Features. The 18th International Conference on Pattern Recognition.*

[5] JHAT Z. A., MIR A. H. AND RUBAB S. 2011. *Fingerprint Texture Feature for Discrimination and Personal Verification. International Journal of Security and its Applications*, Vol. 5, No. 3.

[6] V. Dixit, Deepti Singh, Parul Raj, M. Swathi, P. Gupta, *kd-tree based fingerprint identification system* 01/2008; DOI:10.1109/IWASID.2008.4688340

[7] JAIN A. K., PRABHKAR S., HONG L., AND PANKANTI S. 2000. *Filterbank-Based Fingerprint Matching. IEEE Transactions on Image Processing*, Vol. 9, pp. 846-853.

[8] ZHENGU O., FENG J., SU F., AND CAI A., 2006. *Fingerprint Matching with Rotation-Descriptor Texture Features. The 18th International Conference on Pattern Recognition*, pp. 417-420.

[9] Johan De Boer, Asker M Bazen, Sabih H Gerez, *Indexing fingerprint databases based on multiple features Journal of The Acoustical Society of America-* J ACOUST SOC AMER. 12/2001;

[10] J.H. Wegstein, *A Semi-automated Single Fingerprint Identification System.* NBS Technical Note 481, April 1969.

[11] J.H. Wegstein, *Automated Fingerprint Identification.* NBS Technical Note 538, August 1970.

[12] J.H. Wegstein, *Manual and Computerized Footprint Identification.* NBS Technical Note 712, February 1972.

[13] R.T. Moore, *The Influence of Ink on The Quality of Fingerprint Impressions.* NBS Technical Report NBSIR 74-627, December 1974.

[14] J.H. Wegstein, *The M40 Fingerprint Matcher.* NBS Technical Note 878, July 1975.

[15] J.H. Wegstein, and J.F. Rafferty *The LX39 latent Fingerprint Matcher.* NBS Special Publication 500-36, August 1978.

[16] R.T. Moore, *Results of Fingerprint Image Quality Experiments.* NBS Technical Report NBSIR 81-2298, June 1981.

[17] J.H. Wegstein, *Results of Fingerprint Image Quality Experiments.* NBS Technical Report NBSIR 81-2298, June 1981.

[18] R.T. Moore, *An Automated Fingerprint Identification System.* NBS Special Publication 500-89, February 1982.

[19] R.M. McCabe, and R.T. Moore, *Data Format for Information Interchange.* American National Standard ANSI/NBS-ICST 1-1986, August 1986.

[20] R.T. Moore *Automated Fingerprint Identification Systems - Benchmark Test of Relative Performance*. American National Standard ANSI/IAI 1-1988, February 1988.

[21] R.T. Moore *Automated Fingerprint Identification Systems Glossary of Terms and Acronyms*. American National Standard ANSI/IAI 2-1988, July 1988.

[22] R.T. Moore, R.M. McCabe, and R.A. Wilkinson *AFRS Performance Evaluation Tests*. NBS Technical Report NBSIR 88-3831, August 1988.

[23] C. Watson, *NIST Special Database 4: 8-bit Gray Scale Images of Fingerprint Image Groups*. NBS Technical Report NBSIR 88-3831, March 1992.

[24] C.L. Wilson, G.T. Candela, P.J. Grother, C.I. Watson, and R.A. Wilkinson, *Massively Parallel Neural Network Fingerprint Classification System.* Technical Report NISTIR 4880, July 1992.

[25] C. Wilson, M. Garris, C. Watson, A, Hicklin, *Studies of Fingerprint Matching Using the NIST Verification Test Bed (VTB).* Technical Report NISTIR 7020, July 2003.

[26] C. Watson, C. Wilson, M. Indovina, R. Snelick, K. Marshall, *Studies of One-to-One Matching with Vendor SDK Matchers*. Technical Report NISTIR 7119, July 2004.

[27] NIST Fingerprint Databases *Publicly available fingerprint databases* fingerprint.nist.gov.in

[28] GNU project - *free UNIX-like utilities*. Learn more at http://www.gnu.org.

[29] Linux - *a freely available clone of the UNIX operating system.* Learn more at http://www.linux.org.

[30] Leslie Lamport, *LaTeX: A Document Preparation System.* Addison Wesley, Massachusetts, 2nd Edition, 1994.