NEAR EAST UNIVERSITY

GRADUATE SCHOOL OF APPLIED AND SOCIAL SCIENCES

Software Development for Minutiae-Based Fingerprint Detection

Nehad Hammad

Master Thesis

Department of Computer Engineering

Nicosia - 2005

Nehad Hammad: Software Development for Minutiae-Based Fingerprint Detection

Approval of the Graduate School of Applied and Social Sciences

Prof. Dr. Fakhraddin Mamedov
Director

We certify this thesis is satisfactory for the award of the Degree of Master of Science in Computer Engineering

Examining Committee in charge:

Prof. Dr. Doğan İbrahim, Chairman of Committee, Computer Engineering Department, NEU

Assoc. Prof. Dr. Rahib Abiyev, Member, Computer Engineering Department, NEU

Assist. Prof. Dr. Doğan Haktanır, Member, Electrical and Electronic Engineering Department, NEU

Assoc. Prof. Dr. Adnan Khashman, Supervisor, Electrical and Electronic Engineering Department, NEU

ACKNOWLEDGEMENT

First, I would like to thank my supervisor Assoc. Prof. Dr. Adnan Khashman for his invaluable advice and belief in my work and myself over the course of this MSc. Degree.

Second, I would like to thank my family for their constant encouragement and support during the preparation of this thesis.

Finally, I would also like to thank all my friends for their advice and support specially Mr. Hakki Shalan.

ABSTRACT

Many Automatic Fingerprint Identification Systems (AFIS) are based on minutiae matching. Minutiae are the terminations and bifurcations of the ridge lines in a fingerprint image. A fingerprint image that has undergone binarization, followed by thinning, in order to extract the minutiae, contains hundreds of minutiae, all of which are not so vivid and obvious in the original image. Thus, the set of minutiae that is well-defined and more prominent than the rest should have given higher relevance and importance in the process of minutiae matching.

The work presented within this thesis introduces the technique to detecting fingerprints that is based on calculating distances and angles between consecutive minutiae and thus solves common problems with existing techniques; this technique will be realized using the Delphi software language.

Real life application will also be presented using this novel technique.

CONTENTS

INTRODUCTION

The use of one's fingerprints as a mean of identification had existed long before its common usage today in the field of criminal investigation. Prior to the nineteenth century, fingerprints were primarily used only as a signature for indicating authorship or ownership. Other applications were not acknowledged until about 1860 when William Hershel was regularly imprinting the handprints of those engaged in his contracts. It was not until 1881 when Henry Faulds recognized that fingerprints found at crime scenes may be used to identify the perpetrator. Further exploration into fingerprints followed when Sir Francis Galton began his research in the field and authored the first textbook on fingerprinting in 1892. As a result of the work of these individuals, fingerprinting was soon accepted by Scotland Yard and finally by the United States in 1910.

Today, fingerprints are perhaps the primary means of personal identification although there are many other unique characteristics of an individual that can be used. They include voiceprints, dental impressions, DNA, retinal patterns, and even the shape of the ear lobes. Although these other characteristics are as much unique to the individual as are fingerprints, they lack many advantages which fingerprints have especially for the criminal investigator and forensic scientist. Common to the other distinctive attributes, fingerprints are universal and unique. In other words, everyone has them and no two have ever been found to be identical. Fingerprints are also unchangeable. They are formed before birth and remain until decomposition of the skin occurs some time after death. Although some deformities may result from aging, manual labor, or scarring, the overall pattern always remains distinguishable. What make fingerprints preferable are that they can be easily attained, quickly classified, and are very likely to be found at crime scenes. Not only can they identify criminals but also casualties of disasters such as plane crashes. Another common application is in maintaining access control.

Many fingerprint classification methods, rely on point patterns in fingerprints which form ridges and bifurcations unique for each person. Point patterns belong to either the Wirbel class (whorl and twin loop) or the Lasso class (arch, tented arch, left loop, and right loop). It is useful in deciding when two fingerprints do not match.

Fingerprint images are not perfect every time they are acquired, which can negatively affect their matching integrity. Some of the factors that may contribute to this include smudging, dirt, cuts and bruises, angular variance, and differences in pressure applied to the acquisition device. These factors depend on other factors as well, like which method you will use to acquire your images. Some acquisition methods are more susceptible to problems than others.

The most common problem in any fingerprints detection technique is fingerprint rotation because the minutiae direction will be change when we rotate fingerprint but the newly developed fingerprints detection technique solves this problem by using distances and angles between minutiae because the distance and angle between any sequential minutiae do not change when the fingerprint orientation changes.

The aims of the work presented within this thesis are to:

- Develop a fingerprint detection method that solves problems with the existing techniques (e.g. rotation, fingerprint damage…etc).
- Develop software that simulates the new technique using Delphi.
- Realize the efficiency of the new technique via real life application (fingerprint detection of five persons).

Chapter one describes some characteristic of fingerprints and types of fingerprints. Fingerprints detection techniques that are commonly used will also be presented. Detection technique challenges and difficulties when you design new fingerprints detection techniques will be described.

Chapter two describes a minutiae-based method to detect fingerprints. The minutiae-based method technique depends mainly on minutiae structures, so each minutiae have different characteristics and shape. Most common fingerprint detection techniques depend on ridge ending and ridge bifurcation to catch minutiae.

Chapter three describes the original work of the author, where two fingerprint detection techniques will be developed.

Chapter four describes the real-life application of the newly developed technique by author for fingerprint detection technique represented using fingerprints of five persons to verify the effectiveness of the new technique.

Chapter five represents the software development for the fingerprints detection system. The most advantage of Borland Delphi7 will describe also contain the main steps for any one will using fingerprint software.

CHAPTER ONE
FINGERPRINT DETECTION TECHNIQUES

1.1. Overview

This chapter is a background chapter that describes some characteristic of fingerprints and types of fingerprint. Fingerprints detection techniques that are commonly used will also be presented. Detection technique challenges and difficulties when you design new fingerprints detection techniques will be described.

1.2. Fingerprint Characteristics

Fingerprint based identification is the oldest method which has been successfully used in numerous applications. Everyone is known to have unique immutable fingerprints. A fingerprint is made of a series of ridges and furrows on the surface of the finger. The uniqueness of a fingerprint can be determined by the pattern of ridges and furrows as well as the minutiae points. Minutiae points are local ridge characteristics that occur at either a ridge bifurcation or a ridge ending [1] (See Figure 1.1).

1.3. Fingerprint Classifications

Fingerprints can be classified into three categories based on their major central pattern. These patterns are the arch, loop, and whorl, which are shown in (Figure 1.2) [2].

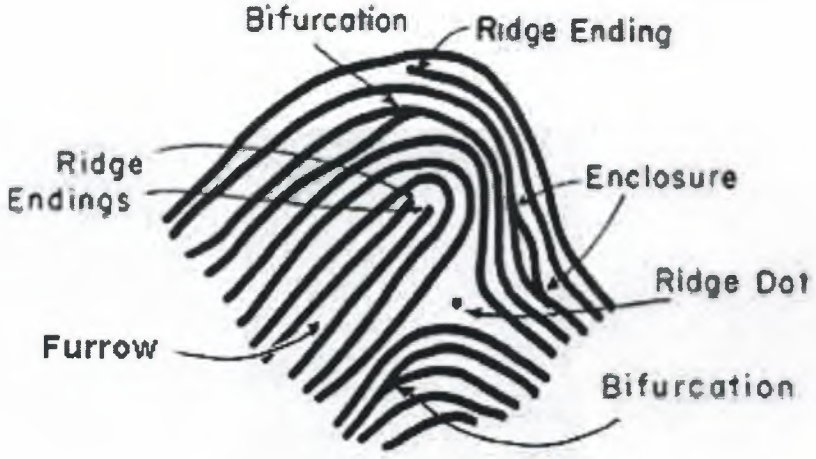


Figure 1.1 Minutiae examples [2].

Arch Loop Whorl

Figure1.2 Three major fingerprint classifiers [2].

1.4. Fingerprints Matching

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

Figure 1.3 Minutiae extraction [6].

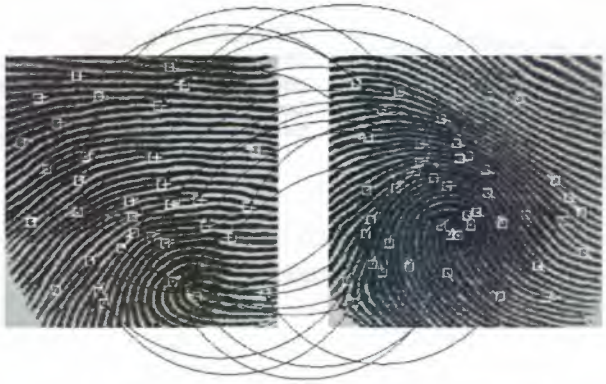


Figure 1.4 Matching fingerprints [6].

Fingerprint matching based on minutiae has problems in matching different sized minutiae patterns. Local ridge structures can not be completely characterized by minutiae [1].

In correlation-based matching, the input image is superimposed upon the template image, and the difference amongst the pixels is calculated for different alignments. At the optimum alignment in the x, y and θ coordinates (Where x is x-axis coordinate and y is y-axis coordinate and θ is angle), the best difference image is obtained, and if the intensity of the difference image is within a certain threshold, a match can be verified.

The minutiae of the fingerprint are small distinguishing characteristics that vary in location, orientation and types from finger to finger. The types of minutiae that can be identified are ridge endings, ridge bifurcations, lakes, independent ridges, islands, spurs and crossovers. Minutiae-based matching is the most commonly used form of matching, likely because this is the type of matching that has been performed by examiners years before automated biometric analysis existed. The two most easily identifiable and important types of minutiae are shown below as ridge endings and ridge bifurcations. The other forms of minutiae are not as reliable because some are harder to detect, and others are the type of minutiae than can be created falsely in the image processing stage.

After the minutiae points are found, their locations (x, y, θ) are stored. The input image is then aligned to the template image to the maximum number of minutiae pairs. If the two images have above a certain amount of minutiae that are matched, the user's identification is verified.

Figure 1.5 These two pictures show the common minutiae that identified during the minutiae detection stage [3].

The easiest way to implement this is using a thinned binary image that can be made in the pre-processing stage. Another way to utilize this system is not to match the minutiae of the input to the template.

The most recent method for fingerprint matching is the ridge feature-based method, the most efficient type of matching technique for low quality fingerprints. This procedure includes finding the more easily detectable (larger) features of a fingerprint, which are ridge pattern, orientation, ridge frequency, ridge shape, and texture information.

A lot was said concerning the alignment of the image. One method for aligning the image is to find the center point and to move it to the center of the image space. This provides a good amount of accuracy for the x and y coordinate matching, and after aligning the θ coordinate, the two images should be aligned in a relatively precise to exact manner. Aligning the image allows for easy comparison of the stored input minutiae location against the template data [3].

The representation is minutiae-based and each minutiae is described by its location (x, y) coordinates. In this case, the fingerprint verification problem may be reduced to a point pattern matching (minutiae pattern matching) problem. In the ideal case, if the correspondence between the template minutiae pattern and input minutiae pattern is known, there are no deformations such as translations, rotations and others, and each minutiae present in a fingerprint image is exactly localized, then fingerprint verification is only a trivial task of counting the number of spatially matching pairs between the two images. Determining whether two representations of a finger extracted from two impressions of it's possibly separated by a long duration of time, are indeed representing the same finger is an extremely difficult problem.

Figure 1.6 Two different fingerprints impressions of the same finger [7].

(Figure 1.6) illustrates the difficulty with an example of two images of the same finger. In particular,

a) The finger may be placed at different locations on the glass platen resulting in a translation of the minutiae from the test representation with respect to those in the reference representation.
b) The finger may be placed with different orientations on the glass platen resulting in a rotation of the minutiae from the test representation with respect to those of the reference representation.
c) The finger may exert a different downward normal pressure on the glass platen resulting in a spatial scaling of the minutiae from the test representation with respect to those in the reference representation.
d) Spurious minutiae may be present in both the reference as well as the test representations. (Manual work, accidents etc. inflict injuries to the finger, thereby changing the ridge structure of the finger, either permanently or semi-permanently).
e) Genuine minutiae may be absent in the reference or test representations. (Skin disease, sweat, dirt, humidity in the air all confound the situation resulting in a non-ideal contact situation).

A matcher may rely on one or more of the above, resulting in a wide spectrum of behavior. At the one end of the spectrum, we have the Euclidean matcher, who allows only rigid transformations among the test and reference representations. At the other extreme, we have the topological matcher, who may allow the most general transformations [4].

The purpose of fingerprint image processing is to extract a condensed representation of the image. This representation (referred to as a template) is used for fingerprint matching. From the ridge flow pattern is extracted the minutiae detail that makes a fingerprint different from other prints. A first part of the detail that is usually used in fingerprint representations is the set of endings and ridge bifurcations in the flow pattern.

The convention (corresponding to the reality of inked cards and FTIR (Frustrated Total Internal Reflection) images) that ridges appear black and valleys between ridges are white, so B represents a ridge ending and A represents a bifurcation. We further follow the intuitive convention that ridges have high values and valleys have low values, although this leads us to have white pixels having low values and black pixels having high values.

Note that the minutiae in (Figure 1.7) are of exceptional quality. In many cases, the presence of minutiae is much less clear-cut. Often, for example, it is difficult to distinguish between a ridge ending and a ridge bifurcation since differences in pressure while acquiring the fingerprint image can join a ridge ending to an adjacent ridge, producing a bifurcation and vice versa.

The minutiae extraction process typically consists of ridge extraction, followed by ridge thinning and minutiae extraction. Ridge extraction, or ridge segmentation, is essentially the step of binarizing the fingerprint image. That is, somehow the fingerprint image I: (x, y) $\rightarrow$ [0, 255] is converted to B: (x, y) $\rightarrow$ [0, 1] , where the value 0 corresponds to valleys and 1 to ridges. One way of accomplishing this is to select a global threshold T and converting the image I (x, y) to a binary image as

$$B(x, y) = \begin{cases} 1, & I(x, y) \geq T \\ 0, & I(x, y) < T \end{cases}$$

Due to the poor quality of many fingerprint images, this approach is most often inadequate for extracting minutiae. In areas of the finger that are dry, no ridges may be detected, while in areas where the finger is wet, no valleys may be detected. The typical solution is to use a threshold T(x, y), which is a function of the spatial location.

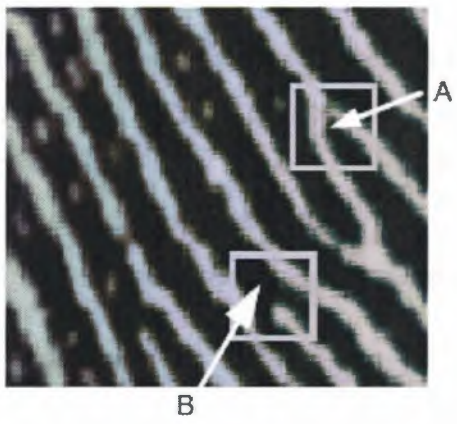
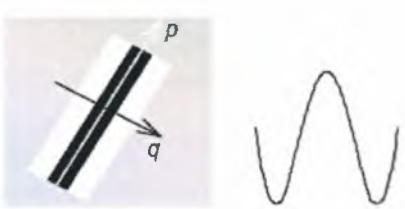


Figure 1.7 Ridge ending (B) and ridge bifurcation (A) [5].

Virtually every published method of feature extraction computes the orientation field of the fingerprint image, which reflects the local ridge direction at every pixel. (Figure 1.8) gives a simple description of this technique as it is applied in.

The local direction p of the ridges is determined by computing gradients in small blocks and averaging these in larger image blocks. Now, consider an image block around a pixel and its projection parallel to the gradient direction onto the q axis in (Figure 1.8). The projected profile is then of the form shown in (Figure 1.8 (b)). The pixel along line q that has maximum intensity value and a few pixels on either side are set to '1' (white) the remaining pixels are set to '0' (black).

At this point, a binary image has been computed which moreorless faithfully represents the original image. The ridges will have width that will vary over the fingerprint images. The next processing steps are typically composed of a sequence of image operations: Directional smoothing, Thinning, Morphological filtering and Minutiae pruning (post-processing). These types of operations may be performed in different order [5].



a b

Figure 1.8 Local thresholding based on expected image function [5].

1.5. Summary

An introduction to fingerprints and their characteristics was presented in this chapter. Common techniques of fingerprint detection were also described. These include minutiae-based, correlation-based and ridge feature-based advantages and disadvantages of these techniques were also presented.

The next chapter will provide more details on fingerprint recognition techniques using minutiae-based method.

CHAPTER TWO
MINUTIAE-BASED FINGERPRINT DETECTION

2.1. Overview

This chapter describes a minutiae-based method to detect fingerprints. The minutiae-based method technique depends mainly on minutiae structures, so each minutiae have different characteristics and shape. Most common fingerprint detection techniques depend on ridge ending and ridge bifurcation to catch minutiae.

2.2. Obstacles in Fingerprint Matching

Fingerprint matching using the minutiae is really a simple problem of determining whether the minutiae on two prints match. However, there are several issues that add to the problem's complexity. First of all, there is the unknown rotation, scale and translation of the prints that may cause the minutiae not to line up [8]. Because people are not likely to place their finger onto the scanner in exactly the same position every time, the prints and thus the minutiae will never line up exactly [9]. There is also the problem of plastic distortion that arises because of the pressure that is put on the finger when it is pressed against a surface. The distortion is not equal every time a fingerprint is taken, therefore this too can cause differences between the two prints from the same finger [11]. Because of these problems, preprocessing must be done on the prints in order to eliminate some of the differences between the two fingerprints [13].

The quality of the prints can also affect the matching algorithm's performance. When using a scanner, poor quality prints can be produced if person's hands are wet, dirty or too dry. These conditions can cause a scanner to pick up false minutiae and not pick up some true minutiae [12]. Either case would adversely affect the matching algorithm. Scars or cuts across a finger can lead to a host of ridge endings that were not in both the print that was stored and the one being matched against [8]. Different scans of the same fingerprint will not have the same minutiae on them [10].

When using ink and paper to initially capture the prints, the problems mentioned above still apply. There are also the additional problems of over inking and smudges in the ink. Poor quality prints are also a possible when one is working with latent prints from a crime scene or from prints used at police department, where the people submitting the prints are less then willing. Because getting identically matching images

of the same fingerprint is impossible [11], a matching algorithm cannot just match each minutiae and declare that there is a match only when all of them can be lined up.

There is no upon standard that defines matching [9]. In the Netherlands 12 minutiae must match for fingerprints to legally be declared as matching. Seven are needed in South Africa and the United States leaves it up to experts in individual cases. The fingerprint matching program's strictness is determined by the program's application. In government security systems the program is likely to be stricter than those used for access to one's home desktop [13].

2.3. Fingerprint Minutiae Extraction

2.3.1. Introduction of minutiae extraction

The smaller minutiae are particulars of a person's ridges and valleys. Features like bifurcation ridge ends, islands, lakes, and dots are formed by the placement of the ridges and valley along the finger [13]. There are about 60 to 80 minutiae can be found in a good print [15]. The minutiae can be used to match fingerprints as everyone has a unique arrangement of these features. Usually only bifurcation and ends are used in automated fingerprint matching systems [16].

2.3.2. Fingerprints feature

The human fingerprint is comprised of various types of ridge patterns, traditionally classified according to the decades-old Henry system: left loop, right loop, arch, whorl, and tented arch. Loops make up nearly 2/3 of all fingerprints, whorls are nearly 1/3, and perhaps 5-10% is arches. These classifications are relevant in many large scale forensic applications, but are rarely used in biometric authentication.

Minutiae, the discontinuities that interrupt the otherwise smooth flow of ridges, are the basis for most fingerprint authentication. Codified in the late 1800's as Galton features, minutiae are at their most rudimentary ridge endings, the points at which a ridge stops, and bifurcations, the point at which one ridge divides into two. Many types of minutiae exist, including dots (very small ridges), islands (ridges slightly longer than dots, occupying a middle space between two temporarily divergent ridges), ponds or lakes

Figure 2.1 Fingerprint structure [17].

Empty spaces between two bridges (small ridges joining two longer adjacent ridges), and crossovers (two ridges which cross each other).

Other features are essential to fingerprint authentication. The core is the inner point, normally in the middle of the print, around which swirls, loops, or arches center. It is frequently characterized by a ridge ending and several acutely curved ridges. Deltas are the points, normally at the lower left and right hand of the fingerprint, around which a triangular series of ridges center [17].

2.4. Feature Extraction

Minutiae extraction refers to the process by which the minutiae points are detected in a fingerprint image. Each minutiae is characterized by its (x; y) location in the image, and the orientation of the ridge on which it is detected. The ridge information in a 64×64 region around the (x; y) point is associated with every minutiae which is useful when two minutiae sets are being matched. The minutiae extraction scheme (Figure 2.2) can be broadly classified into the following stages:

1. Orientation field estimation: The orientation of the fingerprint image is computed in non-overlapping blocks by examining the gradients of pixel intensities in the x and y directions within the block.
2. Ridge detection: The ridges present in the fingerprint image are identified by applying masks that are capable of accentuating the local maximum gray level values along the normal direction of the local ridge direction.

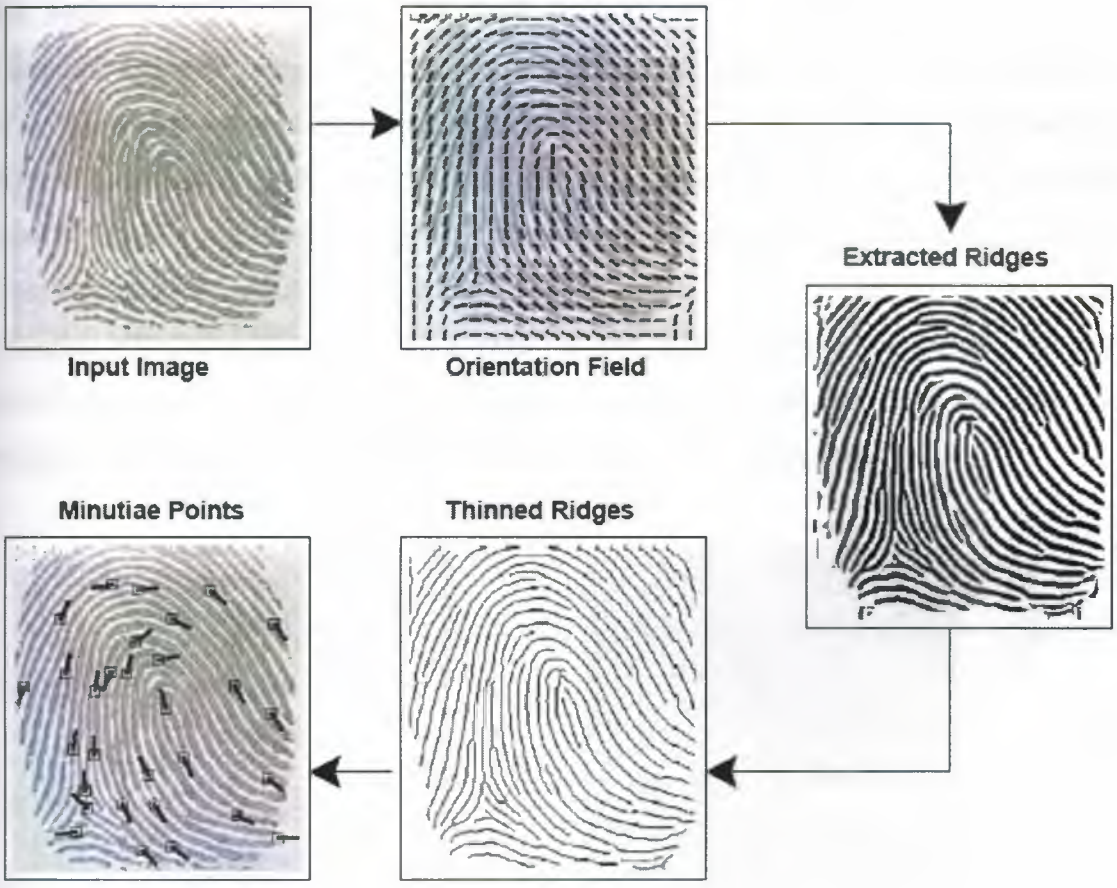


Figure 2.2 Flowchart of the minutiae extraction [18].

3. Ridge thinning: The ridge map constructed in the earlier stage is used to obtain a thinned ridge image.
4. Minutiae detection: A set of rules is applied to the thinned ridges to label minutiae points (ridge endings and ridge bifurcations). As a post processing step, a refinement algorithm is applied to remove spurious minutiae points.

Minutiae matching involve a point matching operation on the two minutiae sets. An elastic string matching technique is employed to compare the two minutiae sets.

The output of the matching process is a matching score that indicates the similarity of the two sets being compared, and a correspondence map that indicates pairing of minutiae points from the two sets. The correspondence map is used to compute the transformation parameters necessary to align the two fingerprint images [18].

2.5. Minutiae Classification

The most minutiae categories are ridge ending and bifurcation (See Figure 2.3). Several of the fingerprint matching algorithms reported in the literature do not use minutiae type information because of the difficulty in designing a robust classifier to identify minutiae type. We use a rule-based minutiae classification scheme and show that the resulting classification of minutiae can indeed improve the overall matching accuracy. In minutiae extraction algorithm, if a pixel in the thinned image has more than two neighbors, then the minutiae is classified as a bifurcation, and if a pixel has only one neighbor, then the minutiae is classified as an ending (See figure 2.4).

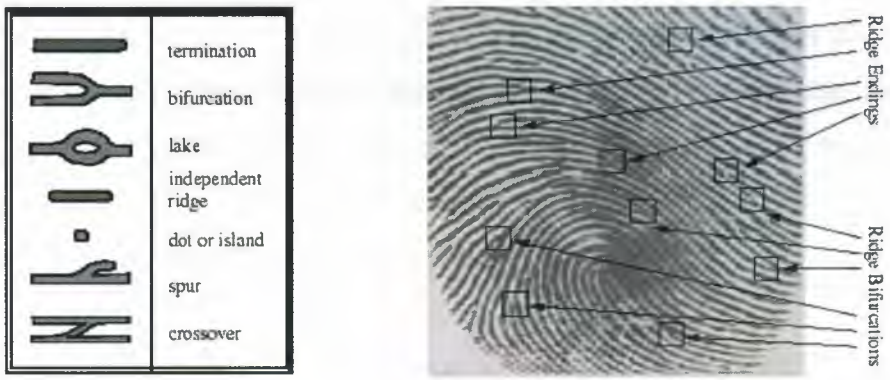


Figure 2.3 (a) Different minutiae types, (b) Ridge ending & Bifurcation [20].

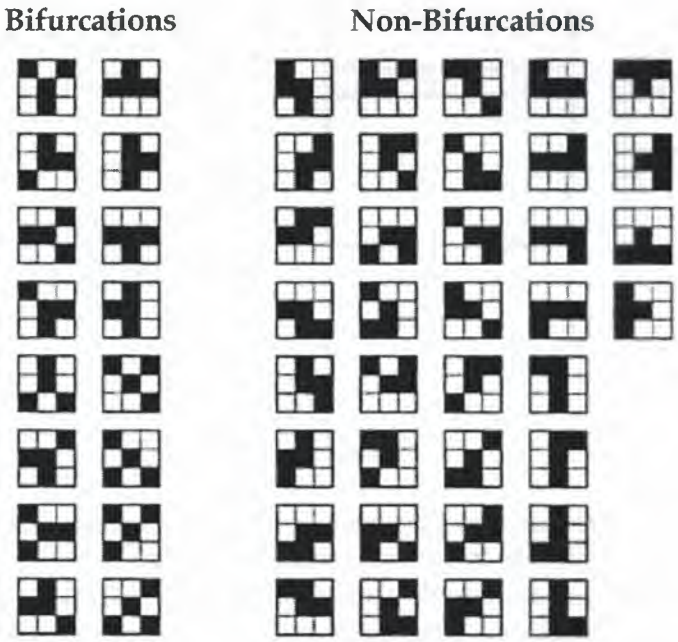


Figure 2.4 Set of 8 neighbors [21].

Note classification of minutiae does not affect any other part of the feature extraction stage. In our experience, there is significantly more number of minutiae endings present in a typical fingerprint than bifurcations; the probability of occurrence of a ridge ending is greater than the probability of occurrence of a ridge bifurcation [19].

2.6. Fingerprints Matching Using Chain Coded

2.6.1 Introduction in Chain Coded Method

The chain code representation is procedurally described as follows. Given a binary image, it is scanned from top to bottom and right to left, and transitions from white (background) to black (foreground) are detected. The contour is then traced counterclockwise (clockwise for interior contours) and expressed as an array of contour elements (Figure 2.5(a)). Each contour element represents a pixel on the contour, contains fields for the x, y coordinates of the pixel, the slope or direction of the contour into the pixel, and auxiliary information such as curvature. The slope convention used by the algorithms described is as shown in (Figure 2.5(b)).

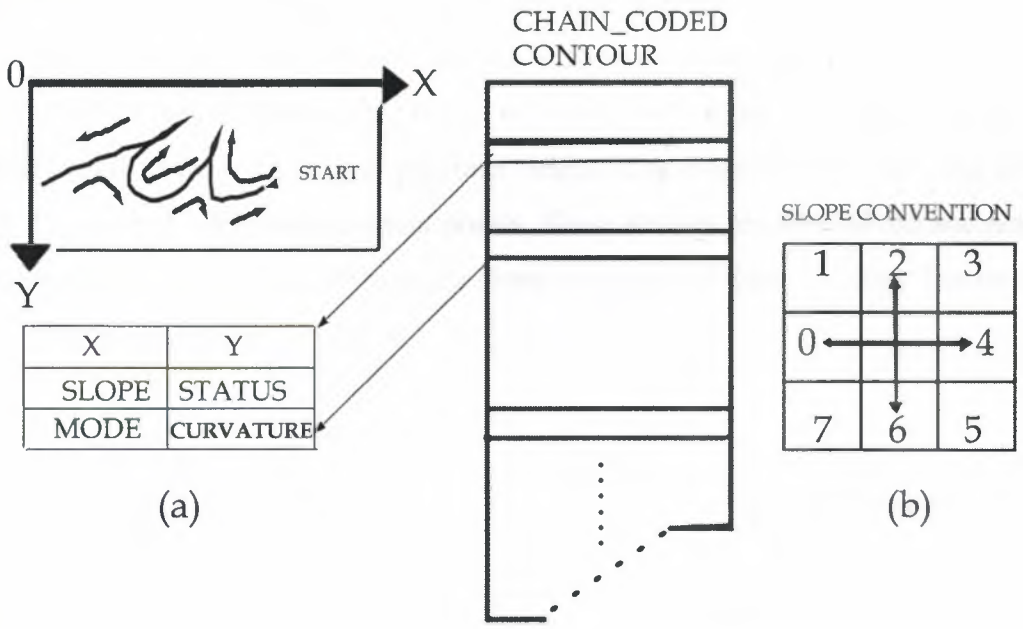


Figure 2.5 Chain code contour representations: (a) contour element, (b) slope convention. Data field in the array contains positional and slope information of each component of the traced contour. Properties stored in the information fields are: coordinates of bounding box of a contour, number of components in the corresponding data fields, area of the closed contour, and a flag which indicates whether the contour is interior or exterior[22].

2.6.2. Minutiae Extraction Using Chain Code

Most of the fingerprint minutiae extraction methods are thinning-based by which the skeletonization process converts each ridge contour to one pixel wide. The minutiae points are detected by tracing the thin ridge contours. When the trace stops, an end point is marked. Bifurcation points are those with more than two neighbors. In practice, thinning methods have been found to be sensitive to noise and the skeleton structure does not match up with the intuitive expectation.

The alternate method of using chain coded contours is presented here. The direction field estimated from chain code gives the orientation of the ridges and information on any structural imperfections such as breaks in ridges, spurious ridges and holes. The standard deviation of the orientation distribution in a block is used to determine the quality of the ridges in that block.

We have used contour tracing in other handwriting recognition applications. We consistently trace the ridge contours of the fingerprint images in a counter-clock-wise fashion. When we arrive at a point where we have to make a sharp left turn we mark a candidate for a ridge ending point. Similarly when we arrive at a sharp right turn, the turning location marks a bifurcation point (Figure 2.6 (a)).

To determine the significant left and right turning contour points from among the candidates marked during the trace, we compute vectors P_{in} leading in to the candidate point P from its several previous neighboring contour points and P_{out} going out of P to several subsequent contour points. These vectors are normalized and placed in a Cartesian coordinate system with P_{in} along the x-axis (Figure 2.6 (b)). The turning direction is determined by the sign of

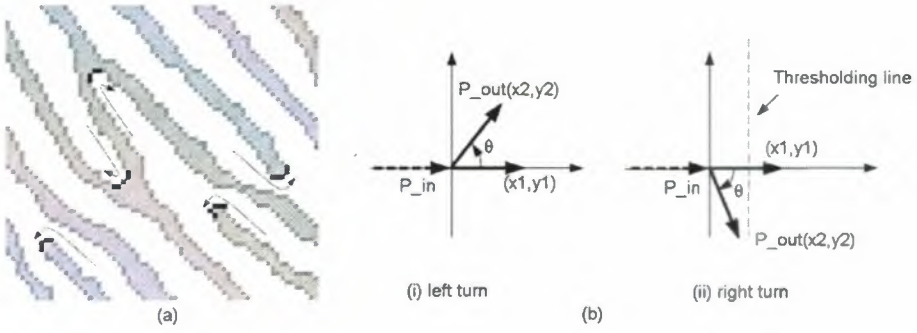


Figure 2.6 (a) Minutiae location in chain code contours, (b) the distance between the thresholding line and the y-axis gives a threshold for determining a significant turn [22].

$$S(P_{in}; P_{out}) = X_1 Y_2 - X_2 Y_1$$

$S(P_{in}; P_{out}) > 0$ indicates a left turn and $S(P_{in}; P_{out}) < 0$ indicates a right turn. A threshold T is then selected such that any significant turn satisfies the conditions:

$$X_1 Y_1 + X_2 Y_2 < T$$

Since the threshold T is the x-coordinate of the thresholding line in Figure 2.6(b), it can be empirically determined to be a number close to zero. This ensures that the angle made by P_{in} and P_{out} is close to or less than 90^0.

The turning point locations are typically made of several contour points. We define the location of minutiae as the center point of the small group of turning pixels. The minutiae density per unit area is not allowed to exceed a certain value. If we consider groups of candidate minutiae forming clusters, all the candidate minutiae in a cluster whose density exceeds this value are replaced by a single minutiae point located at the center of the cluster.

A feature extraction method using the Chain code representation of fingerprint ridge contours is presented for use by Automatic Fingerprint Identification Systems (AFIS). The representation allows efficient image quality enhancement and detection of fine feature points called minutiae. The low quality fingerprint image detection is a disadvantage of this method [22].

2.6. Summary

The minutiae-based method is the most common in fingerprints detections technique because it depends on minutiae; fingerprints detection methods have a lot of steps; these were presented within this chapter.

Next chapter introduces the used technique and how to solve most of fingerprints detection technique problems.

CHAPTER THREE
IMPLEMENTATION OF FINGERPRINT DETECTION ALGORITHM

3.1 Overview

In this chapter the fingerprint detection technique will be described. Types of minutiae and their extraction will be explained. Steps and algorithms for minutiae extraction will be presented.

3.2 Introduction

Fingerprint matching algorithms can be grouped into two categories: core-based match algorithms and structure based match algorithms. Most of the structure based matching algorithms are time consuming, therefore they are not suitable for real application, core-based matching algorithm is more efficient than the structure-based matching algorithm, but it highly depends on core point detection.

Most automatic fingerprint identification systems (AFIS) are based on local ridge features, such as ridge endings and ridge bifurcations.

The used algorithm is based on structure-based matching algorithms, It will be able to solve problems like translation, rotation and does not depend on core point. The developed technique has some advantages in processing speed, rotation angle and noise tolerance.

3.3 Type of minutiae

In this technique two types of minutiae are used; first is the ridge ending, second is the ridge bifurcation. This technique uses 8-nighbors to catch minutiae ,it made by using image pixels if a pixel in the thinned image has more than two neighbors, then the minutiae is classified as a bifurcation, and if a pixel has only one neighbor, then the minutiae is classified as an ending as following:

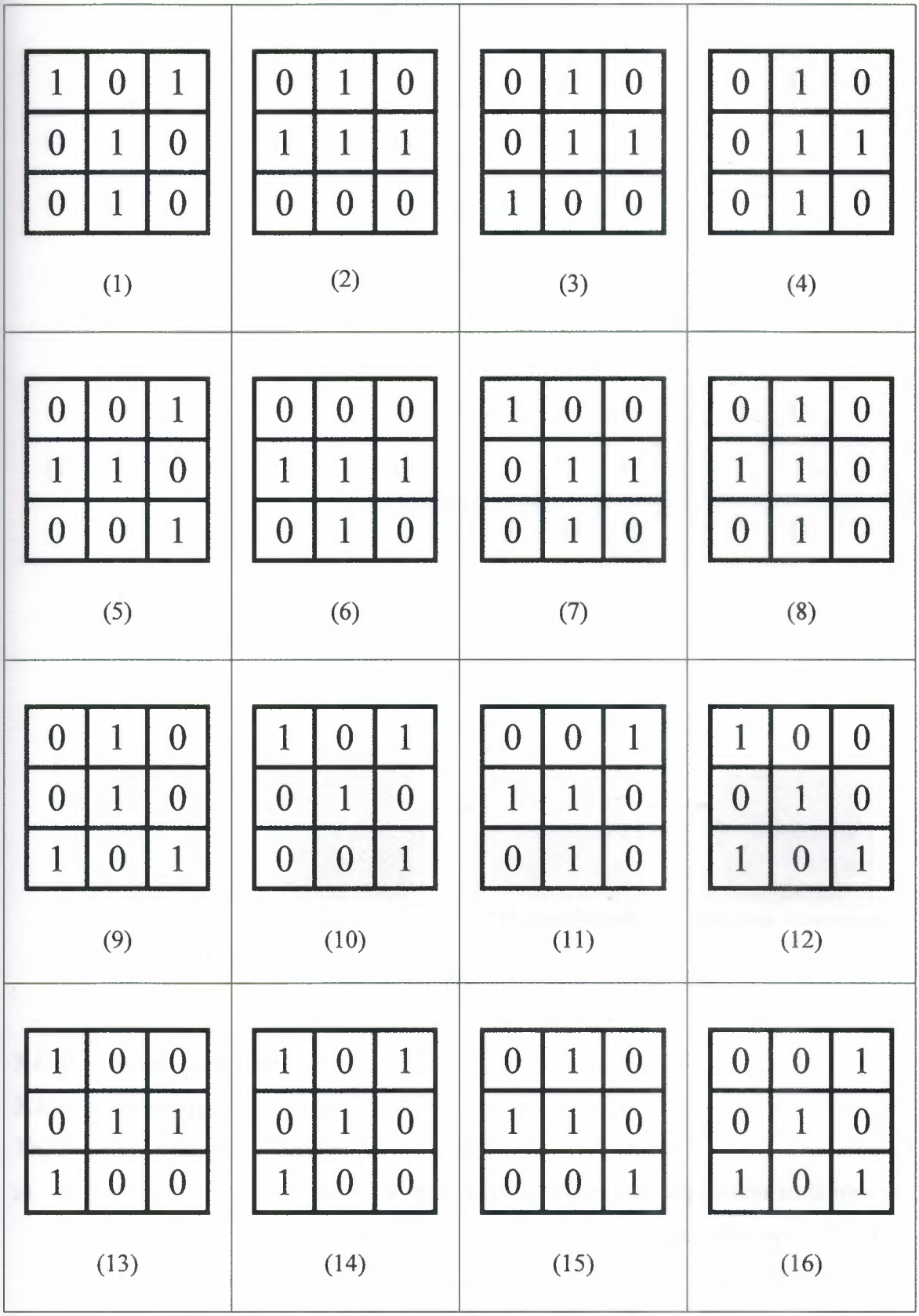


Figure 3.1 Set of ridge bifurcation 8-neighbors.

1	0	0	0	1	0	0	0	1	0	0	0
0	1	0	0	1	0	0	1	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0
(1)			(2)			(3)			(4)		
0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0	0	0	0	0
(5)			(6)			(7)			(8)		

Figure 3.2 Set of ridge ending 8-neighbors.

Figure 3.3 Block diagram of minutiae extraction.

3.4. Developed Algorithm

3.4.1. Scan fingerprints image

The fingerprint acquisition technique is divided in two techniques:

a) Manual technique, the oldest and most known fingerprint acquisition technique is the "ink technique", that is, pressing the finger against a card after spreading the finger skin with ink; this technique is nowadays still largely used by the police in AFIS. The cards are converted into digital form by means of scanners identical to those normally employed for general purpose paper documents. The default resolution is 300 dpi. This technique can produce images including regions

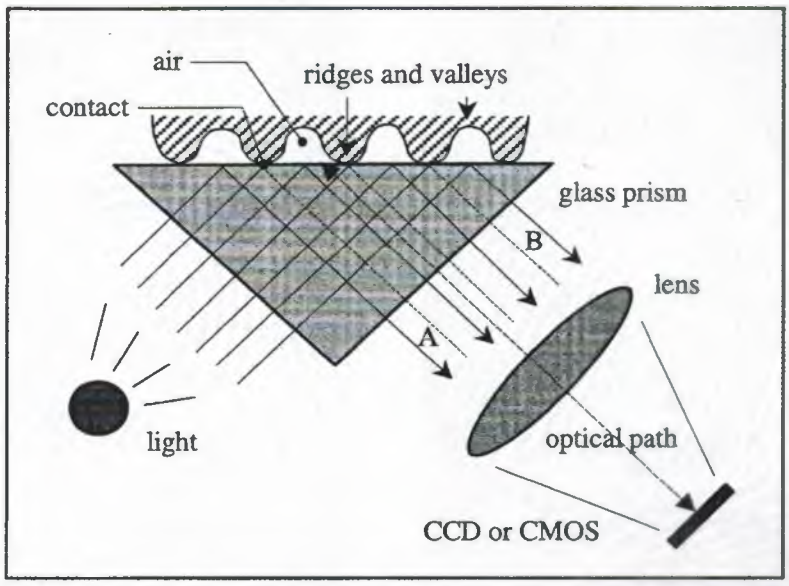


Figure 3.4 Frustrated Total Internal Reflections (FTIR) based fingerprint sensor [23]

Which miss some information, due to excessive inkiness or to ink deficiency, and is obviously limited to forensic applications.

b) The Frustrated Total Internal Reflection (FTIR) is the most used and mature live-scan sensing technique. The finger is illuminated from one side of a glass prism with a LED, while the other side transmits the image through a lens to a CDD/CMOS sensing element which converts light into digital information. The lack of reflection caused by the presence of water particles where the ridges touch the prism allows ridges to be discriminated from valleys.

3.4.2. Fingerprints Binarization

Fingerprint Image Binarization is to transform the 8-bit Gray fingerprint image to a 1-bit image with 0-value for ridges and 1-value for furrows. After the operation, ridges in the fingerprint are highlighted with black color while furrows are white.

A locally adaptive binarization method is performed to binarize the fingerprint image. Such a named method comes from the mechanism of transforming a pixel value to 1 if the value is larger than the mean intensity value (Figure 3.5).

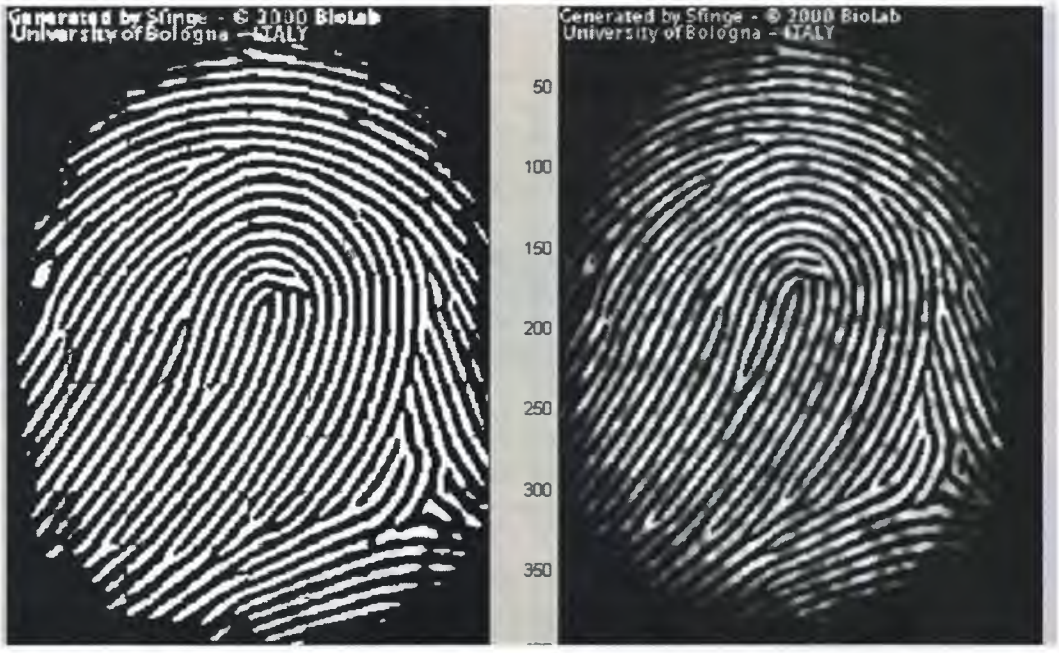


Figure 3.5 The fingerprint image after adaptive binarization
Binarized image (left), Enhanced gray image (right). [23]

3.4.3. Ridge Thinning

The thinning method is adapted from morphological method. We thin the ridge from the contour points, pixels with value 1 having at least one ridge pixel in the 8-neighborhood as depicted below

.

P9	P2	P3
P8	P1	p4
P7	P6	P5

The Various morphological operations are performed on binarized image. The most important of them include thinning and spur removal. A thinned fingerprint image is one in which each ridge is one pixel in width. Thinning is achieved by successive deletion of pixels from all sides of the image. Each of the four sides are eroded away (Figure 3.6) according to some set template.

If the image matches with template, the middle pixel is removed. Initially the two north images are processed, and then the other compass points are overlaid. Once all eight matrices have been sampled on the entire image, the process is repeated again on the newly formed image.

North	South	East	West
0 0 0	1 1 X	0 X 1	X X 0
X 1 X	X 1 X	0 1 1	1 1 0
X 1 1	0 0 0	0 X X	1 X 0
X 0 0	X 1 X	0 0 X	X 1 X
1 1 0	0 1 1	0 1 1	1 1 0
X 1 X	0 0 X	X 1 X	X 0 0

Figure 3.6 Eight matrices based thinning method.

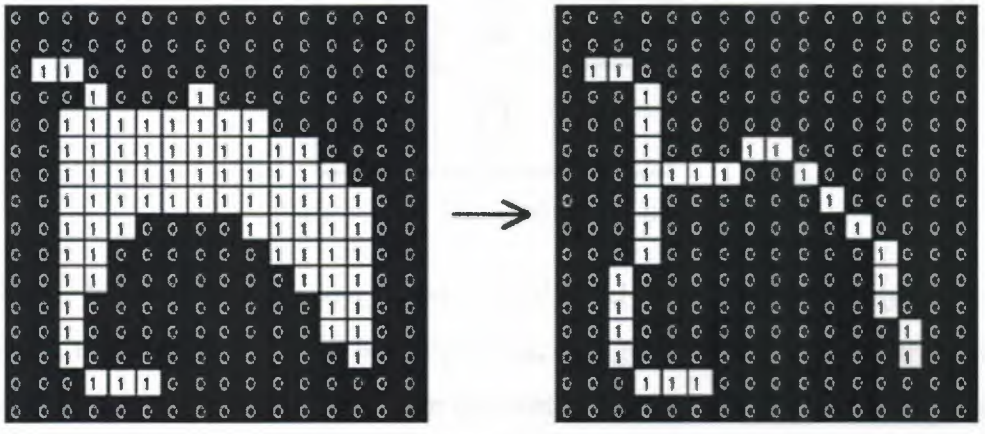


Figure 3.7 Thinning ridge.

Processing only stops when no more pixels can be deleted. Similarly spur operation is performed on thinned image to remove false bifurcations and ends caused by thinning. This removes some real end points from their original locations too. (Block scheme of thinning algorithm is given in appendix 1)

3.4.4. Minutiae Extraction

3.4.4.1 Minutiae marking

After the fingerprint ridge thinning, marking minutiae points is relatively easy. But it is still not a trivial task as most literatures declared because at least one special case evokes my caution during the minutiae marking stage.

0	1	0
0	1	0
1	0	1

Figure 3.8.1 Bifurcation

0	0	0
0	1	0
0	0	1

Figure 3.8.2 Termination

0	1	0
0	1	1
1	0	0

Figure 3.8.3 Triple counting branch

In general, for each 3x3 window, if the central pixel is 1 and has exactly 3 one-value neighbors, then the central pixel is a ridge branch (Figure 3.8.1). If the central pixel is 1 and has only 1 one-value neighbor, then the central pixel is a ridge ending (Figure3.8.2).

Figure 3.8.3 illustrates a special case that a genuine branch is triple counted. Suppose both the uppermost pixel with value 1 and the rightmost pixel with value 1 have another neighbor outside the 3x3 window, so the two pixels will be marked as branches too. But actually only one branch is located in the small region. So a check routine requiring that none of the neighbors of a branch are branches is added.

Also the average inter-ridge width D is estimated at this stage. The average inter-ridge width refers to the average distance between two neighboring ridges. The way to approximate the D value is simple. Scan a row of the thinned ridge image and sum up all pixels in the row whose value is one. Then divide the row length with the above summation to get an inter-ridge width. For more accuracy, such kind of row scan is performed upon several other rows and column scans are also conducted, finally all the inter-ridge widths are averaged to get the D.

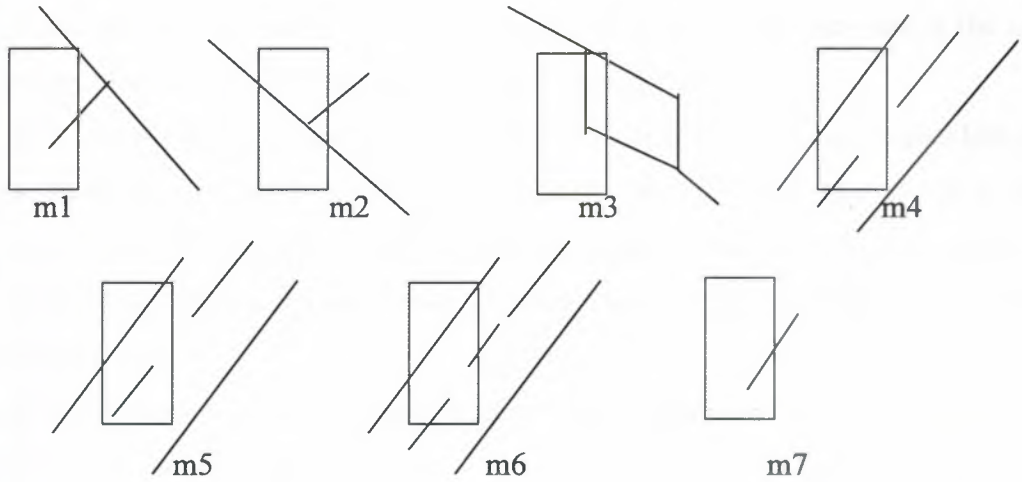


Figure 3.9 False minutiae structures. m1 is a spike piercing into a valley. In the m2 case a spike falsely connects two ridges. m3 has two near bifurcations located in the same ridge. The two ridge broken points in the m4 case have nearly the same orientation and a short distance. m5 is alike the m4 case with the exception that one part of the broken ridge is so short that another termination is generated. m6 extends the m4 case but with the extra property that a third ridge is found in the middle of the two parts of the broken ridge. m7 has only one short ridge found in the threshold window.[25].

3.4.4.2 False minutiae removal

The preprocessing stage does not totally heal the fingerprint image. For example, false ridge breaks due to insufficient amount of ink and ridge cross-connections due to over inking are not totally eliminated. Actually all the earlier stages themselves occasionally introduce some artifacts which later lead to spurious minutiae. These false minutiae will significantly affect the accuracy of matching if they are simply regarded as genuine minutiae. So some mechanisms of removing false minutiae are essential to keep the fingerprint verification system effective.

Only handles the case m1, m4, m5 and m6 have not false minutiae removal by simply assuming the image quality is fairly good. Has not a systematic healing method to remove those spurious minutiae although it lists all types of false minutiae shown in (Figure 3.9) except the m3 case.

The procedures in removing false minutiae are:

1. If the distance between one bifurcation and one termination is less than D and the two minutiae are in the same ridge (m1 case). Remove both of them. Where D is the average inter-ridge width representing the average distance between two parallel neighboring ridges.

2. If the distance between two bifurcations is less than D and they are in the same ridge, remove the two bifurcations. (m2, m3 cases).
3. If two terminations are within a distance D and their directions are coincident with a small angle variation. And they suffice the condition that no any other termination is located between the two terminations. Then the two terminations are regarded as false minutiae derived from a broken ridge and are removed. (case m4,m5, m6).
4. If two terminations are located in a short ridge with length less than D, remove the two terminations (m7).

The procedures in removing false minutiae have two advantages. One is that the ridge ID is used to distinguish minutiae and the seven types of false minutiae are strictly defined comparing with those loosely defined by other methods. The second advantage is that the order of removal procedures is well considered to reduce the computation complexity. It surpasses the way adopted by that does not utilize the relations among the false minutiae types.

3.4.4.3 Minutiae matching

The algorithm aim to match between two fingerprint by uses short time and more efficient, this algorithm solve every old algorithm problems also we can determine any fingerprint by table of number only without keep original fingerprint image because every fingerprint structure has saved as set of numbers (distances and angles). This algorithm depends on the distance between every sequence point (minutiae) and angle between them (Figure 3.10).

1. Take the horizontal line and check to find the first point $P_1(x_1,y_1)$ in fingerprint image after that we check the next point P_2 (x_2,y_2),where x_1, x_2 is X-axis coordinate and y_1, y_2 is y-axis coordinate.

   Calculate the distance between every sequence two points by:

   a. Calculate the horizontal distance X axis by using this equation

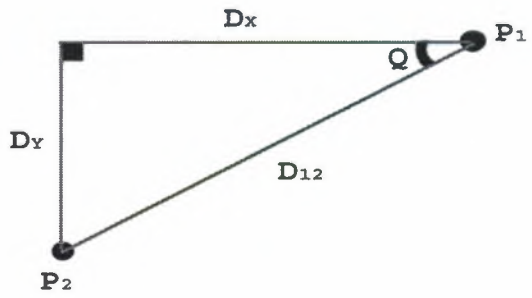


Figure 3.10 Triangle shape.

$$Dx = | X1 - X2 | \qquad 3.1$$

b. Calculate the vertical distance Y axis by using this equation

$$Dy = | Y1 - Y2 | \qquad 3.2$$

c. By using Pythagorean theorem

$$Dxy = \sqrt{Dx^2 + Dy^2} \qquad 3.3$$

Where Dxy is distance between two sequence points

d. By using triangular angle calculation we calculate the angle Q

$$\frac{Sin90^0}{Dxy} = \frac{SinQ}{Dy} \quad \text{, but sin } 90^\circ = 1 \qquad 3.4$$

$$\frac{1}{Dxy} = \frac{SinQ}{Dy} \qquad 3.5$$

$$SinQ = \frac{Dy}{Dxy} \qquad 3.6$$

$$Q = Sin^{-1}\left(\frac{Dy}{Dxy}\right) \qquad 3.7$$

2. Create array to store every point's records, the record have the distance between two points and the angle (See Table 3.1).
3. Compare the distance between two fingerprints sequence records tables, the technique match fingerprints when its find five sequence point the distance in two table has same but if the angles are different, calculate fingerprint rotate angle for every point because every point have a same rotate angle.

Table 3.1 Array of point's record

Index	Distance (pixels)	Angle (Q)
1	20	50
2	60	30
3	70	75
.	.	.
.	.	.
.	.	.

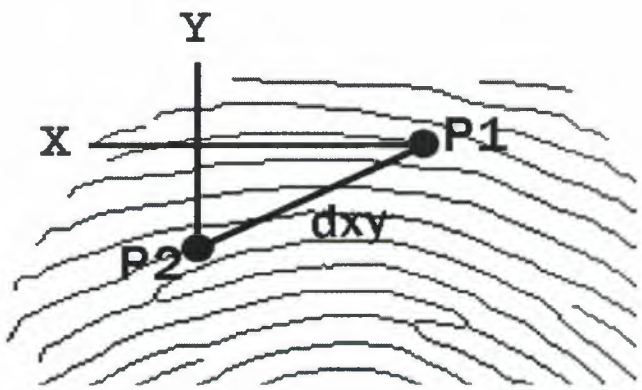


Figure 3.11 Sequence minutiae

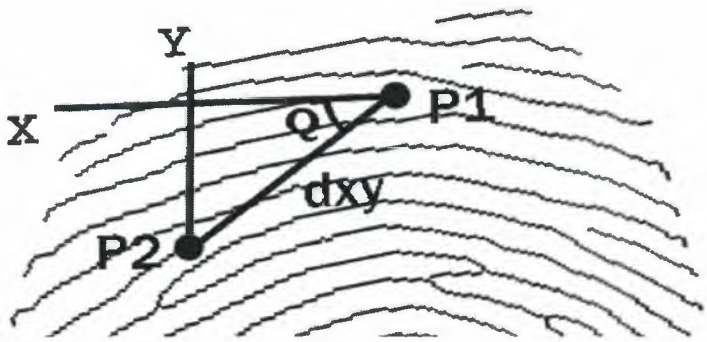


Figure 3.12 Rotate fingerprint image

The matching algorithm verifies any rotated fingerprint. The distance between sequential points are very important to matching fingerprints, so the algorithm uses rotation angles to verify matching by using rotation angles between every sequential points (Figure 3.11).

The aim of Algorithm is to keep every information about any fingerprint inside the array. Using array for matching is easier and faster than using bitmap images. The array contains two important factors, first the distance between sequential points, second is the angle between sequential points on the X-axis.

The distance between adjacent points does not change when the fingerprint is rotated. However the angle between adjacent points could change if the orientation of the fingerprint changes (Figure 3.12).

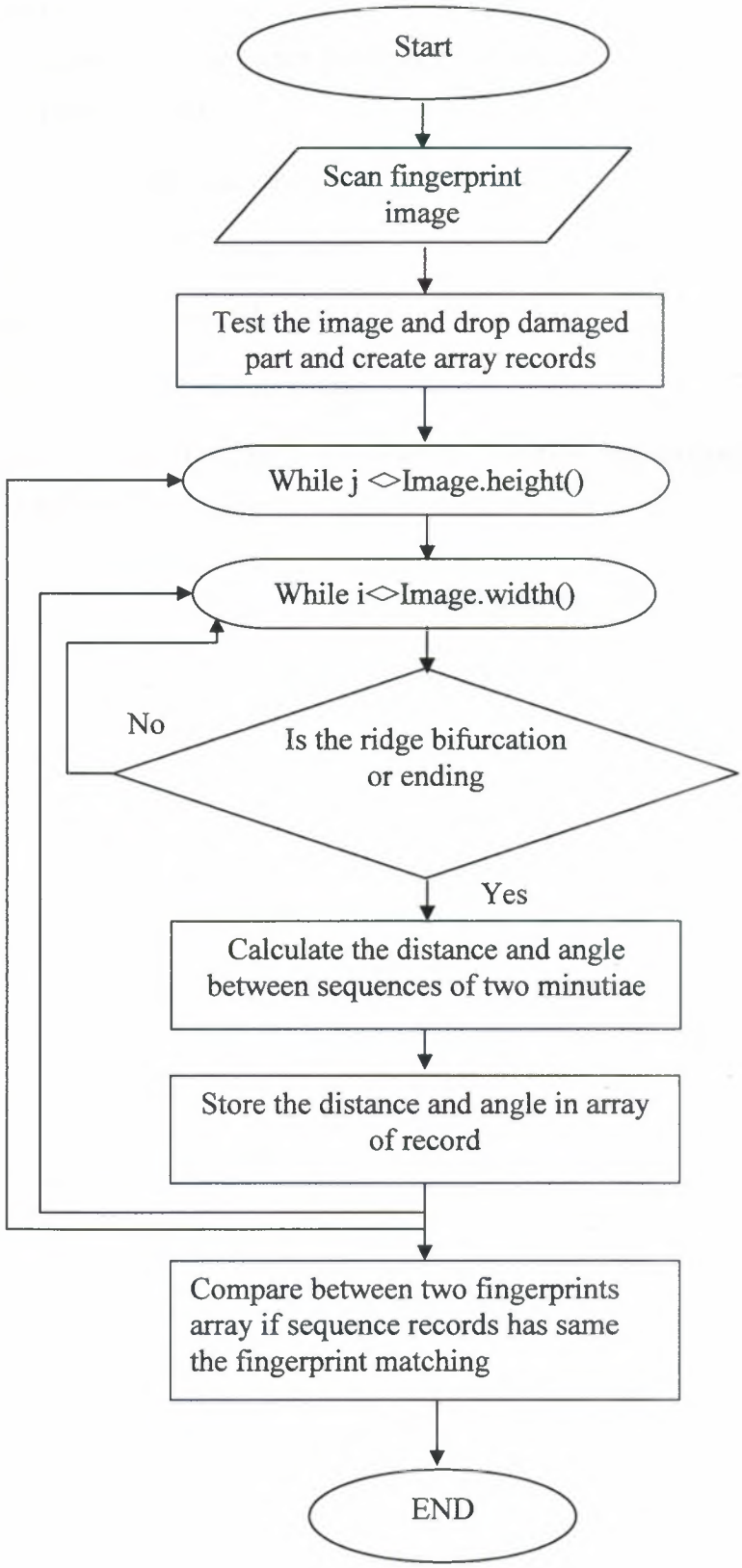


Figure 3.13 Developed algorithm flowcharts.

3.5. Summary

This chapter presents the algorithm for fingerprint detection techniques. This algorithm matches fingerprints as follows:

1. Fingerprint image binarization.
2. Fingerprint image enhancement.
3. Fingerprint thinning ridges.
4. Fingerprint minutiae extraction.

The aim of this technique is to solve the rotation and orientation problems in fingerprint detection techniques.

CHAPTER FOUR
REAL-LIFE IMPLEMENTATION

4.1 Overview

This chapter describes the real-life application of fingerprint detection technique.

This real-life application will be based on using fingerprints of five persons to verify the effectiveness of the developed technique.

4.2 The Fingerprints

This section present the application of the developed technique using fingerprint of different persons; there persons have different fingerprints type to extract the efficiency of new technique by try to enter all fingerprints image and take the processing time and efficiency. Each person record has the name of person, age, the way to get fingerprint image, type of fingerprint and fingerprint image.

1. Person 1
    - Name: Mehmet Aliem
    - Age: 26
    - Way: ink and paper
    - Type of fingerprint: Loop

Figure 4.1 Person 1 fingerprint image

Figure 4.2 Person 2 fingerprint image

2. Person 2
    - Name: Yousif Faruok
    - Age: 26
    - Way: ink and paper
    - Type of fingerprint: Whorl

3. Person 3
    - Name: John Adam
    - Age: 24
    - Way: ink and paper
    - Type of fingerprint: Loop

Figure 4.3 Person 3 fingerprint image.

Figure 4.4 Person 4 fingerprint image

4. Person 4
    - Name: Hakki Ahmet
    - Age: 44
    - Way: ink and paper
    - Type of fingerprint: Loop

5. Person 5
    - Name: Sami Hassan
    - Age: 28
    - Way: ink and paper.
    - Type of fingerprint: Loop.

Figure 4.5 Person 5 fingerprint image.

4.3 Fingerprint Detection Using New Algorithm

Each fingerprint is passing in system steps before extract real result the most five steps important in system as following:

- Fingerprint image scanning.
- Fingerprint binarization.
- Fingerprint enhancement.
- Fingerprint ridge thinning.
- Minutiae extraction.

The developed system uses the algorithms shown in chapter three to generate the fingerprint records table.

The minutiae extraction steps for all persons will be shown next in Figure 4.6 – Figure 4.10.

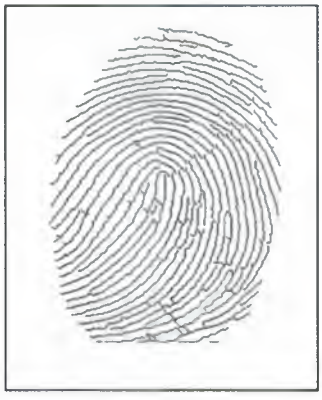
Original fingerprints are obtained via scanner (300dpi).

➢ Person 1

a) Fingerprint scan.

b) Fingerprint binarization.

c) Fingerprint Enhancement



d) Fingerprint ridge thinning

e) Fingerprint minutiae extraction

Figure 4.6 Person 1 minutiae extraction steps.

➢ Person 2

a) Fingerprint scan.

b) Fingerprint binarization.

c) Fingerprint Enhancement

d) Fingerprint ridge thinning

e) Fingerprint minutiae extraction

Figure 4.7 Person 2 minutiae extraction steps.

➢ Person 3

a) Fingerprint scan.

b) Fingerprint binarization.

c) Fingerprint Enhancement

d) Fingerprint ridge thinning

e) Fingerprint minutiae extraction

Figure 4.8 Person 3 minutiae extraction steps.

➢ Person 4

a) Fingerprint scan.

b) Fingerprint binarization.

c) Fingerprint Enhancement

d) Fingerprint ridge thinning

e) Fingerprint minutiae extraction

Figure 4.9 Person 4 minutiae extraction steps.

➢ Person 5

a) Fingerprint scan.

b) Fingerprint binarization.

c) Fingerprint Enhancement

d) Fingerprint ridge thinning

e) Fingerprint minutiae extraction

Figure 4.10 Person 5 minutiae extraction steps.

4.4 Results

4.4.1. Algorithm results

The aim of this technique is verify any two fingerprint; this way it will be by converting each fingerprint image to array of records (distances and angles).

The main steps running on fingerprints before converting are:

1- *Fingerprint image scanning.*

Scan fingerprint image by manual technique (ink and paper) resolution is 300 dpi.

2- *Fingerprint binarization.*

Binarization meaning is convert fingerprint image to binary image (black & white) but depend on density of gray levels.

The main idea is generate factor M to convert color by determined color level if color level bigger than M the color is white else the color is black.

Figure 4.11 Fingerprint image scanning.

Figure 4.12 Fingerprint image binarization.

3- *Fingerprint enhancement*.

The goal of fingerprint enhancement to fill any white color space between black color on ridges before thinning image to add more efficiency for thinning algorithm without change any thing in fingerprint main structure.

Figure 4.13 Fingerprint image enhancements.

4- *Fingerprint image thinning.*

The steps to thinning any fingerprint image has describe in Chapter 3.

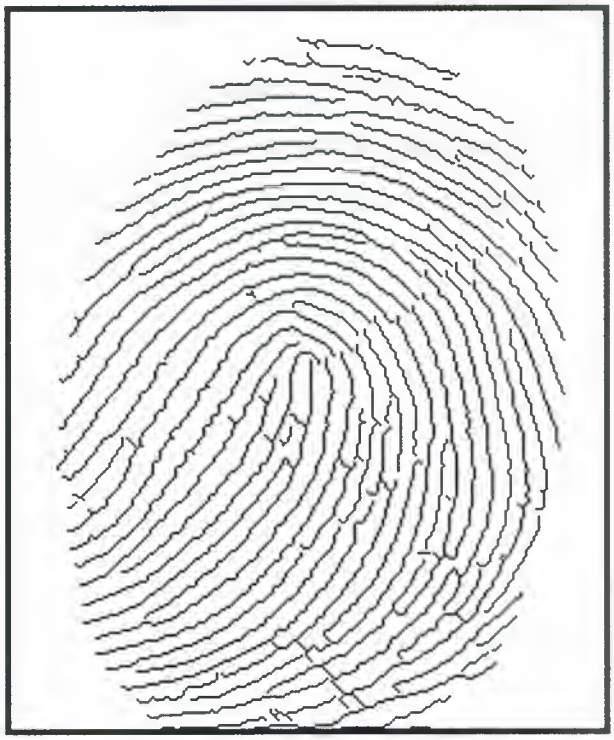


Figure 4.14 Fingerprint image Thinning.

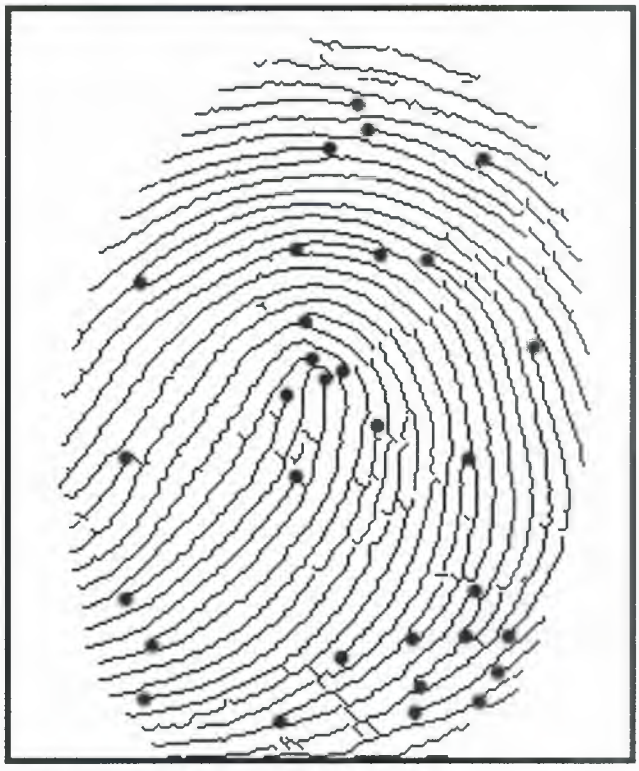


Figure 4.15 Fingerprint image minutiae extraction.

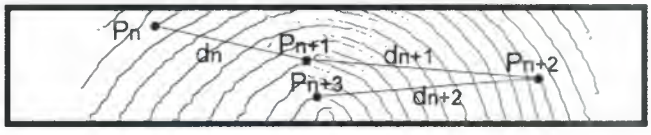


Figure 4.16 Fingerprint image minutiae extraction.

5- *Fingerprint image minutiae extractions.*

The algorithm depend on minutiae to matching fingerprints the (figure 4.16) explain how the algorithm generate array of records for each fingerprints.

Each record contains three fields:

a) Index

b) Distance between two horizontally sequential minutiae.

c) Angle on the X-axis between two horizontally sequential minutiae.

Before creating the table of record, the algorithm calculates for every sequential minutiae the distances and angles.

For example in this case (Figure 4.16) have four minutiae P_n ,P_{n+1} ,P_{n+2} and P_{n+3} The algorithm using equations in Chapter three to calculates distances and angles.

Table 4.1 Table of Minutiae records

Index	**Distance** (inch)	**Angle**
1	–	–
2	–	–
.	–	–
.	–	–
N	0.1936	12.52°
N+1	0.2908	4.34°
N+2	0.2781	5.16°
.	–	–
.	–	–

4.4.2. Efficiency

The efficiency of any fingerprint detection system is calculated by many factors the most important is the results of this system and the kind of fingerprint the system has matching.

Also another important factor the time consumption the system is fast detect or need a lot of time to complete detection process.

The distances and angles between minutiae will add power in detection system because it's impossible to find two sequence minutiae have same distance and angle in any different fingerprints.

4.4.3. Time Cost

The times for each process of developed fingerprint detection system on P4 Celeron processor 2.4 GHZ and 128 MB of RAM included in table 4.2.

Table 4.2 Time consumption for five steps for person 1

Process Name	Time
Scan Fingerprint	-
Fingerprint binarization	2 s
Fingerprint enhancement	2.5 s
Fingerprint ridge thinning	4.5 s
Fingerprint minutiae extraction	3.2 s
Total	**12.2 s**

4.4.4. Analysis

The developed fingerprint detection technique results depend on quality of fingerprint, the method for take fingerprints is very important to get a good results.

Main important point in developed fingerprint detection technique is resolution of fingerprint must be 300 dpi.

Number of minutiae in fingerprint depend on number of bifurcation ridges and end ridges approximately any fingerprint have between 40 to 60 minutiae but the number of minutiae is not important in developed fingerprint matching technique because the result depend on distances and angles.

In developed fingerprint detections system three levels of tolerance

1. Low (L)
2. Mid (M)
3. High (H)

First tolerance level verifies just any two sequential minutiae in any two fingerprints have a same distance and angle, second tolerance level verifies four minutiae ,the third and last level needs to verifies six minutiae to mach any two fingerprints.

Table 4.3 Person 1 tolerance levels and matching time cost.

	Tolerance	No of Minutiae	Recognition	Time
Person 1	L	2	Yes	2.1s
	M	4	Yes	2.2s
	H	6	Yes	2.3s

Table 4.4 Person 2 tolerance levels and matching time cost.

	Tolerance	No of Minutiae	Recognition	Time
Person 2	L	2	Yes	1.7s
	M	4	Yes	1.8s
	H	6	Yes	1.9s

Table 4.5 Person 3 tolerance levels and matching time cost.

	Tolerance	No of Minutiae	Recognition	Time
Person 3	L	2	Yes	2s
	M	4	Yes	2.1s
	H	6	Yes	2.2s

Table 4.6 Person 4 tolerance levels and matching time cost.

	Tolerance	No of Minutiae	Recognition	Time
Person 4	L	2	Yes	2.1s
	M	4	Yes	2.2s
	H	6	Yes	2.3s

Table 4.7 Person 5 tolerance levels and matching time cost.

	Tolerance	No of Minutiae	Recognition	Time
Person 5	L	2	Yes	1.6s
	M	4	Yes	1.7s
	H	6	Yes	1.8s

4.5. Summary

This chapter presented a real-life application of the fingerprint detection technique.

This system was applied to five different fingerprints for five different persons to verify the system ability and efficiency.

The important thing in any fingerprint detection system is the result of this system being true or false.

CHAPTER FIVE
SOFTWARE DEVELOPMENT

5.1 Overview

This chapter presents the software development for the fingerprints detection system. The advantage of using Borland Delphi7 software will be described in this chapter. A guide to using the completed software will also be presented to provide a user-friendly interface for end users.

5.1 Software Environment

Fingerprints detection software built on Borland Delphi7; Delphi7 Studio is a cross-platform rapid application development (RAD) environment that integrates modeling, development, and deployment of Windows-based business and E-Commerce solutions that fully supports new and emerging Web Services. This product also feature full support for new and emerging Web Services, integrated model driven development, and preview capabilities for the Microsoft.NET Framework. Delphi 7 supports a first true model driven architecture with new UML solutions bundled in, updated cross-platform testing capabilities and Linux support with Kylix 3.

Borland is the first to deliver the independent path to .NET with Delphi 7 Studio, a cross-platform rapid application development (RAD) environment for the Windows platform. Delphi 7 Studio also features enterprise application design and deployment allowing developers to take advantage of enterprise application development from concept to production faster with the new UML designer and Model Driven Architecture (MDA) technology.

Features and benefits of Borland Delphi 7:

- Enterprise Application MDA Development: Speeds the development process by allowing developers to take an application from design to deployment while radically reducing the amount of code and development time required.
- RAD Visual Web Development: Allows developers to visually build Web applications within the Delphi 7 Studio environment and eliminates common server side development tasks with its Application Mode framework to handle session management transparently.

- Enterprise Class Reporting Capabilities: Allows developers to create cross-platform reports that help determine the efficiency of application performance.
- Royalty-Free DataSnap Multi-tier Application Deployment (formerly MIDAS): New Delphi 7 Studio DataSnap licensing allows developers to seamlessly scale single-tier and client/server applications to multi-tier without additional runtime fee requirements.
- Built-in Cross-Platform Support for Linux. Delphi 7 Studio will be shipped with the Delphi language version of Borland Kylix 3, the first high-performance, visual integrated development environment (IDE) for rapidly creating database, GUI, Web, and Web Services applications for the Linux operating system.
- Windows XP Applications: Delphi 7 Studio includes Windows XP Theme support, allowing developers to create applications that take advantage of Windows XP User Interface themes.

5.3 System Requirements

- Intel® Pentium® 233 MHz or higher.
- Microsoft Windows XP, 2000, 98.
- 64 MB RAM (128 MB recommended).
- 450 MB hard disk space for full install.
- SVGA or higher resolution monitor.
- Mouse or other pointing device.

5.4 Software Specifications

1. Load button

   Load button used to load fingerprint from machine to system.

2. Binarization button

   Binarization button used to convert gray fingerprint image to {black & white} image, thinning step depend on binarization.

3. Filling button

   Filling button used to fill any white space between black color on ridges before thinning image.

4. Thinning button

   Thinning button used to thinning fingerprint ridges to just 1 pixel width.

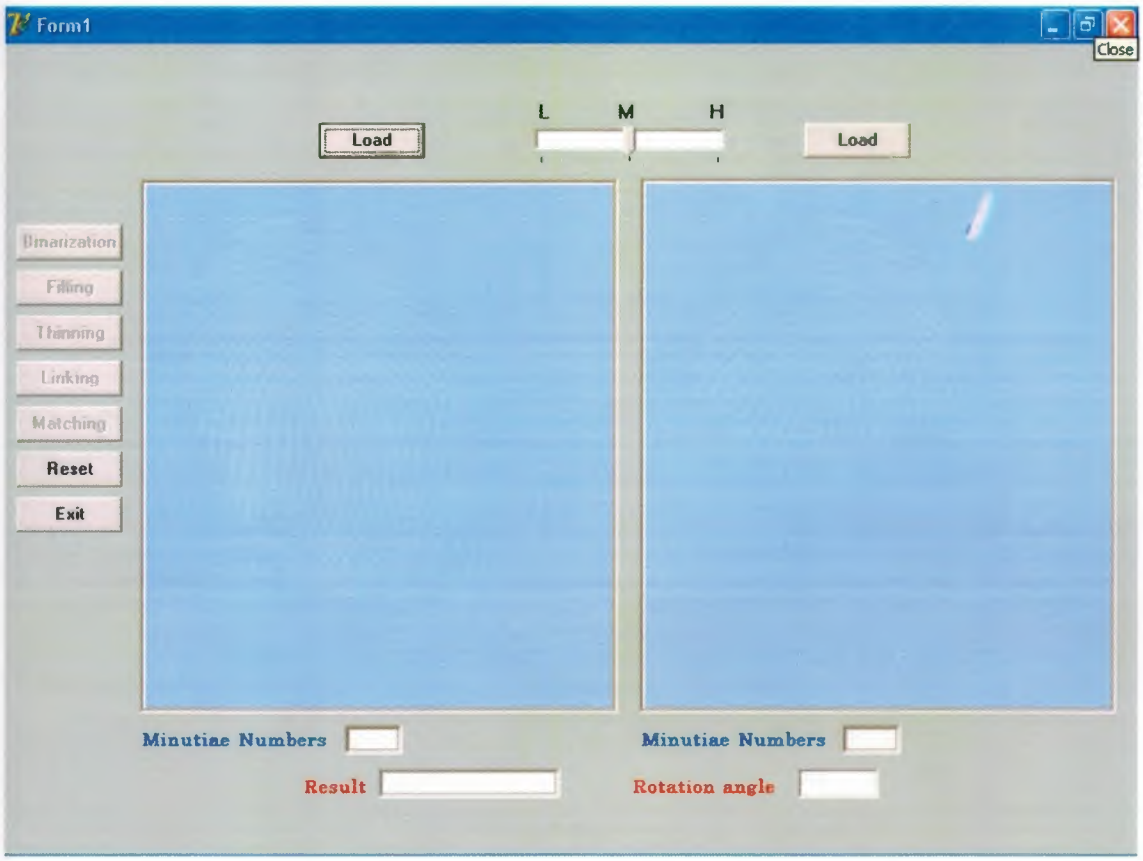


Figure 5.1 System Graphical User Interface (GUI).

5. Linking button

   Linking button used to connect cutting ridges in fingerprint.

6. Matching button

   Matching button is the last step used to verify two fingerprints and give the result of matching.

7. Reset button

   Reset button used to reset system and clear last application to using new application again.

8. Exit button

   Exit button used to shutdown system and exit.

9. Track bar

   Track bar is used to choice security level and determine tolerance by using three levels of tolerance High (H), Mid (M), Low (L).

Figure 5.2 display the result of developed fingerprint detection system when the used loaded two fingerprints and make matching between any two fingerprints.

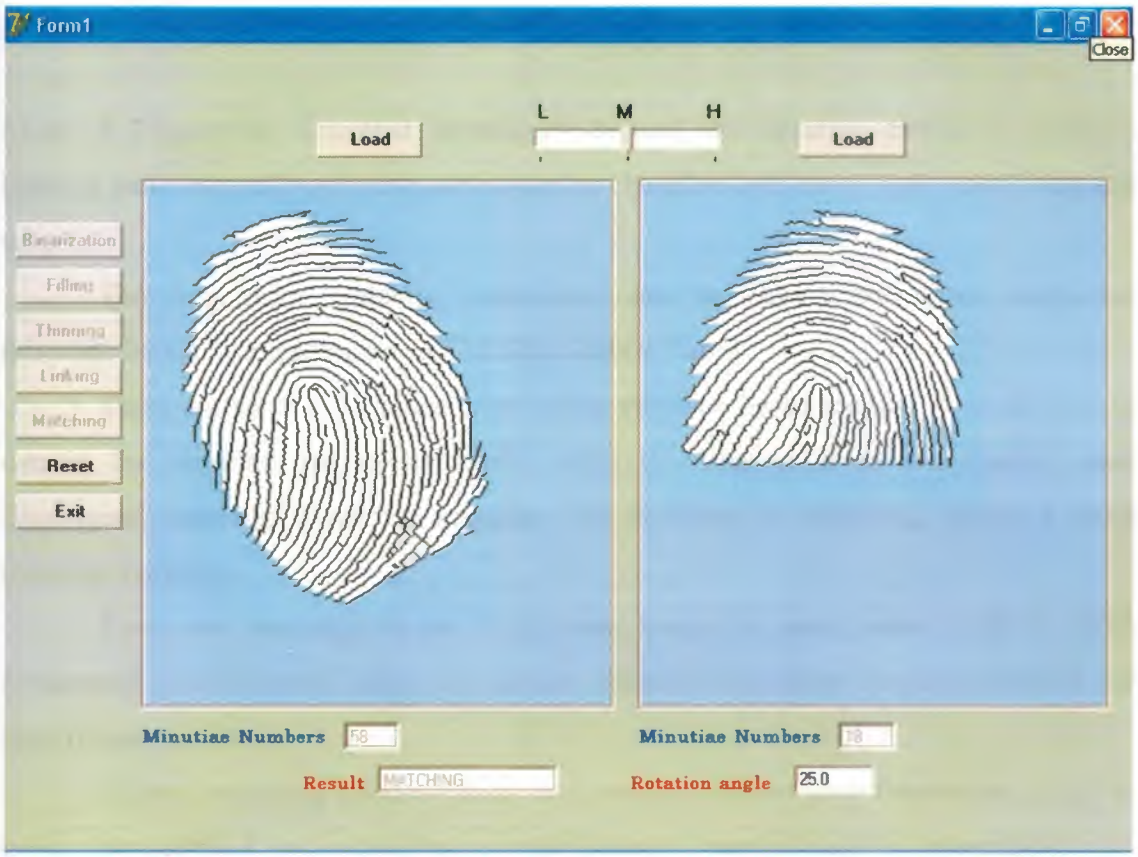


Figure 5.2 System result.

5.5 Summary

This chapter presented the software development for the fingerprints detection system. Also described the advantage of using Borland Delphi7 software. A guide to using the completed software will also be presented to provide a user-friendly interface for end users.

CONCLUSION

Most of Fingerprint detection techniques depend on minutiae extraction method; humans have unique fingerprints which can be classified into three types: arch, loop and whorl.

The fingerprint matching techniques can be placed into three categories minutiae-based, correlation-based and ridge feature-based.

There are some difficulties when using minutiae-based method. It is difficult to extract the minutiae points accurately when the fingerprint is low quality; also fingerprint matching based on minutiae has problems in matching different sized minutiae patterns.

There are two ways to get fingerprints image by using inked cards or FTIR (Frustrated Total Internal Reflection) device, most of fingerprint detection systems use gray fingerprint image.

Before extracting minutiae there is a need to convert gray fingerprint image to binary image (black and white) by generate factor to convert color to determined color level if color level is larger than color factor the color is white else the color is black.
The thinning ridge width to one pixel idea use 3x3 window to make thinning operation and change binary image ridges width to one.

The developed technique uses 8-nighbors to catch minutiae; this is done by using image pixels, if a pixel in the thinned image has more than two neighbors, then the minutiae is classified as a bifurcation, and if a pixel has only one neighbor, then the minutiae is classified as an ending.

The developed fingerprint detection technique creates tables of minutiae records contain distance and angels between horizontal sequential minutiae and fill records in fingerprint table.

The matching fingerprints process will be between two fingerprint tables by check every record inside tables. The developed technique is able to match rotate fingerprint and damaged fingerprint its not important using FTIR device or ink card.

Distances and angles added efficiency in detection system because no sequential minutiae have same distance and angle in any different fingerprints.

The system matching time cost depends on the number of pixels that the fingerprint has. Small fingerprints need less time than bigger fingerprint but our

developed fingerprint detection technique is approximately faster than the other techniques because it detects just two parameters (distances and angles).

The developed fingerprint detection technique may be used to monitor employer may wish to automate timecard clock-ins and clock-outs for employees to ensure that they are working the requisite number of hours and not billing the company for hours they have not spent at work, also we used in criminal justice fingerprint identification system provides the ability to scan a ten print card and capture the fingerprint images from the card or from a live fingerprint scan device. The purpose of entering the ten fingers and other data to the system is to determine if the person represented by the fingerprints has had prior contact with the criminal justice.

The main goal of developed fingerprint scheme is to solve common fingerprint problems with the existing techniques (e.g. rotation, fingerprint damage ...etc), and to develop software that simulates the new technique using Delphi program language.

The developed technique realized the efficiency of the new technique via real life application (fingerprint detection of five persons).

An introduction to fingerprints and their characteristics was presented in chapter one. Common techniques of fingerprint detection were also described. These include minutiae-based, correlation-based and ridge feature-based advantages and disadvantages of these techniques were also presented.

The minutiae-based method is the most common in fingerprints detections technique because it depends on minutiae; fingerprint detection methods have a lot of steps; these were presented within chapter two.

Chapter three contains developed algorithm match fingerprint by some steps ,first before doing any thing must convert fingerprint image to binary image, second enhance fingerprint image and drop damaged part, third thinning ridges ,fourth step is minutiae extraction .

The real-life application of the fingerprint detection technique represented in chapter four.

This system used five different fingerprints for five different persons to verify the system ability and efficiency.

The important thing in any fingerprint detection system is the result of this system is being true or false.

Chapter five for end user because have a lot of important steps for any one need to use fingerprint detection system, Also described which program language the system has built in and the advantage of program language.

REFERENCES

[1] www.findbiometrics.com/Pages/fingerprint_articles/fingerprint_4.html.

[2] www.cis.rit.edu/research/thesis/bs/1999/chang/thesis.html.

[3] http://eprint.iacr.org/2004/021.pdf.

[4] Jain, K., Hong, L., Pankanti, S., Bolle, R.: An identity-authentication system using fingerprints. In: Proc. of IEEE. Volume 85. (1997) 1365{1388

[5] N.K. Ratha, S. Chen and A.K. Jain, "Adaptive flow orientation-based feature extraction in fingerprint images," Pattern Recognition, Vol. 28, No. 11, pp. 1657-1672, 1995.

[6] http://biometrics.cse.msu.edu/fingerprint.html

[7] Jain, K., Hong, L., Pankanti, S., Bolle, R.: An identity-authentication system using fingerprints. In: Proc. of IEEE. Volume 85. (1997) 1365{1388

[8] http://www.cse.msu.edu/cgi-user/web/tech/document

[9] R. Hopkins, An Introduction to Biometrics and Large Scale Civilian Identification, International Review of Law, Computers and Technology, Vol.13, Issue 3, December 1999.

[10] A. Jain, L. Hong and R. Bolle, On-Line Fingerprint Verification, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, No. 4, April 1997.

[11] N. Ratha, K. Karu, S. Chen, and A. Jain, A Real-Time Matching System For Large fingerprint Databases, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, No. 8, August 1996.

[12] Microsystems for Biometrics FingerTIP CMOS Chip and System Data book 3.1, Infineon Technologies AG, CC Applications Group, Minchin, 1999.

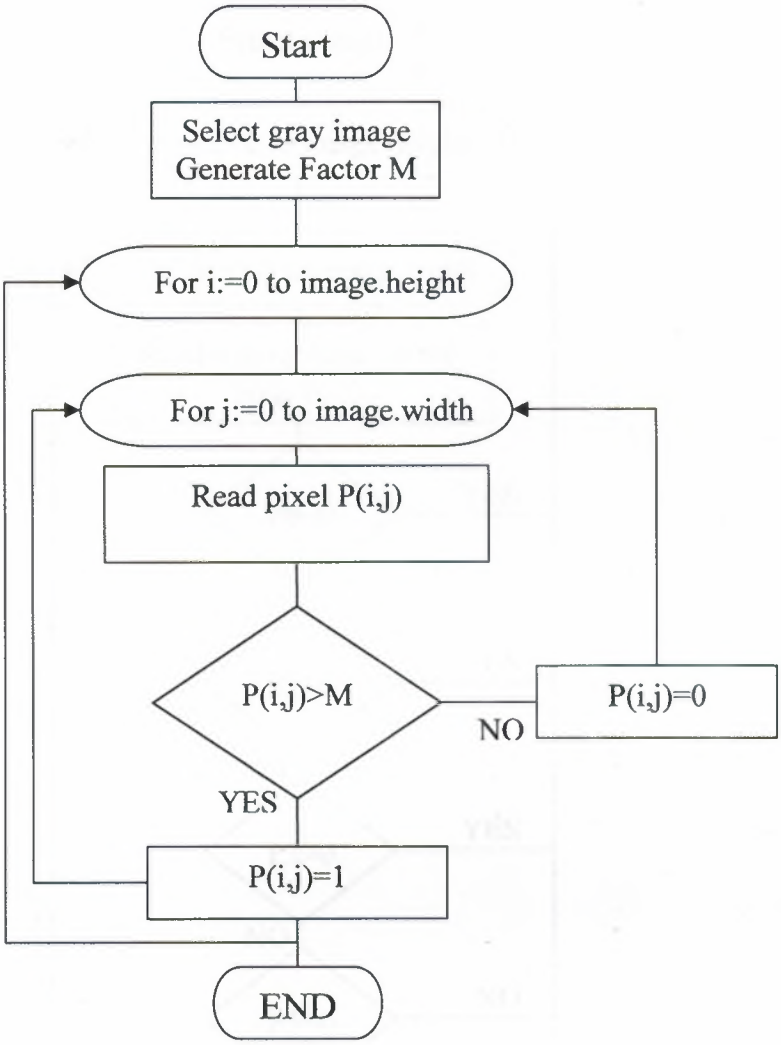
[13] M. Hanson, Fingerprint-Based forensics Identify Argentines Desaperecidos, Retrieved October 2000, http://www.computer.org/cga/articles/fingerprints.htm, 2000.

[14] N. Ratha, K. Karu, S. Chen, and A. Jain, A Real-Time Matching System For Large fingerprint Databases, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, No. 8, August 1996.

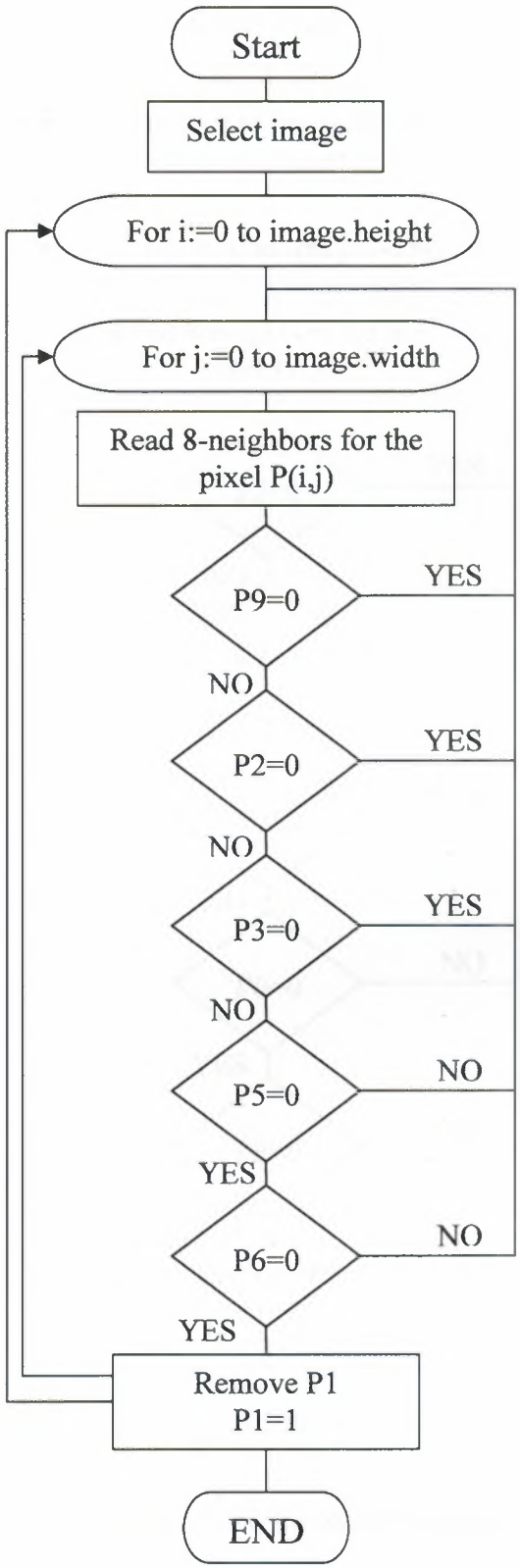
[15] A. Jain, S. Prbhakar, L. Hong and s. Pankanti, Filterbank-Based Fingerprint Matching, IEEE Transactions On Image Processing, Vol. 9, No. 5, May 2000.

APPENDEX A

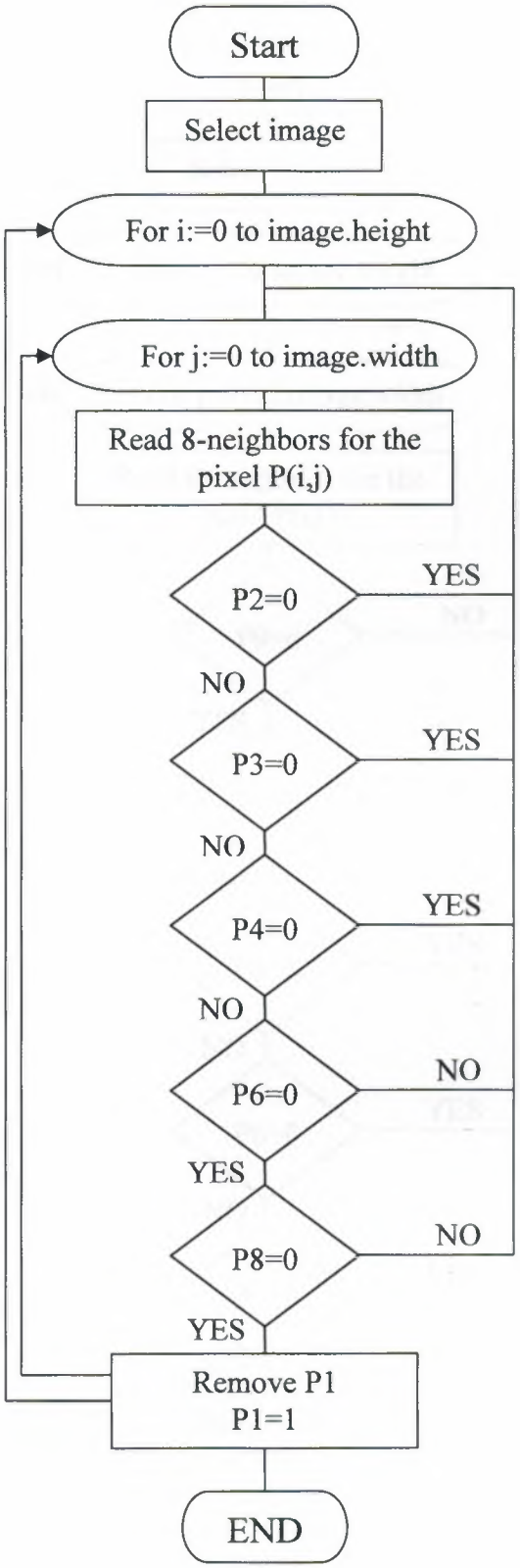
1. Binarization Flowchart



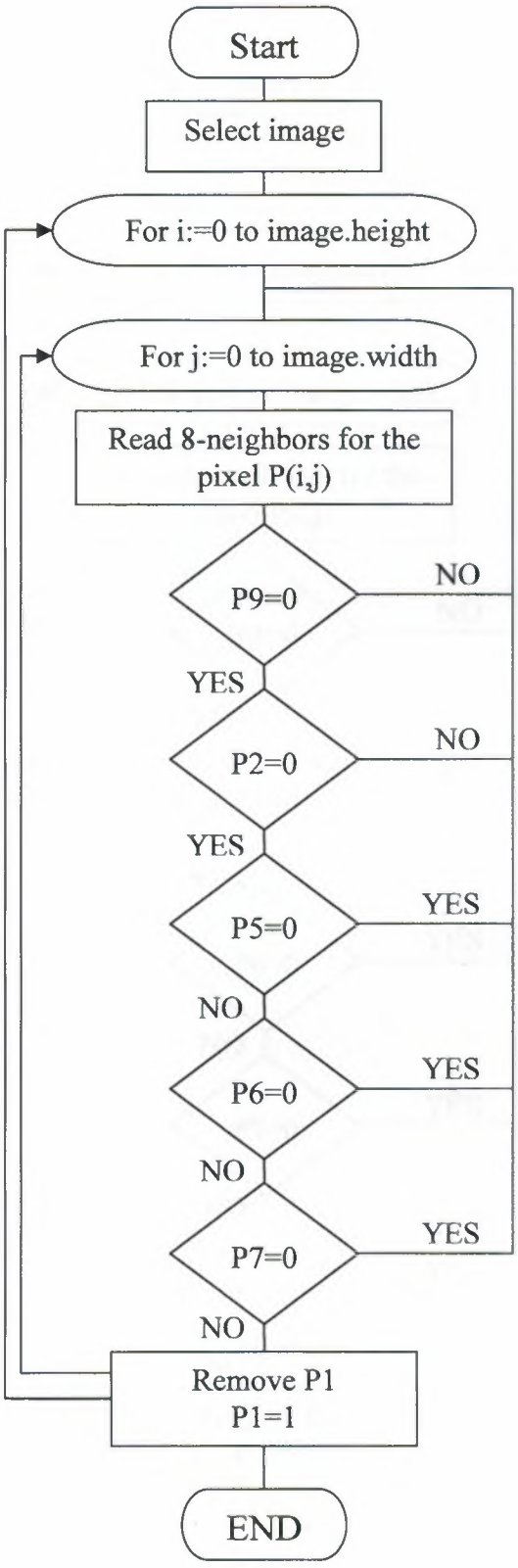
2. Thinning Flowchart



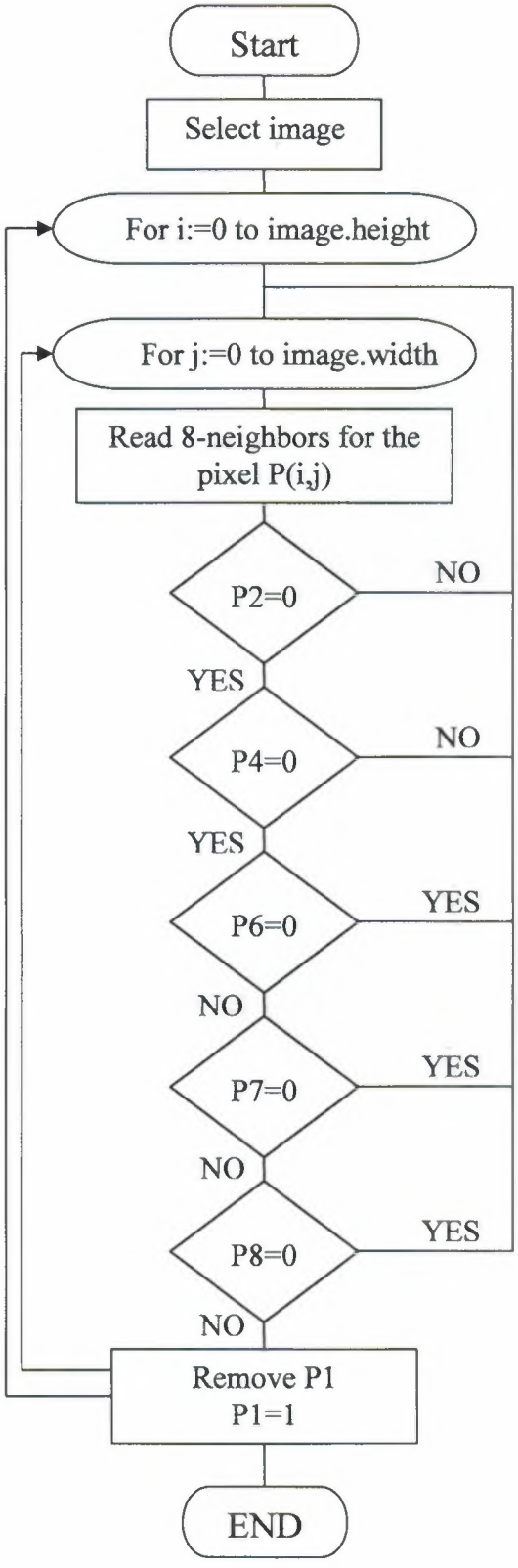
Thinning ridge algorithm Flowchart sample (1)



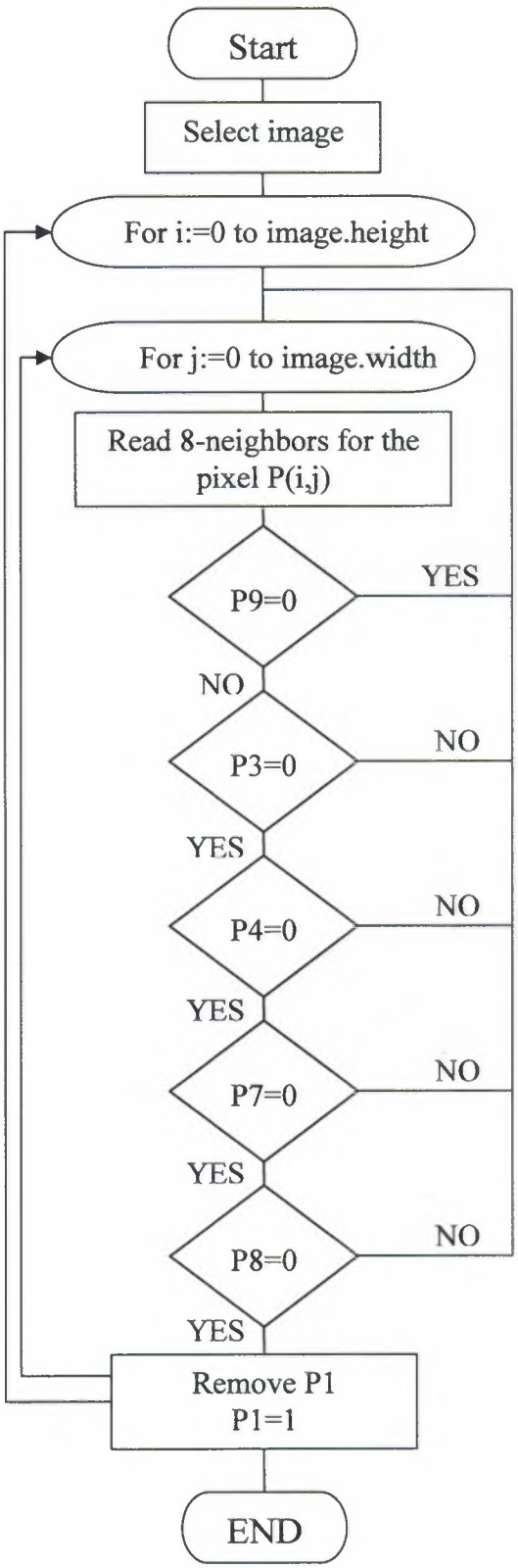
Thinning ridge algorithm Flowchart sample (2)



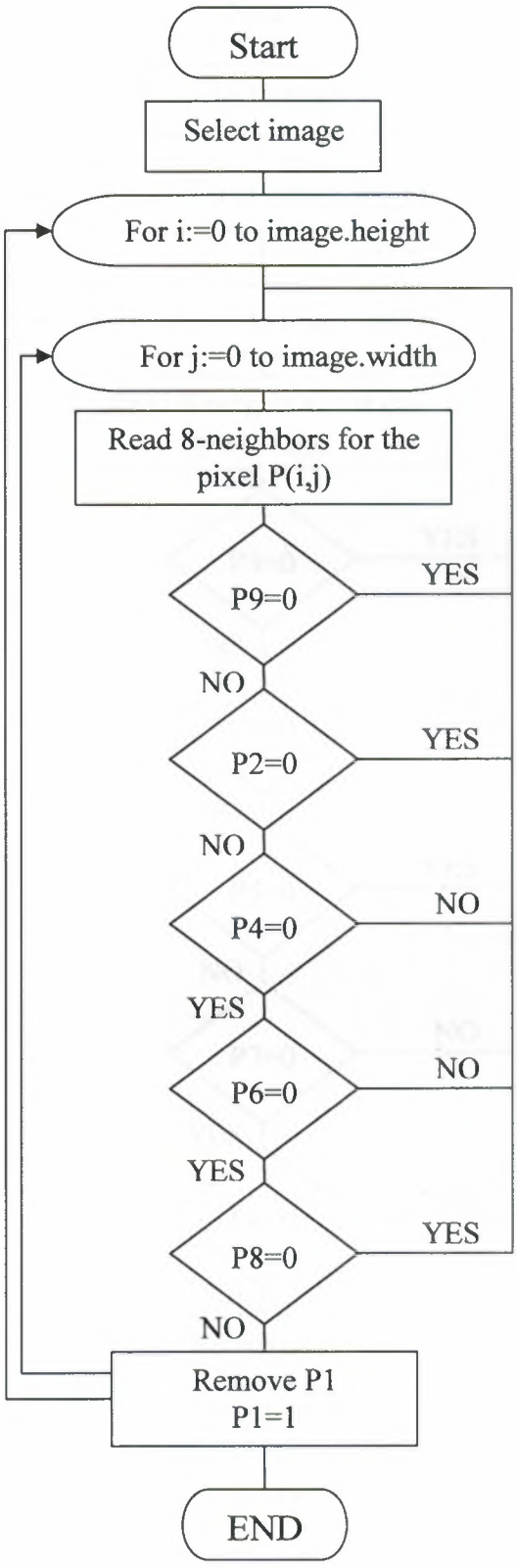
Thinning ridge algorithm Flowchart sample (3)



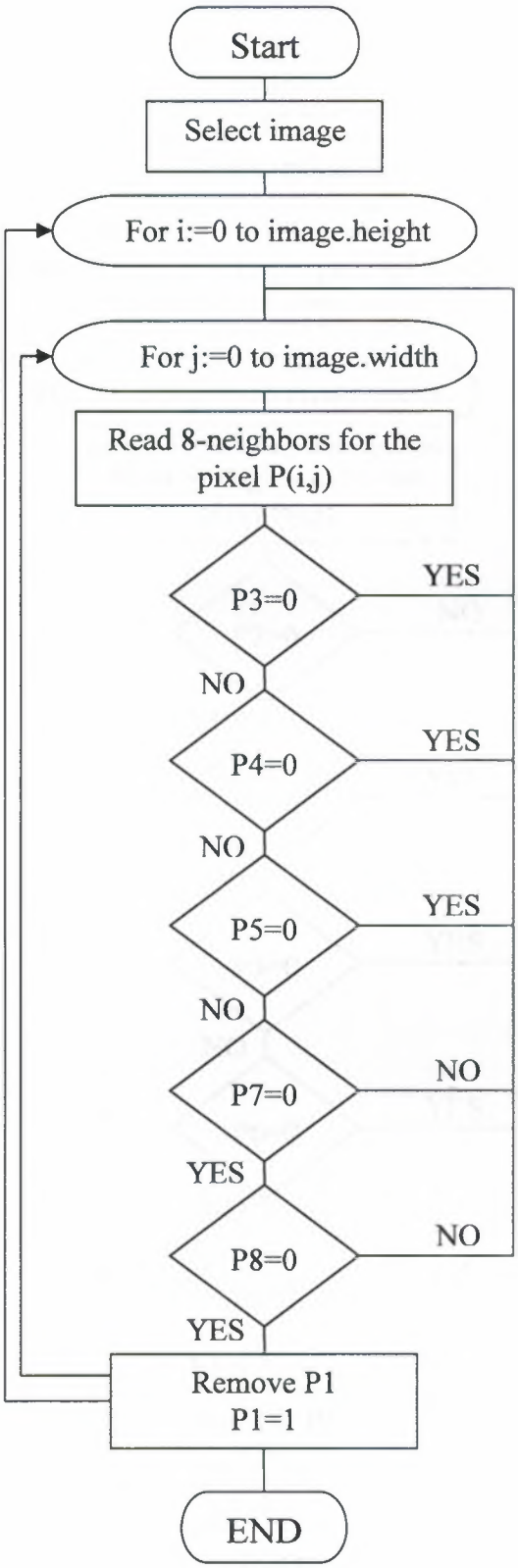
Thinning ridge algorithm Flowchart sample (4)



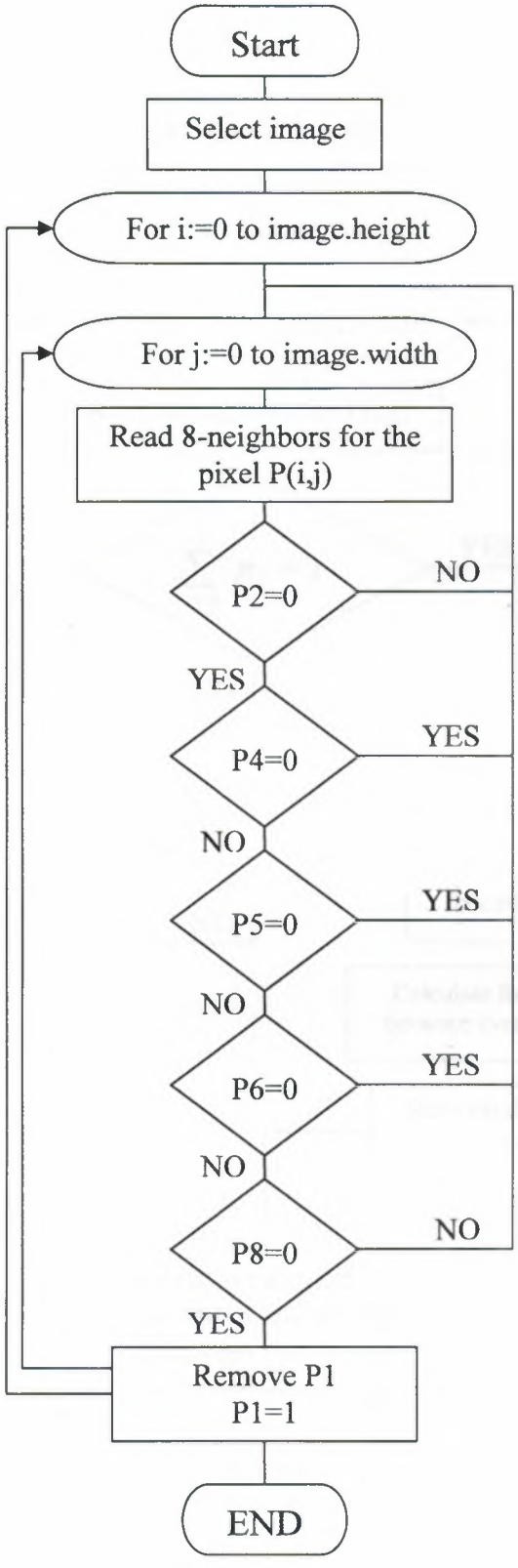
Thinning ridge algorithm Flowchart sample (5)



Thinning ridge algorithm Flowchart sample (6)

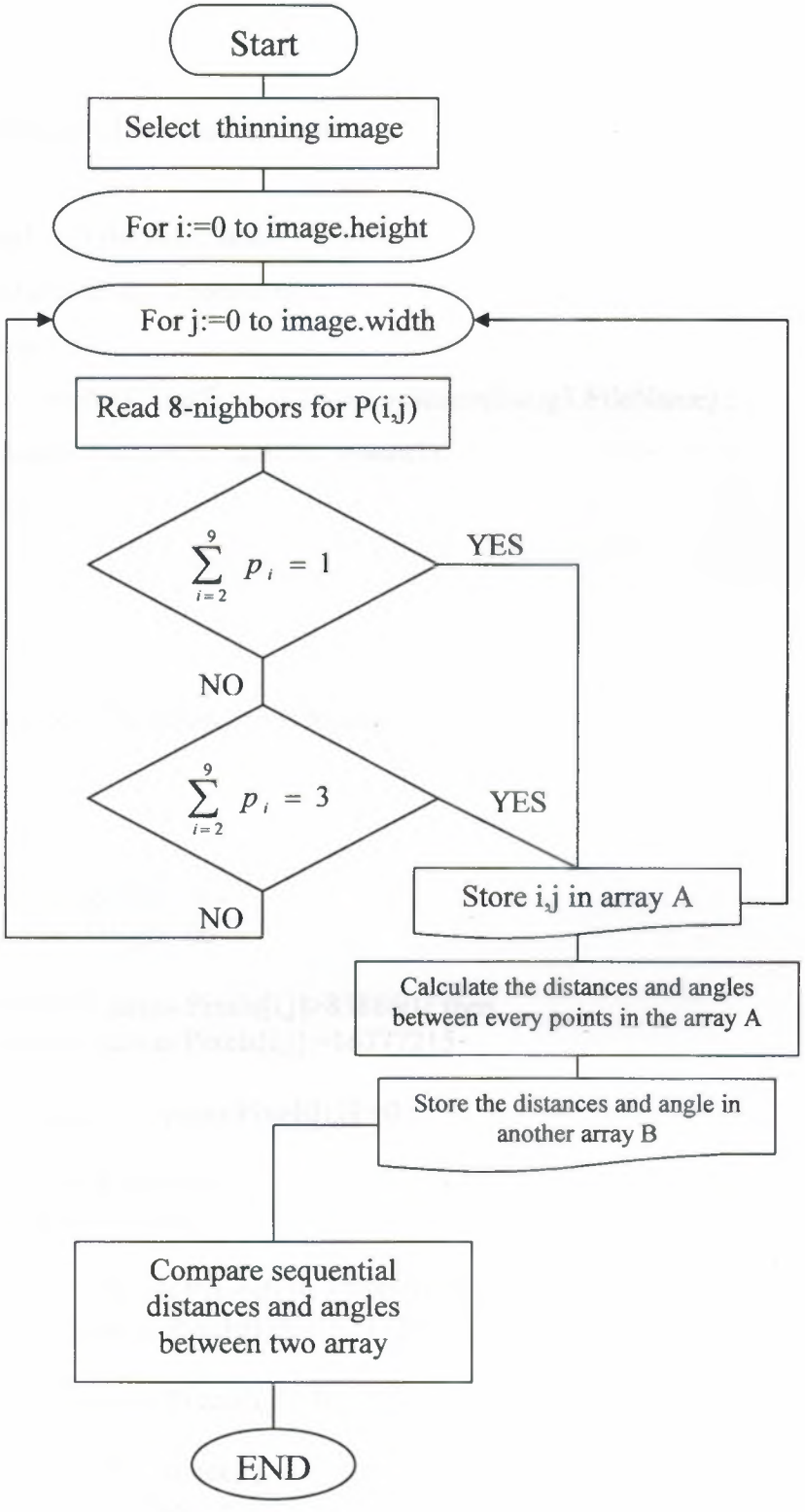


Thinning ridge algorithm Flowchart sample (7)



Thinning ridge algorithm Flowchart sample (8)

3. Matching Flowchart



APPENDIX B

1. Load Button

```
procedure TForm1.Button4Click(Sender: TObject);
    begin
        fin_source1 := TBitmap.Create;
        if openpicturedialog1.Execute then
            begin
              fin_source1.LoadFromFile(openpicturedialog1.FileName) ;
              image1.Picture.Assign(fin_source1);
            end;
   end;
```

2. Binarization

```
procedure TForm1.Button3Click(Sender: TObject);
   var
     i,j:integer;
     s:string;
begin
   for i:=0 to fin_source1.Width   do
   for j:=0 to fin_source1.Height  do
     begin
          if  fin_source1.Canvas.Pixels[i,j]>8388607 then
             fin_source1.canvas.Pixels[i,j]:=16777215
           else
                fin_source1.Canvas.Pixels[i,j]:=0 ;
      end;
    for i:=0 to fin_source2.Width   do
    for j:=0 to fin_source2.Height  do
      begin
          if  fin_source2.Canvas.Pixels[i,j]>8388607 then
            fin_source2.canvas.Pixels[i,j]:=16777215
           else
            fin_source2.Canvas.Pixels[i,j]:=0 ;
      end;
    image1.Picture.Assign(fin_source1);
    image2.Picture.Assign(fin_source2);
end;
```

3. Filling Button

```
procedure TForm1.Button1Click(Sender: TObject);
    var
    i,j:integer;
    np,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,p16,p17,p18,p19,p20,p
    21,p22,p23,p24:longint;
    begin
    with fin_source1 do
     for j:=0 to fin_source1.Height  do
      for i:=0 to fin_source1.Width  do
        begin
          p1:=Canvas.Pixels[i-2,j-2];
          p2:=Canvas.Pixels[i-1,j-2];
          p3:=Canvas.Pixels[i,j-2];
          p4:=Canvas.Pixels[i+1,j-2];
          p5:=Canvas.Pixels[i+2,j-2];
          p6:=Canvas.Pixels[i+2,j-1];
          p7:=Canvas.Pixels[i+2,j];
          p8:=Canvas.Pixels[i+2,j+1];
          p9:=Canvas.Pixels[i+2,j+2];
          p10:=Canvas.Pixels[i+1,j+2];
          p11:=Canvas.Pixels[i,j+2];
          p12:=Canvas.Pixels[i-1,j+2];
          p13:=Canvas.Pixels[i-2,j+2];
          p14:=Canvas.Pixels[i-2,j+1];
          p15:=Canvas.Pixels[i-2,j];
          p16:=Canvas.Pixels[i-2,j-1];
          p17:=Canvas.Pixels[i,j-1];
          p18:=Canvas.Pixels[i+1,j-1];
          p19:=Canvas.Pixels[i+1,j];
          p20:=Canvas.Pixels[i+1,j+1];
          p21:=Canvas.Pixels[i,j+1];
          p22:=Canvas.Pixels[i-1,j+1];
          p23:=Canvas.Pixels[i-1,j];
          p24:=Canvas.Pixels[i-1,j-1];
  if ((p1=0)and(p2=0)and(p3=0)and(p4=0)and
(p5=0)and(p6=0)and(p7=0)and(p8=0)and(p9=0)and
  (p10=0)and(p11=0)and(p12=0)and(p13=0)and(p14=0)and
  (p15=0)and(p16=0)) or  ((p17=0)and(p18=0)and(p19=0)and(p20=0)and
  (p21=0)and(p22=0)and(p23=0)and(p24=0))   then
       begin
          Canvas.Pixels[i,j-1]:=0;
          Canvas.Pixels[i+1,j-1]:=0;
          Canvas.Pixels[i+1,j]:=0;
          Canvas.Pixels[i+1,j+1]:=0;
          Canvas.Pixels[i,j+1]:=0;
          Canvas.Pixels[i-1,j+1]:=0;
          Canvas.Pixels[i-1,j]:=0;
          Canvas.Pixels[i-1,j-1]:=0;
```

```
            Canvas.Pixels[i,j]:=0;
        end;
 end;
 with fin_source2 do
for j:=0 to fin_source2.Height   do
 for i:=0 to fin_source2.Width   do
 begin
   p1:=Canvas.Pixels[i-2,j-2];
   p2:=Canvas.Pixels[i-1,j-2];
   p3:=Canvas.Pixels[i,j-2];
   p4:=Canvas.Pixels[i+1,j-2];
   p5:=Canvas.Pixels[i+2,j-2];
   p6:=Canvas.Pixels[i+2,j-1];
   p7:=Canvas.Pixels[i+2,j];
   p8:=Canvas.Pixels[i+2,j+1];
   p9:=Canvas.Pixels[i+2,j+2];
   p10:=Canvas.Pixels[i+1,j+2];
   p11:=Canvas.Pixels[i,j+2];
   p12:=Canvas.Pixels[i-1,j+2];
   p13:=Canvas.Pixels[i-2,j+2];
   p14:=Canvas.Pixels[i-2,j+1];
   p15:=Canvas.Pixels[i-2,j];
   p16:=Canvas.Pixels[i-2,j-1];
   p17:=Canvas.Pixels[i,j-1];
   p18:=Canvas.Pixels[i+1,j-1];
   p19:=Canvas.Pixels[i+1,j];
   p20:=Canvas.Pixels[i+1,j+1];
   p21:=Canvas.Pixels[i,j+1];
   p22:=Canvas.Pixels[i-1,j+1];
   p23:=Canvas.Pixels[i-1,j];
   p24:=Canvas.Pixels[i-1,j-1];
  if ((p1=0)and(p2=0)and(p3=0)and(p4=0)and
  (p5=0)and(p6=0)and(p7=0)and(p8=0)and(p9=0)and
  (p10=0)and(p11=0)and(p12=0)and(p13=0)and(p14=0)and
  (p15=0)and(p16=0)) or     ((p17=0)and(p18=0)and(p19=0)and(p20=0)and
  (p21=0)and(p22=0)and(p23=0)and(p24=0))    then
        begin
            Canvas.Pixels[i,j-1]:=0;
            Canvas.Pixels[i+1,j-1]:=0;
            Canvas.Pixels[i+1,j]:=0;
            Canvas.Pixels[i+1,j+1]:=0;
            Canvas.Pixels[i,j+1]:=0;
            Canvas.Pixels[i-1,j+1]:=0;
            Canvas.Pixels[i-1,j]:=0;
            Canvas.Pixels[i-1,j-1]:=0;
            Canvas.Pixels[i,j]:=0;
        end;
 end;
  image2.Picture.Assign(fin_source2);
```

```
  image1.Picture.Assign(fin_source1);
end;
```

4. Thinning Button

```
procedure TForm1.Button7Click(Sender: TObject);
var
i,j,k,l:integer;
p:longint;
np,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,p16,p17,p18,p19,p20,p21,p22,
p23,p24:longint;
begin
    with fin_source1 do
   for j:=0 to fin_source1.Height   do
for i:=0 to fin_source1.Width   do
 begin
   p2:=Canvas.Pixels[i,j-1];
   p3:=Canvas.Pixels[i+1,j-1];
   p4:=Canvas.Pixels[i+1,j];
   p5:=Canvas.Pixels[i+1,j+1];
   p6:=Canvas.Pixels[i,j+1];
   p7:=Canvas.Pixels[i-1,j+1];
   p8:=Canvas.Pixels[i-1,j];
   p9:=Canvas.Pixels[i-1,j-1];
   np:=p2+p3+p4+p5+p6+p7+p8+p9;
    begin
     if (((p9<>0) and (p2<>0)) and (((p3<>0) and (p5=0)) and (p6=0))) then
      canvas.Pixels[i,j]:=16777215;
     if (((p9=0) and (p2=0)) and (((p5<>0) and (p6<>0)) and (p7<>0))) then
      canvas.Pixels[i,j]:=16777215;
     if (((p9<>0) and (p8<>0)) and (((p7<>0) and (p3=0)) and (p4=0))) then
      canvas.Pixels[i,j]:=16777215;
     if (((p3<>0) and (p4<>0)) and (((p5<>0) and (p7=0)) and (p8=0))) then
      canvas.Pixels[i,j]:=16777215;
     if (((p2<>0) and (p3<>0)) and (((p4<>0) and (p6=0)) and (p8=0))) then
      canvas.Pixels[i,j]:=16777215;
      if (((p6<>0) and (p7<>0)) and (((p8<>0) and (p2=0)) and (p4=0))) then
      canvas.Pixels[i,j]:=16777215;
      if (((p8<>0) and (p9<>0)) and (((p2<>0) and (p4=0)) and (p6=0))) then
      canvas.Pixels[i,j]:=16777215;
      if (((p4<>0) and (p5<>0)) and (((p6<>0) and (p8=0)) and (p2=0))) then
      canvas.Pixels[i,j]:=16777215;
    end;
 end;
with fin_source1 do
for i:=1 to width do
 for j:=1 to height do
   begin
        p:=Canvas.Pixels[i-1,j-1]+Canvas.Pixels[i-1,j]
          +Canvas.Pixels[i-1,j+1]+Canvas.Pixels[i,j+1]
```

```
        +Canvas.Pixels[i+1,j+1]+Canvas.Pixels[i+1,j]
        +Canvas.Pixels[i+1,j-1]+Canvas.Pixels[i,j-1];
      if (((canvas.Pixels[i,j-1]=0)
                        and (canvas.Pixels[i+1,j]=0)
                        and (canvas.Pixels[i,j+1]=0))
                          or ((canvas.Pixels[i+1,j]=0)
                        and (canvas.Pixels[i,j+1]=0)
                        and (canvas.Pixels[i-1,j]=0))
                          or ((canvas.Pixels[i+1,j-1]=0)
                        and (canvas.Pixels[i+1,j]=0)
                        and (canvas.Pixels[i+1,j+1]=0))
                          or ((canvas.Pixels[i-1,j+1]=0)
                        and (canvas.Pixels[i,j+1]=0)
                        and (canvas.Pixels[i+1,j+1]=0))
                          or ((canvas.Pixels[i-1,j-1]=0)
                        and (canvas.Pixels[i-1,j]=0)
                        and (canvas.Pixels[i,j+1]=0))
                        or ((canvas.Pixels[i-1,j]=0)
                        and (canvas.Pixels[i,j+1]=0)
                        and (canvas.Pixels[i+1,j]=0))
                        or ((canvas.Pixels[i,j-1]=0)
                        and (canvas.Pixels[i+1,j-1]=0)
                        and (canvas.Pixels[i+1,j]=0))
                        or {((canvas.Pixels[i+1,j]=0)
                        and (canvas.Pixels[i+1,j+1]=0)
                        and (canvas.Pixels[i,j+1]=0))
                        or }((canvas.Pixels[i-1,j]=0)
                        and (canvas.Pixels[i-1,j+1]=0)
                        and (canvas.Pixels[i,j+1]=0))
                        or ((canvas.Pixels[i-1,j+1]=0)
                        and (canvas.Pixels[i,j+1]=0)
                        and (canvas.Pixels[i+1,j+1]=0)))
                then
        canvas.Pixels[i,j]:=16777215;
    end;
     with fin_source1 do
   for j:=0 to fin_source1.Height   do
for i:=0 to fin_source1.Width   do
  begin
   p2:=Canvas.Pixels[i,j-1];
   p3:=Canvas.Pixels[i+1,j-1];
   p4:=Canvas.Pixels[i+1,j];
   p5:=Canvas.Pixels[i+1,j+1];
   p6:=Canvas.Pixels[i,j+1];
   p7:=Canvas.Pixels[i-1,j+1];
   p8:=Canvas.Pixels[i-1,j];
   p9:=Canvas.Pixels[i-1,j-1];
   np:=p2+p3+p4+p5+p6+p7+p8+p9;
    begin
```

```
      if (((p9<>0) and (p2<>0)) and (((p3<>0) and (p5=0)) and (p6=0))) then
       canvas.Pixels[i,j]:=16777215;
      if (((p9=0) and (p2=0)) and (((p5<>0) and (p6<>0)) and (p7<>0))) then
       canvas.Pixels[i,j]:=16777215;
      if (((p9<>0) and (p8<>0)) and (((p7<>0) and (p3=0)) and (p4=0))) then
       canvas.Pixels[i,j]:=16777215;
      if (((p3<>0) and (p4<>0)) and (((p5<>0) and (p7=0)) and (p8=0))) then
       canvas.Pixels[i,j]:=16777215;
      if (((p2<>0) and (p3<>0)) and (((p4<>0) and (p6=0)) and (p8=0))) then
       canvas.Pixels[i,j]:=16777215;
       if (((p6<>0) and (p7<>0)) and (((p8<>0) and (p2=0)) and (p4=0))) then
       canvas.Pixels[i,j]:=16777215;
       if (((p8<>0) and (p9<>0)) and (((p2<>0) and (p4=0)) and (p6=0))) then
       canvas.Pixels[i,j]:=16777215;
       if (((p4<>0) and (p5<>0)) and (((p6<>0) and (p8=0)) and (p2=0))) then
       canvas.Pixels[i,j]:=16777215;
     end;
  end;
with fin_source1 do
for j:=0 to fin_source1.Height   do
 for i:=0 to fin_source1.Width   do
  begin
    p1:=Canvas.Pixels[i-2,j-2];
    p2:=Canvas.Pixels[i-1,j-2];
    p3:=Canvas.Pixels[i,j-2];
    p4:=Canvas.Pixels[i+1,j-2];
    p5:=Canvas.Pixels[i+2,j-2];
    p6:=Canvas.Pixels[i+2,j-1];
    p7:=Canvas.Pixels[i+2,j];
    p8:=Canvas.Pixels[i+2,j+1];
    p9:=Canvas.Pixels[i+2,j+2];
    p10:=Canvas.Pixels[i+1,j+2];
    p11:=Canvas.Pixels[i,j+2];
    p12:=Canvas.Pixels[i-1,j+2];
    p13:=Canvas.Pixels[i-2,j+2];
    p14:=Canvas.Pixels[i-2,j+1];
    p15:=Canvas.Pixels[i-2,j];
    p16:=Canvas.Pixels[i-2,j-1];
    p17:=Canvas.Pixels[i,j-1];
    p18:=Canvas.Pixels[i+1,j-1];
    p19:=Canvas.Pixels[i+1,j];
    p20:=Canvas.Pixels[i+1,j+1];
    p21:=Canvas.Pixels[i,j+1];
    p22:=Canvas.Pixels[i-1,j+1];
    p23:=Canvas.Pixels[i-1,j];
    p24:=Canvas.Pixels[i-1,j-1];
    if ((p1<>0)and(p2<>0)and(p3<>0)and(p4<>0)and
   (p5<>0)and(p6<>0)and(p7<>0)and(p8<>0)and(p9<>0)and
   (p10<>0)and(p11<>0)and(p12<>0)and(p13<>0)and(p14<>0)and
```

```
 (p15<>0)and(p16<>0)) or
  ((p17<>0)and(p18<>0)and(p19<>0)and(p20<>0)and
 (p21<>0)and(p22<>0)and(p23<>0)and(p24<>0))
  then
       begin
   Canvas.Pixels[i,j-1]:=16777215;
   Canvas.Pixels[i+1,j-1]:=16777215;
   Canvas.Pixels[i+1,j]:=16777215;
   Canvas.Pixels[i+1,j+1]:=16777215;
   Canvas.Pixels[i,j+1]:=16777215;
   Canvas.Pixels[i-1,j+1]:=16777215;
   Canvas.Pixels[i-1,j]:=16777215;
   Canvas.Pixels[i-1,j-1]:=16777215;
   Canvas.Pixels[i,j]:=16777215;
       end;
 end;
    image1.Picture.Assign(fin_source1);
   with fin_source2 do
  for j:=0 to fin_source2.Height   do
for i:=0 to fin_source2.Width   do
 begin
  p2:=Canvas.Pixels[i,j-1];
  p3:=Canvas.Pixels[i+1,j-1];
  p4:=Canvas.Pixels[i+1,j];
  p5:=Canvas.Pixels[i+1,j+1];
  p6:=Canvas.Pixels[i,j+1];
  p7:=Canvas.Pixels[i-1,j+1];
  p8:=Canvas.Pixels[i-1,j];
  p9:=Canvas.Pixels[i-1,j-1];
  np:=p2+p3+p4+p5+p6+p7+p8+p9;
   begin
    if (((p9<>0) and (p2<>0)) and (((p3<>0) and (p5=0)) and (p6=0))) then
     canvas.Pixels[i,j]:=16777215;
    if (((p9=0) and (p2=0)) and (((p5<>0) and (p6<>0)) and (p7<>0))) then
     canvas.Pixels[i,j]:=16777215;
    if (((p9<>0) and (p8<>0)) and (((p7<>0) and (p3=0)) and (p4=0))) then
     canvas.Pixels[i,j]:=16777215;
    if (((p3<>0) and (p4<>0)) and (((p5<>0) and (p7=0)) and (p8=0))) then
     canvas.Pixels[i,j]:=16777215;
    if (((p2<>0) and (p3<>0)) and (((p4<>0) and (p6=0)) and (p8=0))) then
     canvas.Pixels[i,j]:=16777215;
     if (((p6<>0) and (p7<>0)) and (((p8<>0) and (p2=0)) and (p4=0))) then
     canvas.Pixels[i,j]:=16777215;
     if (((p8<>0) and (p9<>0)) and (((p2<>0) and (p4=0)) and (p6=0))) then
     canvas.Pixels[i,j]:=16777215;
     if (((p4<>0) and (p5<>0)) and (((p6<>0) and (p8=0)) and (p2=0))) then
     canvas.Pixels[i,j]:=16777215;
   end;
 end;
```

```
with fin_source2 do
for i:=1 to width do
 for j:=1 to height do
    begin
         p:=Canvas.Pixels[i-1,j-1]+Canvas.Pixels[i-1,j]
           +Canvas.Pixels[i-1,j+1]+Canvas.Pixels[i,j+1]
           +Canvas.Pixels[i+1,j+1]+Canvas.Pixels[i+1,j]
           +Canvas.Pixels[i+1,j-1]+Canvas.Pixels[i,j-1];
          if (((canvas.Pixels[i,j-1]=0)
                              and (canvas.Pixels[i+1,j]=0)
                              and (canvas.Pixels[i,j+1]=0))
                                 or ((canvas.Pixels[i+1,j]=0)
                              and (canvas.Pixels[i,j+1]=0)
                              and (canvas.Pixels[i-1,j]=0))
                                 or ((canvas.Pixels[i+1,j-1]=0)
                              and (canvas.Pixels[i+1,j]=0)
                              and (canvas.Pixels[i+1,j+1]=0))
                                 or ((canvas.Pixels[i-1,j+1]=0)
                              and (canvas.Pixels[i,j+1]=0)
                              and (canvas.Pixels[i+1,j+1]=0))
                                 or ((canvas.Pixels[i-1,j-1]=0)
                              and (canvas.Pixels[i-1,j]=0)
                              and (canvas.Pixels[i,j+1]=0))
                              or ((canvas.Pixels[i-1,j]=0)
                              and (canvas.Pixels[i,j+1]=0)
                              and (canvas.Pixels[i+1,j]=0))
                              or ((canvas.Pixels[i,j-1]=0)
                              and (canvas.Pixels[i+1,j-1]=0)
                              and (canvas.Pixels[i+1,j]=0))
                              or {((canvas.Pixels[i+1,j]=0)
                              and (canvas.Pixels[i+1,j+1]=0)
                              and (canvas.Pixels[i,j+1]=0))
                              or }((canvas.Pixels[i-1,j]=0)
                              and (canvas.Pixels[i-1,j+1]=0)
                              and (canvas.Pixels[i,j+1]=0))
                              or ((canvas.Pixels[i-1,j+1]=0)
                              and (canvas.Pixels[i,j+1]=0)
                              and (canvas.Pixels[i+1,j+1]=0)))
                     then
             canvas.Pixels[i,j]:=16777215;
     end;
      with fin_source2 do
   for j:=0 to fin_source2.Height   do
for i:=0 to fin_source2.Width   do
 begin
   p2:=Canvas.Pixels[i,j-1];
   p3:=Canvas.Pixels[i+1,j-1];
   p4:=Canvas.Pixels[i+1,j];
   p5:=Canvas.Pixels[i+1,j+1];
```

```
  p6:=Canvas.Pixels[i,j+1];
  p7:=Canvas.Pixels[i-1,j+1];
  p8:=Canvas.Pixels[i-1,j];
  p9:=Canvas.Pixels[i-1,j-1];
  np:=p2+p3+p4+p5+p6+p7+p8+p9;
   begin
    if (((p9<>0) and (p2<>0)) and (((p3<>0) and (p5=0)) and (p6=0))) then
     canvas.Pixels[i,j]:=16777215;
    if (((p9=0) and (p2=0)) and (((p5<>0) and (p6<>0)) and (p7<>0))) then
     canvas.Pixels[i,j]:=16777215;
    if (((p9<>0) and (p8<>0)) and (((p7<>0) and (p3=0)) and (p4=0))) then
     canvas.Pixels[i,j]:=16777215;
    if (((p3<>0) and (p4<>0)) and (((p5<>0) and (p7=0)) and (p8=0))) then
     canvas.Pixels[i,j]:=16777215;
    if (((p2<>0) and (p3<>0)) and (((p4<>0) and (p6=0)) and (p8=0))) then
     canvas.Pixels[i,j]:=16777215;
     if (((p6<>0) and (p7<>0)) and (((p8<>0) and (p2=0)) and (p4=0))) then
     canvas.Pixels[i,j]:=16777215;
     if (((p8<>0) and (p9<>0)) and (((p2<>0) and (p4=0)) and (p6=0))) then
     canvas.Pixels[i,j]:=16777215;
     if (((p4<>0) and (p5<>0)) and (((p6<>0) and (p8=0)) and (p2=0))) then
     canvas.Pixels[i,j]:=16777215;
    end;
 end;

with fin_source2 do
for j:=0 to fin_source2.Height   do
 for i:=0 to fin_source2.Width   do
 begin
  p1:=Canvas.Pixels[i-2,j-2];
  p2:=Canvas.Pixels[i-1,j-2];
  p3:=Canvas.Pixels[i,j-2];
  p4:=Canvas.Pixels[i+1,j-2];
  p5:=Canvas.Pixels[i+2,j-2];
  p6:=Canvas.Pixels[i+2,j-1];
  p7:=Canvas.Pixels[i+2,j];
  p8:=Canvas.Pixels[i+2,j+1];
  p9:=Canvas.Pixels[i+2,j+2];
  p10:=Canvas.Pixels[i+1,j+2];
  p11:=Canvas.Pixels[i,j+2];
  p12:=Canvas.Pixels[i-1,j+2];
  p13:=Canvas.Pixels[i-2,j+2];
  p14:=Canvas.Pixels[i-2,j+1];
  p15:=Canvas.Pixels[i-2,j];
  p16:=Canvas.Pixels[i-2,j-1];
  p17:=Canvas.Pixels[i,j-1];
  p18:=Canvas.Pixels[i+1,j-1];
  p19:=Canvas.Pixels[i+1,j];
  p20:=Canvas.Pixels[i+1,j+1];
```

```
  p21:=Canvas.Pixels[i,j+1];
  p22:=Canvas.Pixels[i-1,j+1];
  p23:=Canvas.Pixels[i-1,j];
  p24:=Canvas.Pixels[i-1,j-1];
 if ((p1<>0)and(p2<>0)and(p3<>0)and(p4<>0)and
 (p5<>0)and(p6<>0)and(p7<>0)and(p8<>0)and(p9<>0)and
 (p10<>0)and(p11<>0)and(p12<>0)and(p13<>0)and(p14<>0)and
 (p15<>0)and(p16<>0)) or
  ((p17<>0)and(p18<>0)and(p19<>0)and(p20<>0)and
 (p21<>0)and(p22<>0)and(p23<>0)and(p24<>0))
  then
       begin
   Canvas.Pixels[i,j-1]:=16777215;
   Canvas.Pixels[i+1,j-1]:=16777215;
   Canvas.Pixels[i+1,j]:=16777215;
   Canvas.Pixels[i+1,j+1]:=16777215;
   Canvas.Pixels[i,j+1]:=16777215;
   Canvas.Pixels[i-1,j+1]:=16777215;
   Canvas.Pixels[i-1,j]:=16777215;
   Canvas.Pixels[i-1,j-1]:=16777215;
   Canvas.Pixels[i,j]:=16777215;
       end;
 end;
  image2.Picture.Assign(fin_source2);
 end;
```

5.Linking Button

```
procedure TForm1.Button12Click(Sender: TObject);
type
link=record
x,y:integer;
key:boolean;
end;
max=record
x,y:integer;
d:extended;
end;
 var
 ar:array[0..2000] of link;
 i,j,cont,k,l,k1,l1 :integer;
 m,pos:max;
 dx,dy,d:extended;
 turn:boolean;
 np,p2,p3,p4,p5,p6,p7,p8,p9:longint;
begin
cont:=0;
with fin_source1 do
 for j:=0 to fin_source1.Height   do
 for i:=0 to fin_source1.Width   do
```

```
 begin
   p2:=Canvas.Pixels[i,j-1];
   p3:=Canvas.Pixels[i+1,j-1];
   p4:=Canvas.Pixels[i+1,j];
   p5:=Canvas.Pixels[i+1,j+1];
   p6:=Canvas.Pixels[i,j+1];
   p7:=Canvas.Pixels[i-1,j+1];
   p8:=Canvas.Pixels[i-1,j];
    p9:=Canvas.Pixels[i-1,j-1];
   np:=p2+p3+p4+p5+p6+p7+p8+p9;
if canvas.Pixels[i,j]=0 then
   begin
   if (((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=16777215))
  or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=0)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=0)))
      then
        begin
          ar[cont].x:=i;
          ar[cont].y:=j;
          ar[cont].key:=false;
          cont:=cont+1;
        end;
   end;
 end;
for i:=0 to cont do
  begin
   m.d:=4;
   for j:=i+1 to cont do
```

```
      begin
      dx:=abs(ar[i].x-ar[j].x);
      dY:=abs(ar[i].Y-ar[j].Y);
      d:=sqrt((sqr(dx)+sqr(dy)));
      if d<m.d then
       begin
        m.x:=ar[j].x;
        m.y:=ar[j].y;
        m.d:=d;
        pos.x:=j;
        turn:=true;
       end; /// for2
      if (m.d<4) and (turn=true)and (ar[i].key=false) and (ar[pos.x].key =false) then
         begin
         fin_source1.canvas.MoveTo(ar[i].x,ar[i].y);
         fin_source1.canvas.LineTo(m.x,m.y);
          turn:=false;
          ar[i].key:=true;
          ar[pos.x].key :=true;
         end;
       end;
cont:=0;
with fin_source2 do
 for j:=0 to fin_source2.Height   do
 for i:=0 to fin_source2.Width   do
  begin
    p2:=Canvas.Pixels[i,j-1];
    p3:=Canvas.Pixels[i+1,j-1];
    p4:=Canvas.Pixels[i+1,j];
    p5:=Canvas.Pixels[i+1,j+1];
    p6:=Canvas.Pixels[i,j+1];
    p7:=Canvas.Pixels[i-1,j+1];
    p8:=Canvas.Pixels[i-1,j];
     p9:=Canvas.Pixels[i-1,j-1];
    np:=p2+p3+p4+p5+p6+p7+p8+p9;
if canvas.Pixels[i,j]=0 then
    begin
    if (((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=16777215))
  or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
```

```
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=0)and(
p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=0)))
      then
        begin
          ar[cont].x:=i;
          ar[cont].y:=j;
          ar[cont].key:=false;
          cont:=cont+1;
        end;
   end;
  end;
for i:=0 to cont do
  begin
   m.d:=4;
   for j:=i+1 to cont do
     begin
     dx:=abs(ar[i].x-ar[j].x);
     dY:=abs(ar[i].Y-ar[j].Y);
     d:=sqrt((sqr(dx)+sqr(dy)));
     if d<m.d then
       begin
        m.x:=ar[j].x;
        m.y:=ar[j].y;
        m.d:=d;
        pos.x:=j;
        turn:=true;
       end;
      end; /// for2
     if (m.d<4) and (turn=true)and (ar[i].key=false) and (ar[pos.x].key =false) then
        begin
        fin_source2.canvas.MoveTo(ar[i].x,ar[i].y);
        fin_source2.canvas.LineTo(m.x,m.y);
         turn:=false;
         ar[i].key:=true;
         ar[pos.x].key :=true;
        end;
      end; ////for
  image1.Picture.Assign(fin_source1);
  image2.Picture.Assign(fin_source2);
```

```
  Button8.Click;
end;
```

6. Matching Button

```
procedure TForm1.Button6Click(Sender: TObject);
type
Fin_table=record
  dist:real;
  angle:real;
  end;
min_pos=record
  x,y:integer;
  end;
var
fin1_axis,fin2_axis:array[1..200]of min_pos;
fin1,fin2:array[1..200]of fin_table;
temp:tbitmap;
off,match:boolean;
st:string;
i,j,k,l,x,y,step1,step2,step3,step4,step5,step6,step7,step8,sp,sp2,fcount1,fcount2:integer;
p:longint;
dx,dy,d:extended;
angle:real;
np,p2,p3,p4,p5,p6,p7,p8,p9:longint;
p21,p31,p41,p51,p61,p71,p81,p91:longint;
begin
button6.Enabled :=false;
temp:=fin_source1;
//// image 1
with fin_source1 do
for j:=0 to fin_source1.Height  do
 for i:=0 to fin_source1.Width  do
  begin
   p2:=Canvas.Pixels[i,j-1];
   p3:=Canvas.Pixels[i+1,j-1];
   p4:=Canvas.Pixels[i+1,j];
   p5:=Canvas.Pixels[i+1,j+1];
   p6:=Canvas.Pixels[i,j+1];
   p7:=Canvas.Pixels[i-1,j+1];
   p8:=Canvas.Pixels[i-1,j];
   p9:=Canvas.Pixels[i-1,j-1];
   np:=p2+p3+p4+p5+p6+p7+p8+p9;
   /// bufurication
 if canvas.Pixels[i,j]=0 then
   begin
  if (((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(
p9=0))
```

```
 or ((p2=0) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=0)and(p9=1677
7215))
 or ((p2=0) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=1677
7215))
 or ((p2=0) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(p9=1677
7215))
 or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=0)and(p9=1677
7215))
 or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=0)and(p9=16777215))
 or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(p9=0))
 or ((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=0)and(p9=1677
7215))
 or ((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=1677
7215))
 or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=0))
 or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=0)and(p9=1677
7215))
 or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=0))
 or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=0))
 or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(
p9=0))
 or ((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=0)and(p9=1677
7215))
 or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=1677
7215)))
 then
 begin
// temp.Canvas.MoveTo(i,j);
// temp.Canvas.Rectangle(i,j,i+3,j+3);
 fin_source1.Canvas.Pixels[i,j]:=clred;
 end;
 /// ridge ending
```

```
if (((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=16777215))
 or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=16777215))
 or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
 or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
 or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
 or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(
p9=16777215))
 or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=0)and(
p9=16777215))
 or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=0)))
 then
   fin_source1.Canvas.Pixels[i,j]:=clyellow ;
end;
end;
 // false minutiea deletion
 step1:=0;
 step2:=0;
 step3:=0;
 step4:=0;
 step5:=0;
 step6:=0;
 step7:=0;
 step8:=0;
 sp:=5;
with fin_source1 do
for j:=0 to fin_source1.Height  do
 for i:=0 to fin_source1.Width  do
 begin
 if fin_source1.Canvas.Pixels[i,j]=clred then
    begin
    step1:=0;
 step2:=0;
 step3:=0;
 step4:=0;
 step5:=0;
```

```
step6:=0;
step7:=0;
step8:=0;
    k:=i;
    l:=j;
 //   while ( fin_source1.Canvas.Pixels[k,l]<>clgreen )do
      begin
       p2:=Canvas.Pixels[k,l-1];
       p3:=Canvas.Pixels[k+1,l-1];
       p4:=Canvas.Pixels[k+1,l];
       p5:=Canvas.Pixels[k+1,l+1];
       p6:=Canvas.Pixels[k,l+1];
       p7:=Canvas.Pixels[k-1,l+1];
       p8:=Canvas.Pixels[k-1,l];
       p9:=Canvas.Pixels[k-1,l-1];
        if p2=0 then
          begin
          x:=k;
          y:=l-1;
          while(off<>true) do
             begin
               step1:=step1+1;
               if step1=sp then
                 off:=true;
               fin_source1.Canvas.Pixels[x,y]:=clgreen;
               p21:=Canvas.Pixels[x,y-1];
               p31:=Canvas.Pixels[x+1,y-1];
               p41:=Canvas.Pixels[x+1,y];
               p51:=Canvas.Pixels[x+1,y+1];
               p61:=Canvas.Pixels[x,y+1];
               p71:=Canvas.Pixels[x-1,y+1];
               p81:=Canvas.Pixels[x-1,y];
               p91:=Canvas.Pixels[x-1,y-1];
                if p21=0 then
                  begin
                  y:=y-1;
                  end
                  else
                if p31=0 then
                  begin
                  x:=x+1;
                  y:=y-1;
                  end
                  else
                if p41=0 then
                  begin
                  x:=x+1;
                  end
                  else
```

```
        if p51=0 then
          begin
          x:=x+1;
          y:=y+1;
          end
          else
        if p61=0 then
          begin
          y:=y+1;
          end
          else
        if p71=0 then
          begin
          x:=x-1;
          y:=y+1;
          end
          else
        if p81=0 then
          begin
          x:=x-1;
          end
          else
        if p91=0 then
          begin
          x:=x-1;
          y:=y-1;
          end
          else
          off:=true;
    end; //while p2
    off:=false;
    k:=i;
    l:=j;
  end;  //if p2
if p3=0 then
  begin
 x:=k+1;
 y:=l-1;
 while(off<>true)do
    begin
      step2:=step2+1;
      if step2=sp then
       off:=true;
      fin_source1.Canvas.Pixels[x,y]:=clgreen;
      p21:=Canvas.Pixels[x,y-1];
      p31:=Canvas.Pixels[x+1,y-1];
      p41:=Canvas.Pixels[x+1,y];
      p51:=Canvas.Pixels[x+1,y+1];
      p61:=Canvas.Pixels[x,y+1];
```

```
  p71:=Canvas.Pixels[x-1,y+1];
  p81:=Canvas.Pixels[x-1,y];
  p91:=Canvas.Pixels[x-1,y-1];
   if p21=0 then
     begin
     y:=y-1;
     end
     else
   if p31=0 then
     begin
     x:=x+1;
     y:=y-1;
     end
     else
   if p41=0 then
     begin
     x:=x+1;
     end
     else
   if p51=0 then
     begin
     x:=x+1;
     y:=y+1;
     end
     else
   if p61=0 then
     begin
     y:=y+1;
     end
     else
   if p71=0 then
     begin
     x:=x-1;
     y:=y+1;
     end
     else
   if p81=0 then
     begin
     x:=x-1;
     end
     else
   if p91=0 then
     begin
     x:=x-1;
     y:=y-1;
     end
     else
     off:=true;
end; //while p2
```

```
    off:=false;
    k:=i;
    l:=j;
   end; //if p3
if p4=0 then
   begin
    x:=k+1;
  y:=l;
  while(off<>true) do
     begin
       step3:=step3+1;
       if step3=sp then
        off:=true;
       fin_source1.Canvas.Pixels[x,y]:=clgreen;
       p21:=Canvas.Pixels[x,y-1];
       p31:=Canvas.Pixels[x+1,y-1];
       p41:=Canvas.Pixels[x+1,y];
       p51:=Canvas.Pixels[x+1,y+1];
       p61:=Canvas.Pixels[x,y+1];
       p71:=Canvas.Pixels[x-1,y+1];
       p81:=Canvas.Pixels[x-1,y];
       p91:=Canvas.Pixels[x-1,y-1];
        if p21=0 then
          begin
          y:=y-1;
          end
          else
        if p31=0 then
          begin
          x:=x+1;
          y:=y-1;
          end
          else
        if p41=0 then
          begin
          x:=x+1;
          end
          else
        if p51=0 then
          begin
          x:=x+1;
          y:=y+1;
          end
          else
        if p61=0 then
          begin
          y:=y+1;
          end
          else
```

```
          if p71=0 then
            begin
            x:=x-1;
            y:=y+1;
            end
            else
          if p81=0 then
            begin
            x:=x-1;
            end
            else
          if p91=0 then
            begin
            x:=x-1;
            y:=y-1;
            end
            else
            off:=true;
      end; //while p2
       off:=false;
       k:=i;
       l:=j;
    end;
  if p5=0 then
     begin
      x:=k+1;
      y:=l+1;
   while(off<>true) do
      begin
        step4:=step4+1;
        if step4=sp then
         off:=true;
        fin_source1.Canvas.Pixels[x,y]:=clgreen;
        p21:=Canvas.Pixels[x,y-1];
        p31:=Canvas.Pixels[x+1,y-1];
        p41:=Canvas.Pixels[x+1,y];
        p51:=Canvas.Pixels[x+1,y+1];
        p61:=Canvas.Pixels[x,y+1];
        p71:=Canvas.Pixels[x-1,y+1];
        p81:=Canvas.Pixels[x-1,y];
        p91:=Canvas.Pixels[x-1,y-1];
         if p21=0 then
            begin
            y:=y-1;
            end
            else
          if p31=0 then
            begin
            x:=x+1;
```

```
            y:=y-1;
            end
            else
          if p41=0 then
            begin
            x:=x+1;
            end
            else
          if p51=0 then
            begin
            x:=x+1;
            y:=y+1;
            end
            else
          if p61=0 then
            begin
            y:=y+1;
            end
            else
          if p71=0 then
            begin
            x:=x-1;
            y:=y+1;
            end
            else
          if p81=0 then
            begin
            x:=x-1;
            end
            else
          if p91=0 then
            begin
            x:=x-1;
            y:=y-1;
            end
            else
            off:=true;
      end; //while p2
       off:=false;
       k:=i;
       l:=j;
      end;
if p6=0 then
  begin
    x:=k;
 y:=l+1;
 while(off<>true) do
    begin
      step5:=step5+1;
```

```
if step5=sp then
 off:=true;
 fin_source1.Canvas.Pixels[x,y]:=clgreen;
p21:=Canvas.Pixels[x,y-1];
p31:=Canvas.Pixels[x+1,y-1];
p41:=Canvas.Pixels[x+1,y];
p51:=Canvas.Pixels[x+1,y+1];
p61:=Canvas.Pixels[x,y+1];
p71:=Canvas.Pixels[x-1,y+1];
p81:=Canvas.Pixels[x-1,y];
p91:=Canvas.Pixels[x-1,y-1];
 if p21=0 then
   begin
   y:=y-1;
   end
   else
 if p31=0 then
   begin
   x:=x+1;
   y:=y-1;
   end
   else
 if p41=0 then
   begin
   x:=x+1;
   end
   else
 if p51=0 then
   begin
   x:=x+1;
   y:=y+1;
   end
   else
 if p61=0 then
   begin
   y:=y+1;
   end
   else
 if p71=0 then
   begin
   x:=x-1;
   y:=y+1;
   end
   else
 if p81=0 then
   begin
   x:=x-1;
   end
   else
```

```
          if p91=0 then
            begin
            x:=x-1;
            y:=y-1;
            end
            else
            off:=true;
      end; //while p2
       off:=false;
       k:=i;
       l:=j;
   end;
if p7=0 then
    begin
      x:=k-1;
  y:=l+1;
  while(off<>true) do
      begin
        step6:=step6+1;
        if step6=sp then
          off:=true;
         fin_source1.Canvas.Pixels[x,y]:=clgreen;
        p21:=Canvas.Pixels[x,y-1];
        p31:=Canvas.Pixels[x+1,y-1];
        p41:=Canvas.Pixels[x+1,y];
        p51:=Canvas.Pixels[x+1,y+1];
        p61:=Canvas.Pixels[x,y+1];
        p71:=Canvas.Pixels[x-1,y+1];
        p81:=Canvas.Pixels[x-1,y];
        p91:=Canvas.Pixels[x-1,y-1];
          if p21=0 then
            begin
            y:=y-1;
            end
            else
        if p31=0 then
            begin
            x:=x+1;
            y:=y-1;
            end
            else
        if p41=0 then
            begin
            x:=x+1;
            end
            else
        if p51=0 then
            begin
            x:=x+1;
```

```
          y:=y+1;
          end
          else
        if p61=0 then
          begin
          y:=y+1;
          end
          else
        if p71=0 then
          begin
          x:=x-1;
          y:=y+1;
          end
          else
        if p81=0 then
          begin
          x:=x-1;
          end
          else
        if p91=0 then
          begin
          x:=x-1;
          y:=y-1;
          end
          else
          off:=true;

    end; //while p2
     off:=false;
     k:=i;
     l:=j;
   end;
if p8=0 then
   begin
    x:=k-1;
  y:=l;
  while(off<>true) do
    begin
      step7:=step7+1;
      if step7=sp then
       off:=true;
      fin_source1.Canvas.Pixels[x,y]:=clgreen;
      p21:=Canvas.Pixels[x,y-1];
      p31:=Canvas.Pixels[x+1,y-1];
      p41:=Canvas.Pixels[x+1,y];
      p51:=Canvas.Pixels[x+1,y+1];
      p61:=Canvas.Pixels[x,y+1];
      p71:=Canvas.Pixels[x-1,y+1];
      p81:=Canvas.Pixels[x-1,y];
```

```
  p91:=Canvas.Pixels[x-1,y-1];
   if p21=0 then
     begin
     y:=y-1;
     end
     else
   if p31=0 then
     begin
     x:=x+1;
     y:=y-1;
     end
     else
   if p41=0 then
     begin
     x:=x+1;
     end
     else
   if p51=0 then
     begin
     x:=x+1;
     y:=y+1;
     end
     else
   if p61=0 then
     begin
     y:=y+1;
     end
     else
   if p71=0 then
     begin
     x:=x-1;
     y:=y+1;
     end
     else
   if p81=0 then
     begin
     x:=x-1;
     end
     else
   if p91=0 then
     begin
     x:=x-1;
     y:=y-1;
     end
     else
     off:=true;

end; //while p2
 off:=false;
```

```
      k:=i;
      l:=j;
    end;
if p9=0 then
 begin
  x:=k-1;
  y:=l-1;
 while(off<>true) do
    begin
     step8:=step8+1;
     if step8=sp then
      off:=true;
      fin_source1.Canvas.Pixels[x,y]:=clgreen;
      p21:=Canvas.Pixels[x,y-1];
      p31:=Canvas.Pixels[x+1,y-1];
      p41:=Canvas.Pixels[x+1,y];
      p51:=Canvas.Pixels[x+1,y+1];
      p61:=Canvas.Pixels[x,y+1];
      p71:=Canvas.Pixels[x-1,y+1];
      p81:=Canvas.Pixels[x-1,y];
      p91:=Canvas.Pixels[x-1,y-1];
       if p21=0 then
         begin
         y:=y-1;
         end
         else
       if p31=0 then
         begin
         x:=x+1;
         y:=y-1;
         end
         else
       if p41=0 then
         begin
         x:=x+1;
         end
         else
       if p51=0 then
         begin
         x:=x+1;
         y:=y+1;
         end
         else
       if p61=0 then
         begin
         y:=y+1;
         end
         else
       if p71=0 then
```

```
                    begin
                    x:=x-1;
                    y:=y+1;
                    end
                    else
                  if p81=0 then
                    begin
                    x:=x-1;
                    end
                    else
                  if p91=0 then
                    begin
                    x:=x-1;
                    y:=y-1;
                    end
                    else
                    off:=true;
                end; //while p2-
                 off:=false;
                 k:=i;
                 l:=j;
              end;  //if p9
          end;
          if step1+step2+step3+step4+step5+step6+step7+step8=sp*3 then
           fin_source1.canvas.Pixels[i,j]:=clblue;
      end;
   end;
   // clear red and green 16240293
with fin_source1 do
 for j:=0 to fin_source1.Height   do
 for i:=0 to fin_source1.Width   do
    begin
if  (fin_source1.Canvas.Pixels[i,j]<>clred) and
(fin_source1.Canvas.Pixels[i,j]<>clwhite) and
(fin_source1.Canvas.Pixels[i,j]<>clyellow) and
(fin_source1.Canvas.Pixels[i,j]<>16240293) and
(fin_source1.Canvas.Pixels[i,j]<>clblue)  then
    fin_source1.Canvas.Pixels[i,j]:=0;
     end;
//// test ridge ending
sp2:=10;
  off:=false;
  step1:=0;
with fin_source1 do
 for j:=0 to fin_source1.Height   do
 for i:=0 to fin_source1.Width   do
    begin
      if Canvas.Pixels[i,j]=clyellow then
        begin
```

```
step1:=0;
x:=i;
y:=j;
while(off<>true) do
        begin
          step1:=step1+1;
          if step1=sp2 then
           off:=true;
          fin_source1.Canvas.Pixels[x,y]:=clgreen;
          p21:=Canvas.Pixels[x,y-1];
          p31:=Canvas.Pixels[x+1,y-1];
          p41:=Canvas.Pixels[x+1,y];
          p51:=Canvas.Pixels[x+1,y+1];
          p61:=Canvas.Pixels[x,y+1];
          p71:=Canvas.Pixels[x-1,y+1];
          p81:=Canvas.Pixels[x-1,y];
          p91:=Canvas.Pixels[x-1,y-1];
           if p21=0 then
             begin
             y:=y-1;
             end
             else
           if p31=0 then
             begin
             x:=x+1;
             y:=y-1;
             end
             else
           if p41=0 then
             begin
             x:=x+1;
             end
             else
           if p51=0 then
             begin
             x:=x+1;
             y:=y+1;
             end
             else
           if p61=0 then
             begin
             y:=y+1;
             end
             else
           if p71=0 then
             begin
             x:=x-1;
             y:=y+1;
             end
```

```
                else
              if p81=0 then
                begin
                x:=x-1;
                end
                else
              if p91=0 then
                begin
                x:=x-1;
                y:=y-1;
                end
                else
                off:=true;
              if ((p21=clred) or (p21=clblue)) or ((p31=clred) or (p31=clblue)) or
              ((p41=clred) or (p41=clblue)) or ((p51=clred) or (p51=clblue)) or
              ((p61=clred) or (p61=clblue)) or  ((p71=clred) or (p71=clblue)) or
              ((p81=clred) or (p81=clblue)) or ((p91=clred) or (p91=clblue)) then
               begin
               off:=true;
               step1:=0;
               end;

           end; //while p2
           off:=false;
          if step1=sp2 then
            fin_source1.Canvas.Pixels[i,j]:=clblue;
          end;  //if p2
  end;
  // clear red and green 16240293
with fin_source1 do
 for j:=0 to fin_source1.Height   do
 for i:=0 to fin_source1.Width   do
   begin
if (fin_source1.Canvas.Pixels[i,j]<>clwhite) and
(fin_source1.Canvas.Pixels[i,j]<>16240293) and
(fin_source1.Canvas.Pixels[i,j]<>clblue)  then
   fin_source1.Canvas.Pixels[i,j]:=0;
    end;
 image1.Picture.Assign(fin_source1);
with fin_source2 do
for j:=0 to fin_source2.Height   do
 for i:=0 to fin_source2.Width   do
 begin
  p2:=Canvas.Pixels[i,j-1];
  p3:=Canvas.Pixels[i+1,j-1];
  p4:=Canvas.Pixels[i+1,j];
  p5:=Canvas.Pixels[i+1,j+1];
  p6:=Canvas.Pixels[i,j+1];
  p7:=Canvas.Pixels[i-1,j+1];
```

```
   p8:=Canvas.Pixels[i-1,j];
   p9:=Canvas.Pixels[i-1,j-1];
   np:=p2+p3+p4+p5+p6+p7+p8+p9;
   /// bufurication
 if canvas.Pixels[i,j]=0 then
   begin
  if (((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(
p9=0))
  or ((p2=0) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=0)and(p9=1677
7215))
  or ((p2=0) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=1677
7215))
  or ((p2=0) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(p9=1677
7215))
  or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=0)and(p9=1677
7215))
  or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=0)and(p9=16777215))
  or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(p9=0))
  or ((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=0)and(p9=1677
7215))
  or ((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=1677
7215))
  or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=0))
  or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=0)and(p9=1677
7215))
  or ((p2=16777215) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=0))
  or ((p2=16777215) and (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=0))
  or ((p2=16777215) and (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(
p9=0))
  or ((p2=0) and (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=0)and(p9=1677
7215))
```

```
 or                ((p2=16777215)          and          (p3=0)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=0)and(p8=16777215)and(p9=1677
7215)))
 then
 begin
// temp.Canvas.MoveTo(i,j);
// temp.Canvas.Rectangle(i,j,i+3,j+3);
 fin_source2.Canvas.Pixels[i,j]:=clred;
 end;
 /// ridge ending
if                (((p2=0)          and          (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=16777215))
 or                ((p2=16777215)          and          (p3=0)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=16777215))
 or                ((p2=16777215)          and          (p3=16777215)and
(p4=0)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
 or                ((p2=16777215)          and          (p3=16777215)and
(p4=16777215)and(p5=0)and(p6=16777215)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
 or                ((p2=16777215)          and          (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=0)and(p7=16777215)and(p8=16777215)and(
p9=16777215))
 or                ((p2=16777215)          and          (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=0)and(p8=16777215)and(
p9=16777215))
 or                ((p2=16777215)          and          (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=0)and(
p9=16777215))
 or                ((p2=16777215)          and          (p3=16777215)and
(p4=16777215)and(p5=16777215)and(p6=16777215)and(p7=16777215)and(p8=16777
215)and(p9=0)))
 then
   fin_source2.Canvas.Pixels[i,j]:=clyellow ;
end;
end;
 // false minutiea deletion
 step1:=0;
 step2:=0;
 step3:=0;
 step4:=0;
 step5:=0;
 step6:=0;
 step7:=0;
 step8:=0;
 sp:=5;
with fin_source2 do
```

```
for j:=0 to fin_source2.Height   do
 for i:=0 to fin_source2.Width   do
  begin
  if fin_source2.Canvas.Pixels[i,j]=clred then
     begin
     step1:=0;
  step2:=0;
  step3:=0;
  step4:=0;
  step5:=0;
  step6:=0;
  step7:=0;
  step8:=0;
       k:=i;
       l:=j;
   //    while ( fin_source1.Canvas.Pixels[k,l]<>clgreen )do
         begin
           p2:=Canvas.Pixels[k,l-1];
           p3:=Canvas.Pixels[k+1,l-1];
           p4:=Canvas.Pixels[k+1,l];
           p5:=Canvas.Pixels[k+1,l+1];
           p6:=Canvas.Pixels[k,l+1];
           p7:=Canvas.Pixels[k-1,l+1];
           p8:=Canvas.Pixels[k-1,l];
           p9:=Canvas.Pixels[k-1,l-1];
            if p2=0 then
              begin
              x:=k;
              y:=l-1;
              while(off<>true) do
                 begin
                   step1:=step1+1;
                   if step1=sp then
                    off:=true;
                   fin_source2.Canvas.Pixels[x,y]:=clgreen;
                   p21:=Canvas.Pixels[x,y-1];
                   p31:=Canvas.Pixels[x+1,y-1];
                   p41:=Canvas.Pixels[x+1,y];
                   p51:=Canvas.Pixels[x+1,y+1];
                   p61:=Canvas.Pixels[x,y+1];
                   p71:=Canvas.Pixels[x-1,y+1];
                   p81:=Canvas.Pixels[x-1,y];
                   p91:=Canvas.Pixels[x-1,y-1];
                    if p21=0 then
                      begin
                      y:=y-1;
                      end
                      else
```

```
        if p31=0 then
          begin
          x:=x+1;
          y:=y-1;
          end
          else
        if p41=0 then
          begin
          x:=x+1;
          end
          else
        if p51=0 then
          begin
          x:=x+1;
          y:=y+1;
          end
          else
        if p61=0 then
          begin
          y:=y+1;
          end
          else
        if p71=0 then
          begin
          x:=x-1;
          y:=y+1;
          end
          else
        if p81=0 then
          begin
          x:=x-1;
          end
          else
        if p91=0 then
          begin
          x:=x-1;
          y:=y-1;
          end
          else
          off:=true;
    end; //while p2
    off:=false;
    k:=i;
    l:=j;
  end;  //if p2
if p3=0 then
  begin
 x:=k+1;
 y:=l-1;
```

```
while(off<>true)do
  begin
    step2:=step2+1;
    if step2=sp then
     off:=true;
     fin_source2.Canvas.Pixels[x,y]:=clgreen;
    p21:=Canvas.Pixels[x,y-1];
    p31:=Canvas.Pixels[x+1,y-1];
    p41:=Canvas.Pixels[x+1,y];
    p51:=Canvas.Pixels[x+1,y+1];
    p61:=Canvas.Pixels[x,y+1];
    p71:=Canvas.Pixels[x-1,y+1];
    p81:=Canvas.Pixels[x-1,y];
    p91:=Canvas.Pixels[x-1,y-1];
     if p21=0 then
       begin
       y:=y-1;
       end
       else
     if p31=0 then
       begin
       x:=x+1;
       y:=y-1;
       end
       else
     if p41=0 then
       begin
       x:=x+1;
       end
       else
     if p51=0 then
       begin
       x:=x+1;
       y:=y+1;
       end
       else
     if p61=0 then
       begin
       y:=y+1;
       end
       else
     if p71=0 then
       begin
       x:=x-1;
       y:=y+1;
       end
       else
     if p81=0 then
       begin
```

```
            x:=x-1;
            end
            else
          if p91=0 then
            begin
            x:=x-1;
            y:=y-1;
            end
            else
            off:=true;
      end; //while p2
     off:=false;
     k:=i;
     l:=j;
   end; //if p3
if p4=0 then
  begin
    x:=k+1;
 y:=l;
 while(off<>true) do
    begin
      step3:=step3+1;
      if step3=sp then
       off:=true;
       fin_source2.Canvas.Pixels[x,y]:=clgreen;
      p21:=Canvas.Pixels[x,y-1];
      p31:=Canvas.Pixels[x+1,y-1];
      p41:=Canvas.Pixels[x+1,y];
      p51:=Canvas.Pixels[x+1,y+1];
      p61:=Canvas.Pixels[x,y+1];
      p71:=Canvas.Pixels[x-1,y+1];
      p81:=Canvas.Pixels[x-1,y];
      p91:=Canvas.Pixels[x-1,y-1];
       if p21=0 then
         begin
         y:=y-1;
         end
         else
       if p31=0 then
         begin
         x:=x+1;
         y:=y-1;
         end
         else
       if p41=0 then
         begin
         x:=x+1;
         end
         else
```

```
          if p51=0 then
            begin
            x:=x+1;
            y:=y+1;
            end
            else
          if p61=0 then
            begin
            y:=y+1;
            end
            else
          if p71=0 then
            begin
            x:=x-1;
            y:=y+1;
            end
            else
          if p81=0 then
            begin
            x:=x-1;
            end
            else
          if p91=0 then
            begin
            x:=x-1;
            y:=y-1;
            end
            else
            off:=true;

      end; //while p2
       off:=false;
       k:=i;
       l:=j;
    end;
  if p5=0 then
     begin
      x:=k+1;
      y:=l+1;
   while(off<>true) do
      begin
        step4:=step4+1;
        if step4=sp then
         off:=true;
        fin_source2.Canvas.Pixels[x,y]:=clgreen;
        p21:=Canvas.Pixels[x,y-1];
        p31:=Canvas.Pixels[x+1,y-1];
        p41:=Canvas.Pixels[x+1,y];
        p51:=Canvas.Pixels[x+1,y+1];
```

```
p61:=Canvas.Pixels[x,y+1];
p71:=Canvas.Pixels[x-1,y+1];
p81:=Canvas.Pixels[x-1,y];
p91:=Canvas.Pixels[x-1,y-1];
 if p21=0 then
   begin
   y:=y-1;
   end
   else
 if p31=0 then
   begin
   x:=x+1;
   y:=y-1;
   end
   else
 if p41=0 then
   begin
   x:=x+1;
   end
   else
 if p51=0 then
   begin
   x:=x+1;
   y:=y+1;
   end
   else
 if p61=0 then
   begin
   y:=y+1;
   end
   else
 if p71=0 then
   begin
   x:=x-1;
   y:=y+1;
   end
   else
 if p81=0 then
   begin
   x:=x-1;
   end
   else
 if p91=0 then
   begin
   x:=x-1;
   y:=y-1;
   end
   else
   off:=true;
```

```
      end; //while p2
       off:=false;
       k:=i;
       l:=j;
      end;
if p6=0 then
  begin
    x:=k;
 y:=l+1;
 while(off<>true) do
    begin
      step5:=step5+1;
      if step5=sp then
        off:=true;
       fin_source2.Canvas.Pixels[x,y]:=clgreen;
      p21:=Canvas.Pixels[x,y-1];
      p31:=Canvas.Pixels[x+1,y-1];
      p41:=Canvas.Pixels[x+1,y];
      p51:=Canvas.Pixels[x+1,y+1];
      p61:=Canvas.Pixels[x,y+1];
      p71:=Canvas.Pixels[x-1,y+1];
      p81:=Canvas.Pixels[x-1,y];
      p91:=Canvas.Pixels[x-1,y-1];
        if p21=0 then
          begin
          y:=y-1;
          end
          else
        if p31=0 then
          begin
          x:=x+1;
          y:=y-1;
          end
          else
        if p41=0 then
          begin
          x:=x+1;
          end
          else
        if p51=0 then
          begin
          x:=x+1;
          y:=y+1;
          end
          else
        if p61=0 then
          begin
          y:=y+1;
          end
```

```
          else
        if p71=0 then
          begin
          x:=x-1;
          y:=y+1;
          end
          else
        if p81=0 then
          begin
          x:=x-1;
          end
          else
        if p91=0 then
          begin
          x:=x-1;
          y:=y-1;
          end
          else
          off:=true;
    end; //while p2
     off:=false;
     k:=i;
     l:=j;
  end;
if p7=0 then
   begin
     x:=k-1;
 y:=l+1;
 while(off<>true) do
    begin
      step6:=step6+1;
      if step6=sp then
       off:=true;
       fin_source2.Canvas.Pixels[x,y]:=clgreen;
      p21:=Canvas.Pixels[x,y-1];
      p31:=Canvas.Pixels[x+1,y-1];
      p41:=Canvas.Pixels[x+1,y];
      p51:=Canvas.Pixels[x+1,y+1];
      p61:=Canvas.Pixels[x,y+1];
      p71:=Canvas.Pixels[x-1,y+1];
      p81:=Canvas.Pixels[x-1,y];
      p91:=Canvas.Pixels[x-1,y-1];
       if p21=0 then
          begin
          y:=y-1;
          end
          else
        if p31=0 then
          begin
```

```
            x:=x+1;
            y:=y-1;
            end
            else
          if p41=0 then
            begin
            x:=x+1;
            end
            else
          if p51=0 then
            begin
            x:=x+1;
            y:=y+1;
            end
            else
          if p61=0 then
            begin
            y:=y+1;
            end
            else
          if p71=0 then
            begin
            x:=x-1;
            y:=y+1;
            end
            else
          if p81=0 then
            begin
            x:=x-1;
            end
            else
          if p91=0 then
            begin
            x:=x-1;
            y:=y-1;
            end
            else
            off:=true;
      end; //while p2
       off:=false;
       k:=i;
       l:=j;
    end;
if p8=0 then
   begin
     x:=k-1;
  y:=l;
  while(off<>true) do
     begin
```

```
step7:=step7+1;
if step7=sp then
 off:=true;
 fin_source2.Canvas.Pixels[x,y]:=clgreen;
p21:=Canvas.Pixels[x,y-1];
p31:=Canvas.Pixels[x+1,y-1];
p41:=Canvas.Pixels[x+1,y];
p51:=Canvas.Pixels[x+1,y+1];
p61:=Canvas.Pixels[x,y+1];
p71:=Canvas.Pixels[x-1,y+1];
p81:=Canvas.Pixels[x-1,y];
p91:=Canvas.Pixels[x-1,y-1];
 if p21=0 then
   begin
   y:=y-1;
   end
   else
 if p31=0 then
   begin
   x:=x+1;
   y:=y-1;
   end
   else
 if p41=0 then
   begin
   x:=x+1;
   end
   else
 if p51=0 then
   begin
   x:=x+1;
   y:=y+1;
   end
   else
 if p61=0 then
   begin
   y:=y+1;
   end
   else
 if p71=0 then
   begin
   x:=x-1;
   y:=y+1;
   end
   else
 if p81=0 then
   begin
   x:=x-1;
   end
```

```
          else
         if p91=0 then
           begin
           x:=x-1;
           y:=y-1;
           end
           else
           off:=true;
     end; //while p2
      off:=false;
      k:=i;
      l:=j;
    end;
if p9=0 then
  begin
    x:=k-1;
    y:=l-1;
  while(off<>true) do
     begin
      step8:=step8+1;
      if step8=sp then
       off:=true;
       fin_source2.Canvas.Pixels[x,y]:=clgreen;
       p21:=Canvas.Pixels[x,y-1];
       p31:=Canvas.Pixels[x+1,y-1];
       p41:=Canvas.Pixels[x+1,y];
       p51:=Canvas.Pixels[x+1,y+1];
       p61:=Canvas.Pixels[x,y+1];
       p71:=Canvas.Pixels[x-1,y+1];
       p81:=Canvas.Pixels[x-1,y];
       p91:=Canvas.Pixels[x-1,y-1];
         if p21=0 then
           begin
           y:=y-1;
           end
           else
         if p31=0 then
           begin
           x:=x+1;
           y:=y-1;
           end
           else
         if p41=0 then
           begin
           x:=x+1;
           end
           else
         if p51=0 then
           begin
```

```
                    x:=x+1;
                    y:=y+1;
                    end
                    else
                  if p61=0 then
                    begin
                    y:=y+1;
                    end
                    else
                  if p71=0 then
                    begin
                    x:=x-1;
                    y:=y+1;
                    end
                    else
                  if p81=0 then
                    begin
                    x:=x-1;
                    end
                    else
                  if p91=0 then
                    begin
                    x:=x-1;
                    y:=y-1;
                    end
                    else
                    off:=true;
                end; //while p2-
                 off:=false;
                 k:=i;
                 l:=j;
              end;  //if p9
          end;
          if step1+step2+step3+step4+step5+step6+step7+step8=sp*3 then
           fin_source2.canvas.Pixels[i,j]:=clblue;
      end;
   end;
   // clear red and green 16240293
  with fin_source2 do
   for j:=0 to fin_source2.Height   do
   for i:=0 to fin_source2.Width   do
     begin
  if  (fin_source2.Canvas.Pixels[i,j]<>clred) and
  (fin_source2.Canvas.Pixels[i,j]<>clwhite) and
  (fin_source2.Canvas.Pixels[i,j]<>clyellow) and
  (fin_source2.Canvas.Pixels[i,j]<>16240293) and
  (fin_source2.Canvas.Pixels[i,j]<>clblue)  then
     fin_source2.Canvas.Pixels[i,j]:=0;
      end;
```

```
sp2:=10;
 off:=false;
 step1:=0;
with fin_source2 do
 for j:=0 to fin_source2.Height   do
 for i:=0 to fin_source2.Width   do
    begin
     if Canvas.Pixels[i,j]=clyellow then
       begin
       step1:=0;
       x:=i;
       y:=j;
       while(off<>true) do
              begin
                step1:=step1+1;
                if step1=sp2 then
                 off:=true;
                fin_source2.Canvas.Pixels[x,y]:=clgreen;
                p21:=Canvas.Pixels[x,y-1];
                p31:=Canvas.Pixels[x+1,y-1];
                p41:=Canvas.Pixels[x+1,y];
                p51:=Canvas.Pixels[x+1,y+1];
                p61:=Canvas.Pixels[x,y+1];
                p71:=Canvas.Pixels[x-1,y+1];
                p81:=Canvas.Pixels[x-1,y];
                p91:=Canvas.Pixels[x-1,y-1];
                 if p21=0 then
                   begin
                   y:=y-1;
                   end
                   else
                 if p31=0 then
                   begin
                   x:=x+1;
                   y:=y-1;
                   end
                   else
                 if p41=0 then
                   begin
                   x:=x+1;
                   end
                   else
                 if p51=0 then
                   begin
                   x:=x+1;
                   y:=y+1;
                   end
                   else
                 if p61=0 then
```

```
                    begin
                    y:=y+1;
                    end
                    else
                  if p71=0 then
                    begin
                    x:=x-1;
                    y:=y+1;
                    end
                    else
                  if p81=0 then
                    begin
                    x:=x-1;
                    end
                    else
                  if p91=0 then
                    begin
                    x:=x-1;
                    y:=y-1;
                    end
                    else
                    off:=true;
                  if ((p21=clred) or (p21=clblue)) or ((p31=clred) or (p31=clblue)) or
                  ((p41=clred) or (p41=clblue)) or ((p51=clred) or (p51=clblue)) or
                  ((p61=clred) or (p61=clblue)) or  ((p71=clred) or (p71=clblue)) or
                  ((p81=clred) or (p81=clblue)) or ((p91=clred) or (p91=clblue)) then
                    begin
                    off:=true;
                    step1:=0;
                    end;
                end; //while p2
                off:=false;
               if step1=sp2 then
                 fin_source2.Canvas.Pixels[i,j]:=clblue;
               end;  //if p2
    end;
with fin_source2 do
 for j:=0 to fin_source2.Height   do
 for i:=0 to fin_source2.Width   do
    begin
if (fin_source2.Canvas.Pixels[i,j]<>clwhite) and
(fin_source2.Canvas.Pixels[i,j]<>16240293) and
(fin_source2.Canvas.Pixels[i,j]<>clblue)  then
    fin_source2.Canvas.Pixels[i,j]:=0;
    end;

 image2.Picture.Assign(fin_source2);
fcount1:=0;
 fcount2:=0;
```

```
 with fin_source1 do
for j:=0 to fin_source1.Height   do
 for i:=0 to fin_source1.Width   do
  begin
  if fin_source1.Canvas.Pixels[i,j]=clblue then
     begin
      fcount1:=fcount1+1;
      fin1_axis[fcount1].x:=i;
      fin1_axis[fcount1].y:=j;
      end;
  end;
  str(fcount1,st);
  edit2.Text :=st;
  /// number of minutiea finger2
  with fin_source2 do
for j:=0 to fin_source2.Height   do
 for i:=0 to fin_source2.Width   do
  begin
  if fin_source2.Canvas.Pixels[i,j]=clblue then
     begin
      fcount2:=fcount2+1;
      fin2_axis[fcount2].x:=i;
      fin2_axis[fcount2].y:=j;
      end;
  end;
  str(fcount2,st);
  edit3.Text :=st;
for i:=1 to fcount1-1   do
  begin
      dx:=abs(fin1_axis[i].x-fin1_axis[i+1].x);
      dY:=abs(fin1_axis[i].Y-fin1_axis[i].Y);
      d:=sqrt((sqr(dx)+sqr(dy)));
    angle:= arcsin(dy/d);
     fin1[i].dist:=d;
     fin1[i].angle:=angle;
  end;
for i:=1 to fcount2-1   do
  begin
      dx:=abs(fin2_axis[i].x-fin2_axis[i+1].x);
      dY:=abs(fin2_axis[i].Y-fin2_axis[i].Y);
      d:=sqrt((sqr(dx)+sqr(dy)));
     angle:= arcsin(dy/d);
     fin2[i].dist:=d;
     fin2[i].angle:=angle;
  end;
  match:=false;
  step1:=0;
 for i:=1 to fcount1 do
   for j:=1 to fcount2 do
```

```
    begin
      if ((fin1[i].dist<=fin2[j].dist+3) and (fin1[i].dist>=fin2[j].dist-3) ) and (match=false)
then
          begin
           k:=i;
           l:=j;
             while      (step1<>tol-1)      and      ((fin1[k].dist<=fin2[l].dist+3)      and
(fin1[k].dist>=fin2[l].dist-3) )do
                 begin
                  k:=k+1;
                  l:=l+1;
                  step1:=step1+1;
                  if (fin1[k].dist<=fin2[l].dist+3) and (fin1[k].dist>=fin2[l].dist-3) then
                    match:=true
                    else
                    match:=false;
                 end;
          end;
     end;
     if match=true then
       edit1.Text :='MATCHING'
       else
       edit1.Text :='NOT MATCHING'
end;
```