The development of an accurate MOC algorithm without the computation of a core reference point

KP Baruni

21508259

Dissertation submitted in fulfilment of the requirements for the degree *Master of Science* in Computer and Electronic Engineering at the Potchefstroom Campus of the North-West University

Supervisor: Prof ASJ Helberg

Co-Supervisor: Johan Van der Merwe

Assistant Supervisor: Dr K Nair

October 2017

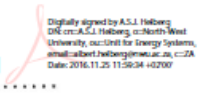
Declaration

I declare that this research titled “The development of an accurate fingerprint Match-on-Card (MoC) algorithm without the computation of the core reference point" is my original research work, and it has never been presented by others. All the information used in this work has been fully acknowledged both in text and in the references.

K P Baruni

24 November 2016

Date

Supervisor 

Prof. ASJ Helberg

2016/11/25

Date

Editor's certificate

NORTH-WEST UNIVERSITY
YUNIBESITI YA BOKONE-BOPHIRIMA
NOORDWES-UNIVERSITEIT
MAFIKENG CAMPUS

Tel: +27 18 3892328

Fax: +27 86 6068714

E-mail: Dave.Kawadza@nwu.ac.za

Internet: http://www.nwu.ac.za

22nd November 2016

Language Editing Certificate

This serves to confirm that I have read and edited the MSc Dissertation of Ms Kedimotse P. Baruni (Student Number: 21508259) entitled:

"The Development of an Accurate Fingerprint Match-on-Card (MoC) Algorithm without the Computation of the Core Reference Point"

The candidate corrected the language errors identified to the satisfaction of the supervisor.

The document presentation is of an acceptable academic and linguistic standard.

Thank you

Dr T.D. Kawadza

Language Editor

Faculty of Agriculture, Science and Technology

Acknowledgement

I would like to express my special thanks and gratitude to everyone who participated in making this research successful. Firstly, I would like to thank the Lord Almighty for giving me the potential and the strength to complete this research.

I am also grateful to my project supervisors: Prof: A.S.J. Helberg, Johan Van der Merwe, and Dr. K. Nair, my mentor: M. Shabalala and the Council of Scientific and Industrial Research (CSIR) for giving me an opportunity to do a Master's degree with them. Without their help and guidance this research would not have been completed.

I also want to extend my sincere thanks and appreciation to my parents, Kingsley Sunday, my elder brother, younger brother and my son for their prayers, encouragement, tremendous contributions and their moral support towards the completion of this project.

Not forgetting to thank my fellow colleagues A. De kock, S. Ntshangase, M. Ratsoma, P. Mabuza and T. Thejane for their ideas and support.

Publication derived from this dissertation

1. K.P Baruni, A. Helberg, K. Nair, "**Fingerprint Matching on Smart Card: A Review**" *International Conference on Computational Science and Computational Intelligence (CSCI'16), 2016, IEEE,* ***Published***
2. M. Ratsoma, A. Helberg, K. Nair, K P Baruni "**Effects of minutiae template on accuracy of fingerprint matchers"**, *International Conference on Computational Science and Computational Intelligence (CSCI'17), 2017, IEEE,* ***Submitted***
3. K P Baruni, A. Helberg, K. Nair, M Ratsoma "**The development of a fingerprint Match-on-Card (MoC) algorithm without the computation of a core reference minutia"** *Pattern Recognition (2017), Elsevier,* ***Submitted***

Abstract

Fingerprint Match-on-Card (MoC) technology offers the highest degree of security and privacy protection to cardholders as the fingerprint template never leaves the secure environment of a smart card. The level of security of a fingerprint matching system is evaluated by the type of the device which is used to compare the fingerprints. Fingerprint MoC compares the fingerprints inside the secure environment of a smart card and makes it possible for cardholders to verify themselves without the use of the central database. However it is challenging to implement an accurate fingerprint matching algorithm inside a smart card and produce an acceptable matching speed. This is due to the limited working memory and processing speed that the smart card provides. This research aimed at implementing an accurate MoC algorithm without the computation of a core reference point. This is because the core reference minutia is not reliably located in poor quality images and is not present in plain arch fingerprint classification. The research focus was on the matching accuracy and speed of the Match-on-Card fingerprint algorithm. Although the accuracy of the minutiae extractor affects the matching accuracy, minutiae extraction is out of the scope of this research. This research deployed a minutiae-based matching algorithm using multiple reference neighbourhood minutiae. The proposed algorithm used multiple reference minutiae to create neighbourhood minutiae circular tessellations. The proposed algorithm used circular tessellations to convert fingerprint features into finger codes. Finger codes are used to compare the fingerprints. The main advantage of the proposed algorithm is that it does not use the computationally intensive process of template alignment. The proposed algorithm also offers the advantages of matching speed with an Equal Error Rate (EER) of 5.5%. The experimental procedures of the proposed MoC algorithm were carried out on the public database DB1-a of Fingerprint Verification Competition 2002 (FVC2002) on a PC using MATLAB.

Definition of Concepts

Terminology	Definitions
Algorithm	A set of unambiguous steps to be followed in calculations or other problem-solving operations, especially by a computer.
Antenna	An electric conductor which has the ability to send, transmit, and receive signals such as microwaves, satellite, and radio signals
Authentication	The process of identifying the individual in order to grant access
Biometrics	Technology that is used to measure biological traits of an individual
Electrical Chip	A tiny piece of semi-conductor (commonly silicon) which is embedded in an integrated circuit
Clock generator	A circuit which is responsible for producing time signal for the purpose of synchronizing the circuit operations
Co-processor	Computer processor which is used to process certain functions quickly in order to assists the Central Processing Unit (CPU).
Core	A centre area of a fingerprint.
Delta	A triangular-shaped pattern where different fingerprint ridges converge.
Encryption	A way of enhancing the security of messages by hiding the messages, so that the message can be readable for those who have the unique key that allows the information to be transformed to its readable form.
Euclidean distance	Distance between two points in Euclidean space.
Fingerprint Alignment	The process of positioning an image with its impression by rotating and/or translating it.
Hamming distance	A metrics used to denote the difference between two binary strings of equal length.
Loop	A pattern where ridges make a backward turn without converging
MATLAB	A high-performance language which combines computation, visualization and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.
Microcontroller	A small computer which is made up of CPU, Memory, system clock, and input/output peripherals.
Off-card matching	The matching process which is performed outside the smart card.
On-card matching	The matching process which is performed inside the smart card.

Plain arch	A pattern where the ridges enters in one direction, make a rise in the centre and flows out in the opposite direction.
Radio Frequency	A frequency which is acceptable for use in telecommunications
Singularity point	The area where the ridge curvature is at maximum than usual and where the direction of the ridges changes swiftly.
Tented Arch	A fingerprint pattern where ridges move to the same direction, make a rise in the centre and flow out upon the opposite direction, with ridges in the centre adjoining each other and intersecting with each other upwards.
Verification	The process of proving the validity of something, truth or accuracy.
Whorl	A fingerprint pattern which is composed of ridges of almost concentric circles.

Lists of Figures

List of Tables

List of Acronyms

CPU	Central Processing Unit
DB	Database
EER	Equal Error Rate
EPPROM	Electrically Erasable Programmable Read-Only Memory
FMR	False Matching Rate
FNMR	False Non-Matching Rate
FTE	Failure To Enrol rate
FVC	Fingerprint Verification Competition
GAR	Genuine Acceptance Rate
GN	Ground
ID	Identity
ISO/IEC	International Organization for Standardization/International Electro-technical Commission
MATLAB	Matrix Laboratory
MB	Megabyte
MoC	Match-on-Card
NPU	Network Processing Unit
PC	Personal Computer
PIN	Personal Identification Number
RAM	Random Access Memory
ROC	Receiver Operating Characteristic
ROM	Read-Only Memory
RST	Reset
ToC	Template-on-Card
TMR	True Match Rate
TRR	True Rejection Rate
VCC	Power Supply

Chapter 1

1. Introduction

Due to technological advancement and the need for stronger personal data security, embedded devices such as tokens and smart cards are widely used for different applications in identification and verification [1, 2]. Different applications include the National Identity Document (ID), Government ID, Corporate ID, Electronic purse, and health cards [1]. Smart cards are portable devices, which resemble a credit card in size and shape. They are used to store personal data securely and to process information through the microprocessor chip embedded in its plastic body [2]. Smart cards usually hold information such as encryption keys, biometric data, and internal functions such as mutual authentication, encryption, and cryptographic algorithms which can also be executed inside the smart card [3]. Figure1.1 shown below illustrates the taxonomy of a smart card:

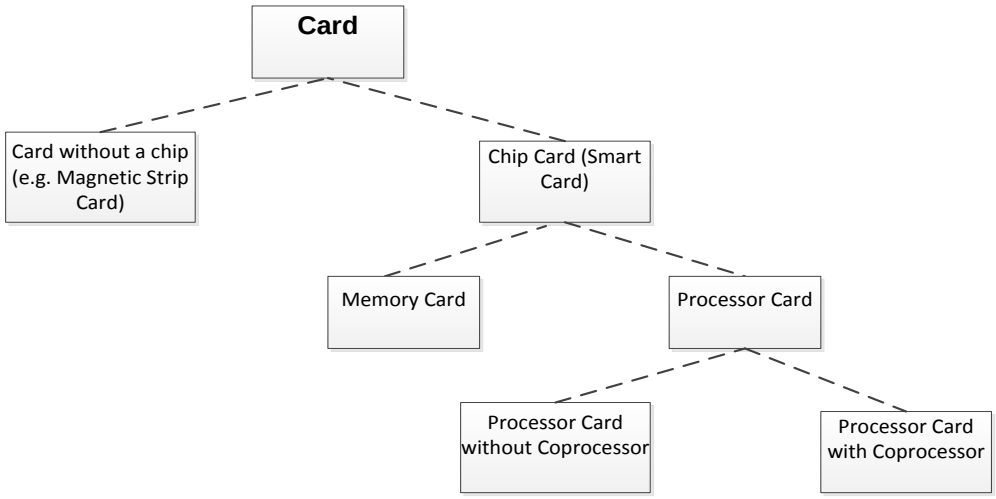


Figure 1.1: Brief taxonomy of a smart card

A card can either be in the form of a chip card or a magnetic stripe card. Chip cards implant the chip within the smart card and have the capability to store and process data [4]. The processing of data can be either done with or without a co-processor. Magnetic stripe cards store digital data using the magnetic stripe of a magnetic material.

Smart cards are a reliable form of secure information repository. Smart cards can be combined with biometric technologies to strengthen the data security when compared to Personal Identification Number (PIN) codes. A PIN code can be easily guessed, detected or stolen via fraudulent means [5, 6]. Biometric technologies make use of a person's unique biological traits and behavioural characteristics such as signature, gait and speech to verify the identity of a person [1]. Biometric modalities that are popularly used include fingerprint, face, voice and iris [5, 7]. Unlike other biometric modalities, fingerprints are mostly used because of their acceptability, accuracy, performance, reliability, numerous sources (ten fingers) available for collection, and their success in law enforcement [7-10].

A fingerprint impression is an image that consists of furrows and minutiae points, which are extracted using ink on paper or by electronic sensors [11, 12]. Furrows are spaces between ridge lines. Minutiae points are Cartesian co-ordinates of a ridge bifurcation and/or ridge ending [8]. Minutiae-based fingerprint matching algorithms are made up of minutiae extraction and minutiae matching algorithms [8, 13].

Singularity points can also be used to assists fingerprint matching processes [14]. There are two types of singularity points, namely: core and delta points [15]. A fingerprint may have more than one, one, or no core reference point. Plain arch fingerprint classification does not have a core point. The core point may be found in whorl, loop, and tented arch fingerprint classification [16]. Table 1.1 illustrates the number of core(s) that are found in the whorl, loop, and tented arch fingerprint classification.

Table 1.1: The location and number of a core(s) in different fingerprint classifications

Fingerprint classification	**Number of cores**	**Location of a core in a fingerprint**
Whorl	2	Located at the centre of a spiral.
Loop	1	Located at the upper point area of an inner loop.
Tented arch	1	Located at the centre of a fingerprint

Figure 1.2, figure 1.3, and figure 1.4 illustrate the fingerprints with a core reference point.

Figure 1.2 : A core reference point (red) for a loop fingerprint [15]

Figure 1.3: A core reference point (red) for a whorl fingerprint [15]

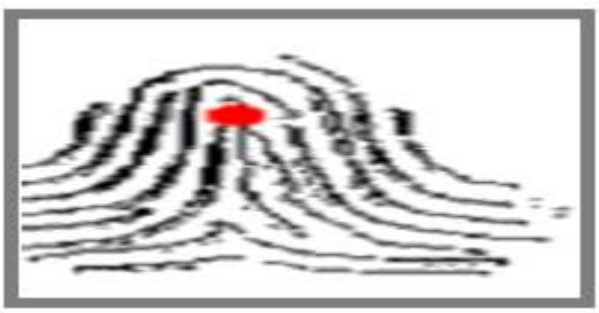


Figure 1.3 : A core reference point (red) for a tented arch fingerprint [15]

Core fingerprint matching algorithms are more efficient when compared to the non-core fingerprint matching algorithms [15]. However, the core point is not reliably extracted from a fingerprint in cases where the image is of poor quality and not all fingerprint classification has a core minutia. Plain arch fingerprints, for example, do not have core minutia [15].

In order to use fingerprint for verification of a person‘s claimed identity, the fingerprint has to be enrolled/registered in a database or a smart card so that it can be used later for comparison/matching [17]. The enrolled fingerprint is termed a reference template whereas the live fingerprint to be compared to the enrolled fingerprint is the query template. During the matching process the enrolled fingerprint will be compared with the query fingerprint inside the terminal or inside the smart card. After comparing/matching the fingerprints, the system rejects or accepts the query fingerprint depending on the match score [17].

Fingerprint recognition systems can be used in two modes, namely verification and identification [3]. In the verification mode, reference template in the database/smart card is compared with the query fingerprint, whereas in identification mode the query fingerprint is compared to the entire database until a suitable match is found [5].

The process of comparing two fingerprints in the terminal is called Template-on–Card (ToC) and comparing the fingerprints in a smart card is called Match–on-Card (MoC) [18].

Figure 1.5 illustrates a fingerprint Template-on-Card process.

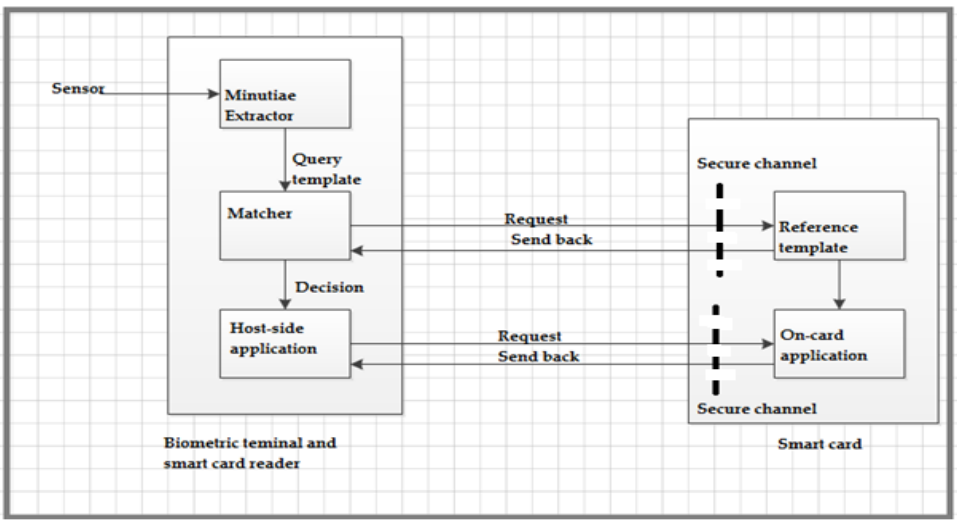


Figure 1.4 : Fingerprint Template-on-Card process [19]

ToC algorithms include the entire process of biometric data acquisition, feature extraction, feature comparison, and score computation on the terminal side.
Figure 1.6 illustrates a fingerprint Match-on-Card process.

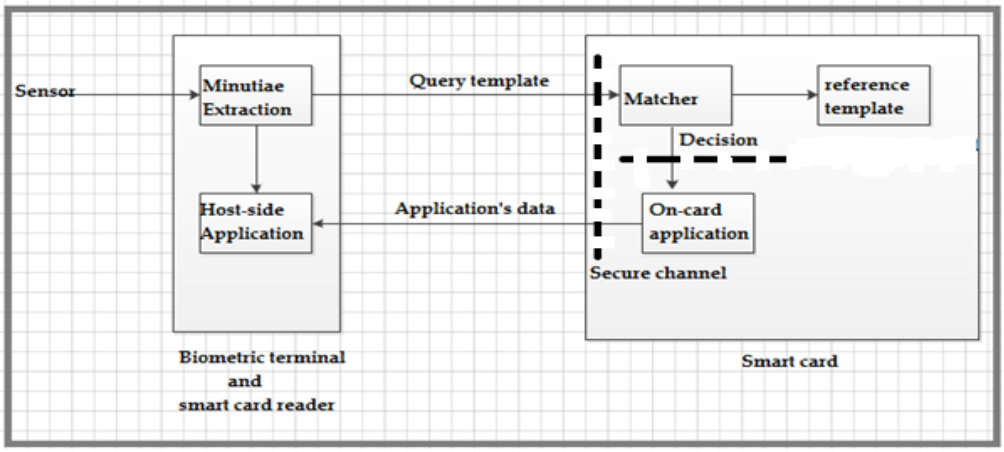


Figure 1.5: Fingerprint Match-on-Card process [19]

MoC algorithms include processes of biometric data acquisition and feature extraction inside the terminal and the comparison of templates inside the smart card [20]. The distinction between MoC and the traditional biometric process (ToC) is the location of where the matching takes place. In ToC algorithms, the smart card sends the enrolled template to the terminal for off-card matching. In MoC, the enrolled template never leaves the secure environment of a smart card. Fingerprint MoC algorithms offer higher data security when compared to ToC algorithms. However, it is quite challenging to develop a fingerprint algorithm which runs under the hardware constraint of a smart card and still produces an acceptable matching accuracy. MoC algorithms which are considered accurate have an Equal Error Rate (EER) which is less than 8% [21]. A fingerprint MoC algorithms which is considered to be fast has a constant, linear or quadratic time complexity. Fingerprint ToC algorithms introduce security concerns due to the template that is transmitted to the terminal via a communication channel during the comparison of the fingerprints. This approach allows the reference template information to be compromised when it is transmitted to the terminal [19].

The proposed algorithm uses minutiae-based matching techniques and multiple reference minutiae to compare the fingerprints. This algorithm uses binary representation for finger codes (minutiae information) and Hamming distances to measure the similarity between the two templates.

1.1. Motivation

The motive behind this work is to prevent identity fraud by using Match-On-Card, which guarantees the physical presence of the user [22, 23]. This work intends to implement a fingerprint MoC algorithm which has a low EER. The fingerprint MoC algorithms makes it hard for an imposter to read the user's fingerprint template as the template is stored in a secure environment of a smart card and also eliminates the need of using a central database [19]. Introducing a MoC fingerprint algorithm that accurately compares fingerprints and is fast, inside the smart card, has always been a challenge, due to the fact that smart cards have limited computational memory and processing capacity. This implies that there is more research that has to be done about the implementation of a fast and accurate MoC algorithm. The fact that passwords and PIN codes are no longer reliable further creates the need to come up with a fast and accurate MoC algorithm [20].

1.2. Problem statement

The problem is that it is quite challenging to develop a fingerprint algorithm which runs under the hardware constraint of a smart card and still produce an acceptable matching accuracy and speed [24]. The proposed fingerprint MoC algorithm avoids using a core reference minutia to align the template. A core minutia is not reliably detected in a fingerprint. The inaccuracy of the MoC usually allows the imposter to access the system, hence promoting identity fraud. The speed of the fingerprint matcher is also important for its eventual success [25].

1.3. Research questions

This research project intends to answer the following questions:

- How can all limited resources that are available inside the smart card be used to implement a light matching algorithm without the computation of the core reference minutia?
- In which way can an accurate and fast fingerprint MoC algorithm be developed?

1.4. Research goal

- Develop and implement an accurate and fast fingerprint MoC algorithm without the computation of a core minutia.

1.5. Research objective

In order to achieve the research goal, the following objectives will be set:

- To investigate different existing fingerprint matching algorithms.
- To implement the proposed fingerprint MoC algorithm.
- To evaluate the speed and matching accuracy of the proposed algorithm.
- To compare the matching accuracy and time complexity of the proposed algorithm with the work of Benhammadi and Bey [21].

The proposed algorithm is compared to the work of Benhammadi and Bey due to the following reasons:

- To prove that the proposed algorithm can obtain the matching accuracy that is comparable to a core-based matching algorithm.
- The work of Bey and Benhammadi offers the advantage of simplicity. It does not use complex mathematics. Hence it makes it easy to be implemented on a smart card unlike other algorithms which are presented in chapter 3 (literature review).

1.6. Delimitations, Limitations and Assumptions

- It is assumed that the number of missing and false minutiae in the templates which are used to evaluate the performance of a matcher is less than the total number of the minutiae which are correctly extracted. This is due to the obtained matching accuracy of the proposed algorithm. After conducting the experiments, the experimental results showed that the matching accuracy of the proposed algorithm was only affected by 0.49% errors which are caused by the minutiae extractor.
- The research focus is on implementing an accurate and fast fingerprint MoC algorithm.

- An algorithm will be implemented using the existing extracted templates. Extraction of minutiae points and the securing of the system are outside the scope of the proposed research.

1.7. Contributions

The research contributions envisaged are in what follows:

- Reduce the effect of distortions by using more than one minutia point in the template as a reference minutia [26].
- The proposed algorithm can be deployed in different applications such as National ID, Electronic purse, banking, retail and insurance cards.

1.8. Dissertation layout

This dissertation layout is as follows:

Chapter 1 gives an introduction about a problem to be solved. It also includes the motive behind this research, research goal, research objectives, research questions, research contributions, and the scope of this research.

Chapter 2 gives a brief history of smart cards and a detailed background to this dissertation. It presents the brief history on the smart card and the biometric technologies.

Chapter 3 conducts the literature review of the existing work done by other researchers. It describes how other authors presented fingerprint matching algorithms and the fingerprint matching accuracy of those fingerprint matching algorithms.

Chapter 4 gives details about how the problem was solved and how the experiments were conducted. This chapter provides a detailed methodology that was used in this research to achieve the aims and objectives of this study.

Chapter 5 explains how the methodology was implemented. This chapter discusses the implementation environment of this research methodology.

Chapter 6 analyses and presents the experimental results of the approach. It also provides the verification and the validation of this research methodology.

Chapter 7 gives a summary and the conclusion drawn from the previous chapters and the future work of this research dissertation.

Chapter 2

2. Background

This chapter presents the background to the research for this dissertation**.** It begins by briefly explaining the architecture of a smart card and how biometrics can be used with smart cards. As the chapter proceeds it explores the concept of MoC and also the traditional biometric process (ToC). Section 2.3 describes the features which can be used to compare the fingerprints and how the features can be used. This chapter concludes by describing different fingerprint matching techniques.

2.1. Smart cards technology

Smart cards were invented at the end of the mid-seventies by Michel Ugon [21]. Around that time there was a lot of discussion about standardization of the contact location, the standardization of the signal, and protocols which resulted in the ISO/IEC 7816 smart card standards [21]. Protecting the information was also required; it was obvious that cryptography codes were required [22]. However, it was challenging to implement some cryptography in a smart card environment due to hardware constraints on the smart card processor.

Nowadays, smart cards can have 8, 16 or 32 bit processors with limited volatile memory of 2-16 Kbytes of Random Access Memory (RAM), 64-300 Kbytes of Read-Only Memory (ROM) and 32-150 Kbytes of Electrically Erasable Programmable Read-Only Memory (EEPROM) with options of Flexible Architecture for Shared Memory (FLASH) [27, 28]. RAM is a kind of storage where the data that is currently used in a program is stored so that it can be quickly reached by the processor [29]. ROM is non-volatile and non-writable storage which contains the operating system and encryption algorithms. EEPROM is a read/write non-volatile memory which stores program data, user data, and operating system routines [28]. The processor is embedded in a chip and it is connected to the outside world through eight contacts as shown in figure 2.1.

Figure 2.1 illustrates the internal components of a smart card.

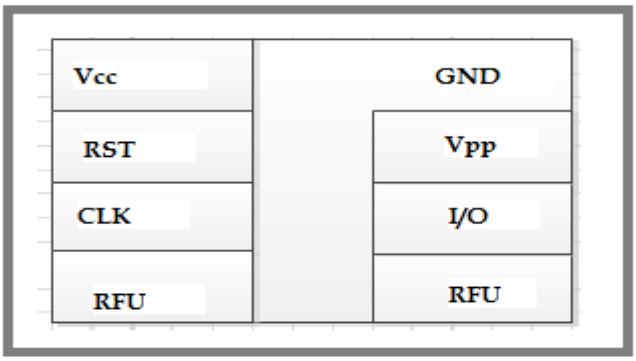


Figure 2.1 : The typical architecture of a contact interface smart card [30]

In figure 2.1, Vcc is the supply voltage that drives the chips. CLK is a clock signal which is used to drive the logic of the integrated circuit and it is also used as a reference to serial communication [31]. After the power is on, Reset (RST) is used to initiate the state of the integrated circuit [31]. GND is the ground reference voltage. Vpp connector supplies the programming voltage input in a smart card. Input/output (I/O) provides a serial data communication between the interfacing device and the smart card. RFU is reserved for future use [31].

Smart card usage has advanced tremendously over the past few years as well as the use of biometric technology because smart cards can store data (biometric traits) securely [2, 4, 21]. This is because according to ISO/IEC 24787 (2010) on-card biometric comparison standard, there is no software that can be used to download the stored template(s) inside the smart card [32]. Biometric technology has shifted from research labs to real-world applications and can well be used with smart cards [21, 23]. Since 1999 a variety of vendors have been developing MoC fingerprint algorithms [24].

2.2. Different classes of fingerprint

Fingerprint matching can be done using texture (region) of a fingerprint, minutiae points, image-based matching, and the combination of both fingerprint texture-based matching and minutiae-based matching [33].

2.2.1 Image-based fingerprint representations

Image-based fingerprint matching algorithms make use of local orientation, frequency, ridge shape, ridge count and texture information to compare the fingerprints. Image-based fingerprint representation offers the following advantages:

- All the features that are used during the matching process are extracted more reliably unlike the minutiae-based algorithms [34].
- They can be used with minutiae-based algorithms to enhance matching accuracy [5].
- Invariance to affine transform[1], hence they can be used to deal with distinct input states [34].

The introduction of ridge counts was also used due to the difficulty to obtain the exact measurements of the Euclidean distance between the minutiae [5]. The disadvantage of these algorithms is that they do not have sufficient capability to track down differences in position, scale, and rotational angle of the fingerprint. They are affected by differences in image quality, scars, brightness variations, large amount of global distortion in the fingerprint image and they also suffer from deformation of fingerprints in the long run [5, 35]. These algorithms find the centre point or the singularity point of the fingerprint to align the template, and then the reference template is graphically matched with the query template to compute the matching score. Detailed image-based fingerprint algorithms are often too large to be stored on a smart card [36].

2.3. Minutiae-based fingerprint algorithms

Minutiae-based algorithms use point-to-point matching to identify the similarity between two templates by using the minutiae [33]. The reference template (T) and the query template (I) are used in order to check whether the templates are from the same finger by matching all the corresponding minutiae pairs from both T and I. Minutiae can be characterized by making use of the feature vector representation [5]:

$$m_k = (x_k, y_k, \theta_k, t_k)$$

where x_k and y_k represent the position of the x-coordinates and the y-coordinates of the minutia respectively, θ_k indicates the minutia angle, t_k indicates the minutia type

[1] Affine transform is any transform that preserves collinearity

(ridge bifurcation and ridge ending), and k represents the minutia number. Characteristics which are related to points and their structural properties, such as Euclidean distance between points are usually used in minutiae-based techniques to reduce the exponential number of search paths. Most of the ordinary fingerprint minutiae matching algorithms use only three feature vectors to represent the minutia $m_k = (x_k, y_k, \theta_k)$ as illustrated below.

Each set of minutiae T and I can be represented as:

$T = \{m_1, m_2, \ldots . m_k\}$, $m_i = \{x_i, y_i, \theta_i\}$, $i = 1 \ldots . k$

$I = \{m'_1, m'_2, \ldots m'_p\}, m'_j = \{x'_j, y'_j, \theta'_j\}, j = 1 \ldots . p$

where k and p represent minutia number in T and I, respectively. In order to compare the minutia m_i in T and m'_j in I, a matcher chooses a certain threshold for the spatial distance (sd) and the minutia direction (dd) which is going to be used to indicate whether the two minutiae match [5].

$$\boldsymbol{sd}(\boldsymbol{m_i}, \boldsymbol{m_j}) = \sqrt{\left(\boldsymbol{x'_j} - \boldsymbol{x_i}\right)^2 + \left(\boldsymbol{y'_j} - \boldsymbol{y_i}\right)^2} \leq \boldsymbol{r_k} \text{ and} \quad (2.1)$$

$$\boldsymbol{dd}(\boldsymbol{m_i}, \boldsymbol{m'_j}) = \boldsymbol{min}(\boldsymbol{\theta_i} - \boldsymbol{\theta'_j}) \text{ , } 360 - \boldsymbol{\theta_i} - \boldsymbol{\theta'_j} \leq \boldsymbol{\theta_k} \quad (2.2)$$

The chosen thresholds θ_k and r_k are required to compensate for the mistakes made by feature extraction algorithms and to account for the small plastic distortions that cause the minutiae positions to change [5].

Minutiae-based algorithms are mostly used for MoC fingerprint verification because of their stronger matching accuracy due to the following reasons:

- It is unusual for a large amount of minutiae from different fingers to match [37].
- Every set of minutiae is unique in each finger and is well extracted when compared to other features such as pores, delta and ridge structures. Moreover, they require low computational cost in smart card implementations [35]. The delta structure is a triangular-shaped pattern where different fingerprint ridges converge.

- Minutiae-based algorithms are responsive to different kinds of fingerprint matching degradation (fingerprint distortion, rotation, translation) and they usually use a small amount of memory [20].

The commonly used minutiae-based fingerprint matching algorithm is the Hough transform (HT) method [35]. Minutia-based matching algorithms are usually initiated by choosing a reference singular point or a reference minutia in a chosen neighbourhood of minutiae.

Normally when minutiae-based representations are used in MoC, the extracted minutiae are sent to the smart card for comparison by pairing all the corresponding minutiae points between T and I and also to calculate the matching score. The final stage is decision making by either accepting or rejecting the query tempate according to the obtained matching score.

2.4. Combining minutiae-based algorithm with image-based algorithms

During minutiae extraction, when a minutia is detected, it is recorded with its position, direction, and minutia type $(x_k, y_k, \theta_k, t_k)$. Minutiae-based algorithms can be combined with image-based algorithms in order to enhance the matching accuracy. However, this technique usually takes a long time to execute. It is largely affected by noisy images and high amounts of distortion [21].

2.5. Fingerprints matching techniques

The following fingerprint matching techniques are used in biometric and smart card technologies [38]:

- Alignment techniques
- Work sharing on-Card techniques (WSoC)
- Neighborhood Minutiae techniques

Fingerprint alignment is a very computationally intensive process. Due to the memory and processing speed that the smart card provides, other researchers presented WSoC algorithms [39]. WSoC algorithms make use of the terminal to perform computationally intensive processes such as minutiae extraction and template alignment in order to reduce the workload on the smart card. During the matching process, the smart card sends the enrolled template to the terminal via a

communication channel to perform the most intensive computational steps. The computed results are sent back to the smart card to calculate the final matching score. Neighbourhood minutiae techniques are usually used in fingerprint MoC to avoid template alignment and WSoC. Neighbourhood minutiae are local minutiae structures of the fingerprint. Neighbourhood minutia structure is characterized by attributes that are rotation and translation invariant.

2.5.1. Alignment techniques

Fingerprint alignment is the process of positioning a reference template with a query template by rotating (θ) and translating (x,y) it. This kind of method increases the number of matching minutiae in a different impressions of the same finger [2]. Minutia alignment involves matching n minutiae points in the reference template with m minutiae points in the query template. This procedure has an exponential time complexity O(M^N) [40]. Pre-aligning the query template and the reference template before the minutiae matching stage increases the matching accuracy of the fingerprint algorithms. This method is able to handle noisy and insufficient data. Nevertheless the method is very fast (e.g. 10 or more matches per second) [2].

Figure 2.2 illustrates a fingerprint alignment process.

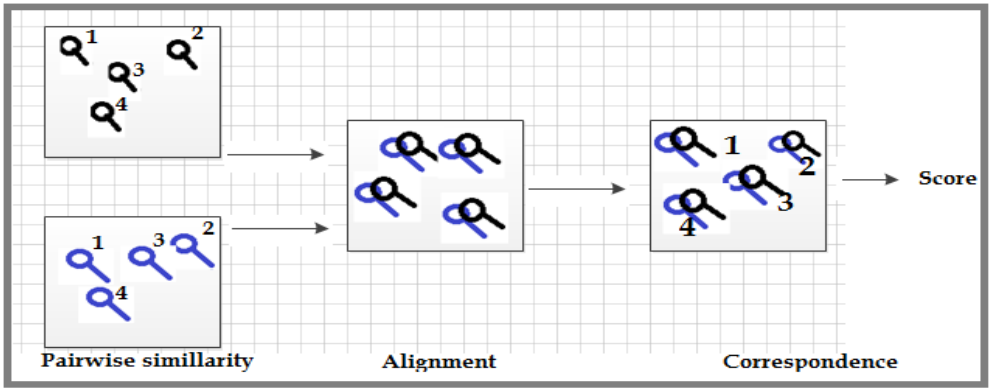


Figure 2.2 : Fingerprint alignment process [40]

In order to align the two templates, the matcher computes the pairwise similarity between the minutiae in the reference template and the query template by comparing minutiae discriptors which are translation and rotation invariant. Then the two templates will be aligned with regard to the most simillar minutiae pair. The obtained overlapping minutiae will be used to calculate the matching score. If the

amount of overlap is high, it implies that T and I belong to the same finger. Contrarily a small overlap implies that there is higher possibility that T and I are not from the same finger.

There are two types of pre-alignment techniques which are used, namely: the absolute pre-alignment technique and the relative pre-alignment technique. The relative pre-alignment pre-aligns the query template with regards to the reference template. Relative pre-alignment technique is used commonly and it can be executed by superimposing the singularities and comparing ridge features. In cases of singularity based algorithms, if the core location and the core orientation is detected accurately for the query and the reference template, it becomes easier for the transformation to lead to a proper alignment. The absolute pre-alignment mode pre-aligns each fingerprint indepedently of each other before it is stored.

2.5.2. Work Sharing On-card (WSoC) techniques

Work Sharing on-Card is a process of making use of the terminal and the smart card to match the fingerprint. It is mostly used for smart cards as the majority of them do not have sufficient computational power and memory to process fingerprint data [39].

Figure 2.3 illustrates the architecture of the Work Sharing on-Card.

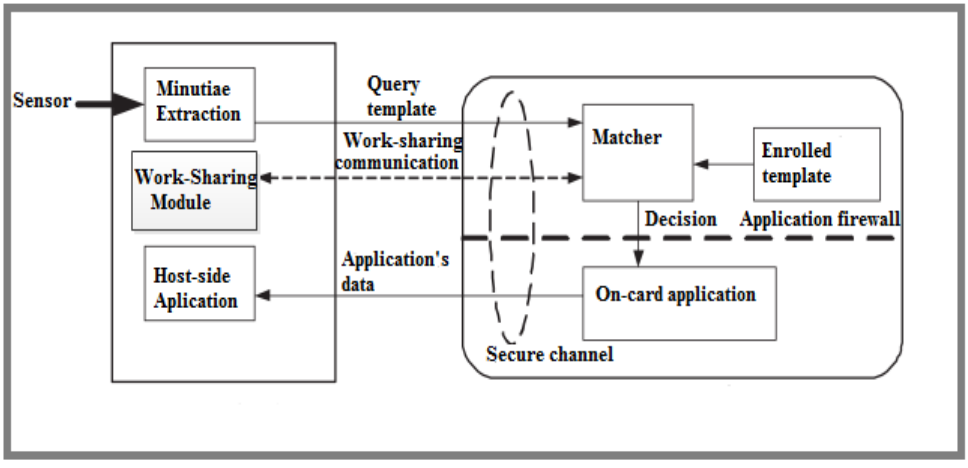


Figure 2.3: Architecture of Work Sharing on-Card [19]

WSoC algorithms make use of the terminal to perform computationally intensive processes such as minutiae extraction and template alignment in order to reduce the workload on the smart card. During the matching process, the smart card sends the enrolled template to the terminal via a communication channel to perform the most

computationally intensive steps. The computed results are sent back to the smart card to compute the final matching score [15]. The advantage of WSoC algorithms is that the Personal Computer (PC) relieves the computational load of aligning the templates from the smart card. The existing pre-computations which are done outside the smart card for MoC algorithms produce very accurate matching results. However, this technique introduces security concerns, since the reference template can be intercepted due to the reference template information which is sent out of the smart card. WSoC requires secure communication between the smart card and the terminal. However, it is a challenging to properly implement secure communication between a smart card and the terminal.

The disadvantage of these algorithms is that they usually take too long to match the fingerprint because of the communication delay between the PC and the smart card during the verification process.

2.5.3. Neighbourhood minutiae techniques

In the fingerprint MoC, neighbourhood minutiae refer to the number of all the minutiae points which are surrounding the reference singular point or the reference minutia point which is going to be used to compare the fingerprint images. Neighbourhood minutiae techniques often use geometric transformations in order to compensate for alignment. In most cases using geometric transformation rather than rotation and translation can result in a large number of new possible alignments which can introduce the chances of getting false matches. Therefore, the method needs to be carefully assessed [2]. Neighborhood minutiae can be used to construct a finger code.

Figure 2.4 shown below illustrates a finger code.

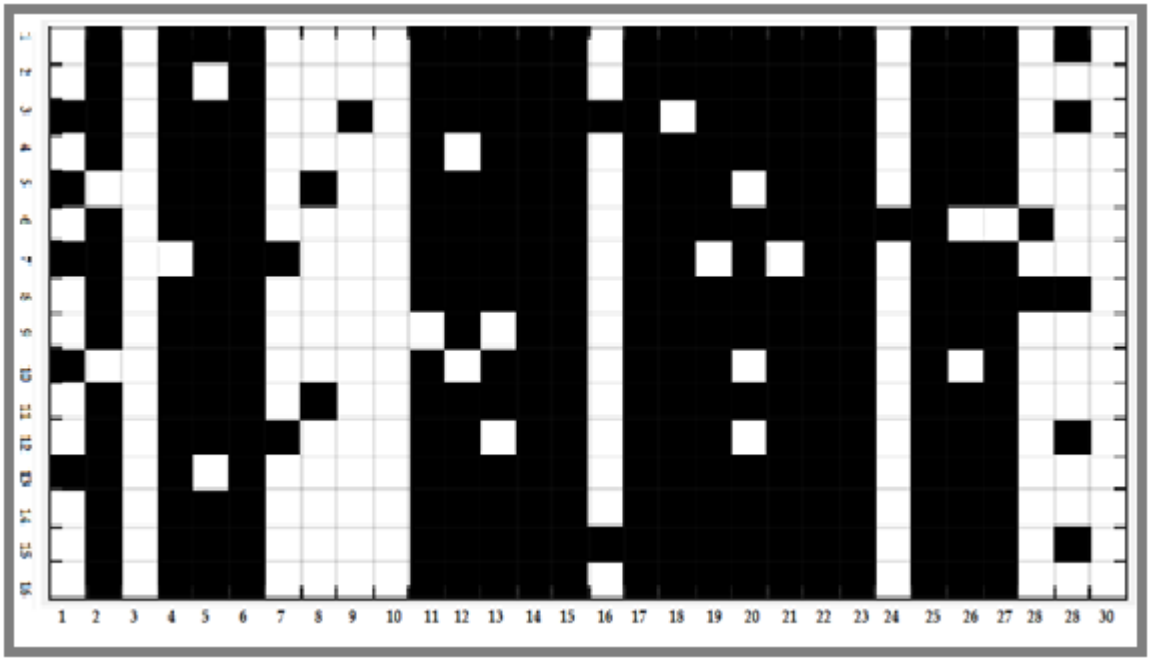


Figure 2.4 : A finger code with 16 rows and 30 columns

Every white cell in a finger code represent a bit value one (1) and every black cell in a finger code represent a bit value zero (0). Finger code makes use of binary codes to match the fingerprints. The finger code matching method is very fast when compared to the traditional neighborhood minutiae fingerprint matching. Finger codes usage offers the advantage of matching speed and less processing inside the smart card due to the usage of binary code. This is because the smart card/computer uses binary code language. A smart card holds information in a form of binary codes which make finger codes suitable for MoC.

2.6. Summary

Fingerprint MoC algorithms provide stronger template security in comparison to ToC and WSoC algorithms. Minutiae-based techniques are usually used in MoC algorithms because they are more accurate and robust than image-based representations. Detailed image-based fingerprint algorithms are often too big to be stored on a smart card. Neighbourhood minutiae matching techniques requires less processing computations when compared to alignment matching techniques. Hence neighbourhood minutiae matching techniques are suitable for MoC algorithms.

Chapter 3

3. Literature Review

This chapter presents distinct minutiae-based and image-based fingerprint Match-on-Card (MoC) algorithms which are concerned with matching accuracy. The next chapter describes processes that are used during the verification process and the level of accuracy obtained for each fingerprint matcher depends on the kind of algorithm used.

3.1 Alignment techniques

Fingerprint alignment is the process of positioning a reference template with the query template by rotating and/or transforming it. The purpose of alignment is to estimate the translation and rotation parameters between the query template and the reference template. Using fingerprint alignment techniques to align minutiae usually leads to the development of a robust algorithm, which can accurately compare noisy fingerprints. However computing template alignment is time and resource consuming [5].

3.1.1. Minutiae-based and image-based matching algorithms

Current MoC algorithms extremely underperform as compared to terminal or personal computer matching algorithms in terms of accuracy due to limited resources available inside the smart card [24]. J Feng *et al.* presented a personal computer matching algorithm. This method make use of minutiae and ridges to perform fingerprint matching [9]. This method used FVC2002 DB1, DB2, DB3 and DB4 to evaluate the accuracy of the algorithm. The method improves the matching accuracy for minutiae pairing. However it uses more time to execute the matching.

Jain and Feng proposed a hybrid approach to improve matching accuracy [41, 42]. These algorithms make use of minutiae points and texture-based (region) matching to compare the fingerprints. Different templates from the same subject (finger) may have a small region of overlap. Hence, minutiae-based matching has less probability of performing well when dealing with these subjects. Therefore, the combination of minutiae-based and texture-based algorithms was a solution. The algorithm [41] showed a matching improvement when compared to the minutiae based algorithm

which uses ridge patterns for matching [14]. However, the algorithm did not account for non-linear deformation present during fingerprint matching.

3.2. Minutiae-based algorithms for smart cards

Sanchez-Reillo *et al.* presented a minutiae-based MoC fingerprint algorithm [43, 44]. It uses an elastic matching algorithm, in order to handle fingerprint elasticity. The algorithm introduced new variations, such as the distance between ridges. The algorithm locates the reference minutia point for T and I, and converts all the neighbourhood minutiae points to polar coordinates with respect to the reference minutia. Then feature vectors are further aligned to calculate the pairing minutiae between the query template and the reference template. The results showed that the implementation code was required to be optimized because of intensive processing requirements.

Pan *et al.* presented an optimized fingerprint minutiae-based MoC algorithm [45]. The algorithm does not only focus on accuracy but also considers the memory requirement and processing power which is used for every step during the matching process. To minimize the memory requirements, a small-sized accumulator array was employed to perform more computations at a course-grain to fine-grain resolution on the accumulator array. The accumulator array is a matrix that is used to store values (Δx ($change\ in\ the\ x_{axis}\ position$), Δy ($change\ in\ the\ y_{axis} position$), $\Delta\theta$ ($change\ in\ an\ agle$)) of the accumulated votes. According to the experimental results, the algorithm obtained a comparable EER to typical algorithm with minimum memory requirements.

A. S. Rikin *et al.* proposed a fast memory efficient fingerprint MoC algorithm [35]. This algorithm uses minutiae-based matching and use of HT for alignment [46]. However, this technique fails to perform alignment of two patterns with inadequate overlapping matching minutiae. The algorithm offers the advantages of fast matching time, reduced memory requirements, small template size over minutiae-based MoC algorithms. The minutiae ridge shape matching technique offers the advantages of providing more information per bit. Hence, it requires a smaller template size when using minutiae ridge shape matching technique. The algorithm obtained the minimum FMR of 2.97% using a template size of 64 bytes.

3.3. Singularity-based techniques

In cases where a core is located properly for two templates, the transformation leading to the correct alignment becomes easy. It is difficult to reliably locate the core point in noisy images.

Ishida *et al.* proposed an image based algorithm which computes the reference singular point (core) and stores the reference template inside the smart card during enrolment [4]. During verification the host sends the smart card only a certain portion of the fingerprint sequentially in relation to the core. The smart card compares the aligned portions from the query template with the corresponding portions from the reference template using correlation. This algorithm saves computational time by tracing the pseudo ridges to determine the fingerprint classification instead of thinning and converting the fingerprint to binarized format. The algorithm also improves the classification accuracy. However, the matching results of the algorithm were required to be modified. This algorithm failed to obtain the exact directions of ridges for poor quality images and detecting singular points.

Tommaso *et al.* proposed a fingerprint MoC algorithm which constructs a graph of minutiae structure, encodes minutiae points and also uses a core in relation to its neighbourhood minutia for fingerprint matching [47]. The graph is bounded by the in-degree and out-degree to represent the limitation of all the neighbourhood minutiae points. This algorithm uses minutiae points in the template to build spanning, ordered tree touching as many nodes as possible, starting from the core from both the reference template and the query template and going through the two graphs by matching edges.

Figure 3.1 illustrates how this algorithm selects the minutiae for matching.

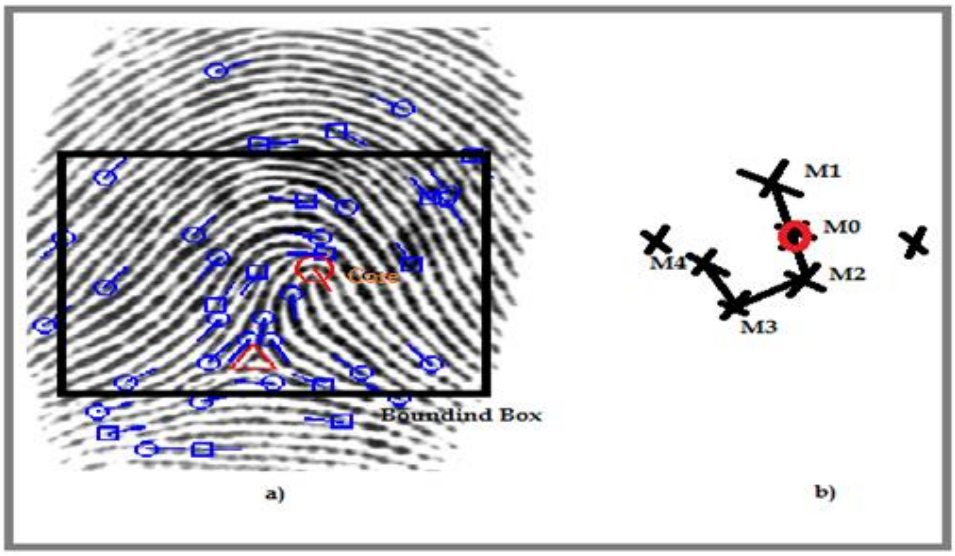


Figure 3.1 : a) Bounding box surrounding minutiae and a core reference point (red circle) b) Spanning ordered tree which is created from the position of the minutiae with respect to the core [47]

The core point from figure 3.1(a) is a reference point which is used to initiate the matching process and a bounding box includes the region of a fingerprint that is going to be used for the matching. In figure 3.1(b) the algorithm starts by drawing a line from m_0 (core) to the neighbourhood minutiae (m_1,m_2,m_3,…m_k), in order to construct a spanning ordered tree. The spanning ordered tree is used for minutiae pairing. The critical factor of this algorithm is the computation of the location of the core.

3.4. Work Sharing on Card (WSoC) algorithms

Due to the limited memory and processing speed that the smart card provides, other researchers presented WSoC algorithms.

Moon *et al.* presented a pre-match computation in the host computer in order to reduce the workload of the smart card [48]. The host is responsible for calculating the average position and orientation of T and I. Afterwards, the host computes the average position differences and orientation between T and I. All this information is sent to the smart card. Even though not all computations are done inside the smart card, the algorithm makes it hard for an imposter to get hold of the minutiae

information. This is due to the fact that information that is sent to the smart card can only reveal the average position and the average orientation of the minutiae points, but not the location and the orientation of the minutiae. The smart card is responsible for performing the point-to-point matching and computing the matching score, whereas the host is responsible for transforming the fingerprint.

Figure 3.2 below illustrates the Share-on-Card algorithm.

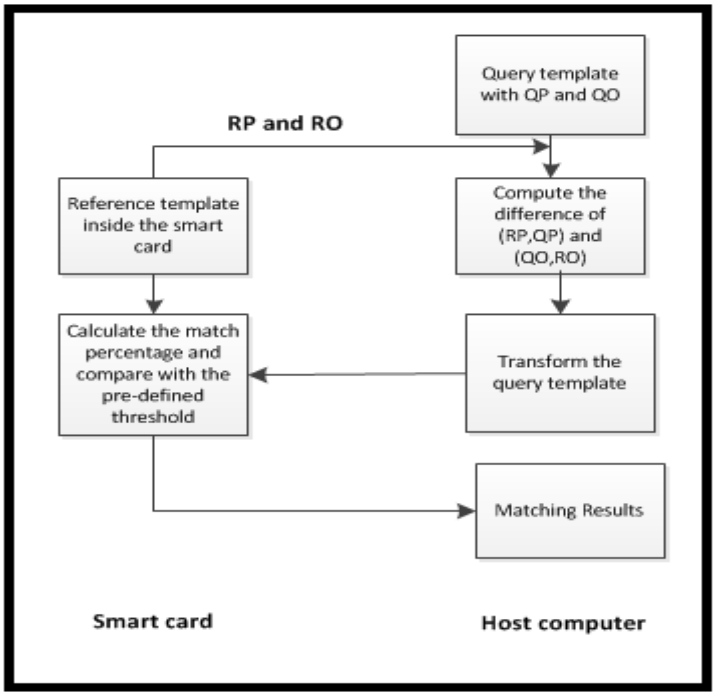


Figure 3.2 : Share-on-Card algorithm presented by Y Moon *et al.* [48]

In figure 3.2 illustrated above QP is the average position of a query template, QO is the average orientation of the query template, RP is the average position of a reference template and RO is the average orientation of the reference template.

In order to eliminate the security concerns of the information that goes in and out of the smart card, Y. Moon *et al.* decided to do the entire fingerprint matching inside the smart card [48]. The algorithm rotates the fingerprint by making use of the angular difference between T and I, which is computed from the polar angle of each minutia. This algorithm is less computationally intensive than the two previous algorithms that were previously introduced by Y Moon *et al.* Instead of using the mean position and the mean orientation of the minutiae for pre-alignment, Moon *et al.* improved their work by using a robust core to align the template [49]. This also has its own

disadvantages because there are cases where the core cannot be located accurately. Moreover, it can lead to matching errors since the arch type fingerprint does not have the core.

Figure 3.3 illustrates the modified MoC from the work of Moon *et al.*

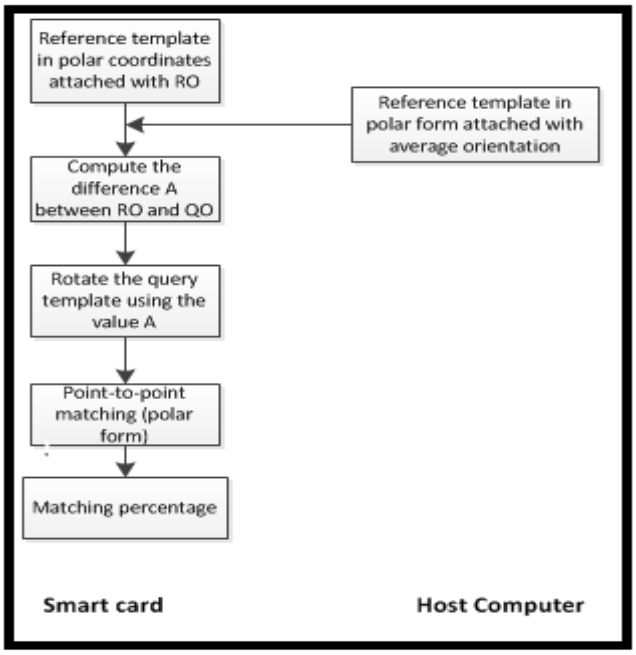


Figure 3.3 : Modified MoC algorithm [22]

H.K. Lam also proposed a fingerprint pre-alignment matching algorithm which does not utilise large amounts of memory inside the smart card [50]. This matcher computes an off-card reference point and an off-card template alignment. The algorithm computes the location of the reference singular point/core and extracts five different regions from the reference template. The first region is located at the reference singular point and the rest of the regions are located around the core. This method extracts five regions of 35*35 pixels. The triple representation of the regions is used during template alignment to bring into line the query template. The method transfers only the triplet representation and normalized x-y coordinates of the five regions out of the smart card. The terminal usually uses the core as an alignment point to align the templates. This is because the core is highly unique, so the highest correlation is usually found in first region (core). In cases where the image quality of a fingerprint is bad and the core cannot be accurately detected, this method finds the highest correlation between region 2, region 3, region 4, and region 5. Then the

region with the highest correlation score is used as a point of alignment. The advantage of this algorithm is that, it can match the fingerprint even without the computation of the core. Figure 3.4 illustrates five minutiae which are used to perform the matching.

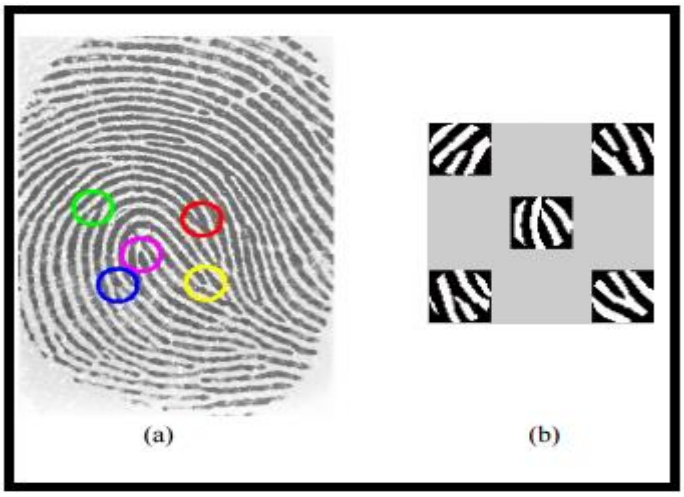


Figure 3.4 : (a) Core reference point (purple) surrounded by neighbourhood minutiae (regions) (b) Binarized neighbourhood minutiae (regions) [50]

In T P Chen *et al*. proposed a minutiae-based fingerprint matching inside the smart card using an 8-bit smartcard [39]. This method computes reference minutia template alignment in the terminal and uses the smart card for comparison of minutiae template. This method divides a template into two portions namely: open portion and secure portion. The secure portion contains minutiae positions (x-y coordinates), minutiae orientation, and minutiae type. The secure portion never leaves the secure environment of a smart card. An open potion is sent to the terminal to compute a reference minutia and template alignment to speed up the matching process. Open portion contains relative minutiae with limited number of nearest neighbourhood minutiae information. Hence it is not easy to use this information to reverse engineer the original minutiae information. The communication between the smart card and the terminal is encrypted using ISO/IEC7816-4. This method obtained good average recognition rate. However, this method introduces security vulnerabilities due to the template alignment which takes place outside the secure environment of the smart card.

3.5 Neighbourhood Minutiae Techniques

Victor and Suandi [51] presented minutiae-based fingerprint matching algorithm using finger codes. Finger codes make use of binary digits to compare the query fingerprint. This method generates a circular tessellation by using a core as a centre point. Then tessellate the neighborhood minutiae into 4 circular bands and 16 sectors. The radius of 20 pixels from the core reference minutia is excluded from the finger code.

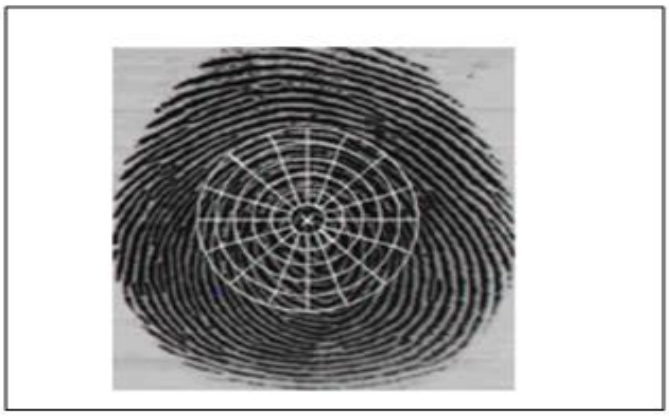


Figure 3.5: Circular tessellated image with “X” as a core minutia

After the tessellation is define and the core is extracted, the window size of 199*199 pixels which includes the entire circular tessellation image is cropped from the fingerprint. This method only use the window size of 199*199 which contains a circular tessellated for matching to speed up the matching process. The matcher uses 8 finger codes to represent the local features of the fingerprint for higher matching accuracy. Every finger code which is constructed is rotated 25° from the previous finger code. This is used to make the fingerprint verification rotation invariant. Each finger code has 64 pixel values which represent the sector. The 8 finger codes are concatenated into 1 dimensional vector to form a final finger code of 512 pixel values. Euclidean distance is used to compare the enrolled finger code and the query finger code. If the Euclidean distance is more than the threshold, the n^{th} enrolled finger code (in the final enrolled finger code) is compared to the (n+1) query finger code in the final query finger code. This loops 7 times to compare with the other finger codes unless the match is found before the loop reaches the 7^{th} alteration. This method performs the smart card verification using the password and

PIN code. If the smart card verification is successful, the fingerprint verification process is carried out. This method obtained True Acceptance Rate (TAR) of 90% using 40 fingerprints of 10 different fingers, with each finger containing 4 impression of the same finger. The disadvantage of this method is that it does not use a standard database. Hence the matching accuracy of this method is not compared fairly. The other disadvantage of this method is that it does not always accurately detect the core reference minutia which degrades the matching accuracy of a matcher. This method introduces security concerns. This is because the comparison of the fingerprints is done in the terminal's side and enrolled template is stored in the terminal. In order to overcome security concerns, fingerprint WSoC and MoC algorithms were presented.

T.P. Chen *et al.* presented a method which uses a smaller portion (clusters) of minutiae instead of using all the minutiae in the template to reduce computational load and increase the matching accuracy [32]. This method searches for the corresponding cluster of minutiae in different locations between the query template and enrolled template. To avoid false acceptance which is caused by the presence of false minutiae inside the clusters, this method deploys mahalanobis distance to measure the inter-class similarity and remove the incorrectly matched clusters. This method uses the combination of matched minutiae in clusters and the geometrical structure between clusters to calculate the overall matching score between the enrolled template and the query template. The matcher computes the local similarity score and the group similarity vector to find the matched clusters. This approach sorts the minutiae according to ISO/IEC 24787 to increase searching speed for the first cluster. This method uses a smaller portion of the fingerprint for matching to reduce the computational load in the smart card. The method can also be implemented in a lower cost smart card (8-bit smart card) and offers a good recognition rate. However, accuracy of this method is mostly affected by minutiae extractor (false minutiae).

Stefano *et al.* proposed an asymmetric neighbourhood minutiae-based fingerprint MoC algorithm which makes use of the following local comparison features [52]:

- Euclidean distance and the ridge count between the reference minutia and its neighbour.

- The difference angle between the reference minutia and the neighbour ridge orientation angle.
- The angle between the segment D in figure 3.6 which is formed by the reference point and the neighbourhood minutia and the reference minutia ridge direction.

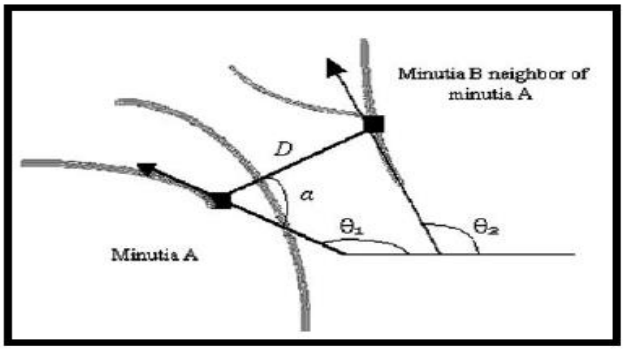


Figure 3.6 : Local comparison feature [52]

After making use of the above comparison features, the two neighbourhood minutiae with the lowest similarity value are discarded and the rest of the remaining matching neighbourhood minutiae are summed together. The summation of the minutiae is used to decide whether two templates match by making use of the threshold when computing the global score. During the verification, the algorithm exits the matching process as soon as a very good average similarity value is found. The approach offers the advantages of speed because of its asymmetric nature regarding execution time. The disadvantage of this approach is that it performs lots of computations to pair the minutiae.

Govan *et al.* also proposed an asymmetric neighbourhood minutiae-based fingerprint MoC algorithm which is similar to Stefano *et al.* [52, 53]. This algorithm uses the entire local comparison feature that is used by Stefano *et al.* excluding using the ridge count between the reference minutia and its neighbour and the reference minutia ridge direction. In order to eradicate the use of alignment that is computationally intensive, the algorithm uses basic arithmetic functions [53]. The algorithm deals well with displacement, rotation and deformation. In some instances the algorithm encountered errors due to the partial overlap area between samples arising from noise in the templates.

Benhammadi *et al.* proposed a fingerprint matching algorithm which makes use of a reference point/core to initiate the matching process and circular tessellate the templates starting from the core [21]. The algorithm constructs a finger code from a neighbourhood minutiae tessellation. The matcher tessellates the fingerprint into 32 sectors and 16 circular bands, using a core as a centre point. Once the circular tessellation is generated, the matcher constructs a finger code out of the tessellation. The circular tessellation consists of 512 bits (32 sectors *16 circular bands). Figure 3.7 illustrates how the finger code is obtained from the circular neighbourhood minutiae tessellation.

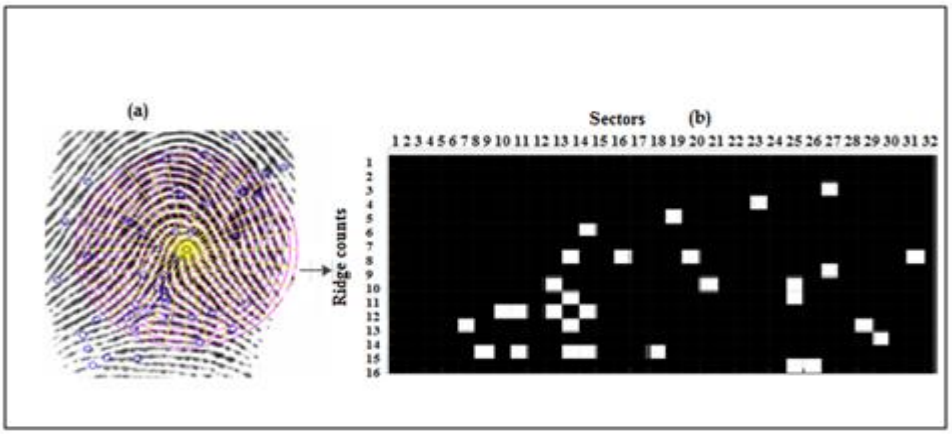


Figure 3.7 : (a) Neighbourhood minutiae tessellation (b) Finger code

Figure 3.7 (a) illustrates the neighbourhood minutiae tessellation and figure 3.7 (b) illustrates a finger code. The white cell in the finger code represents the presence of minutiae and the black cell represents the absence of minutiae. Hamming distance is used to measure the similarity between the finger codes. It is a metric used to denote the difference between two binary strings of equal length. The proposed algorithm offers the advantages of matching speed, simplicity, and good recognition rate. The matching process relies on the minutiae neighbourhood near the singular reference minutia (core). The disadvantage of this algorithm is that, it depends entirely on the core minutia to compute the finger code. The core minutia is not present in every fingerprint (arch fingerprint classification). The core is not always reliably located in poor quality fingerprint images (such as arch fingerprint classifications), which can lead to matching errors.

Cappelli *et al.* proposed a Minutiae Cylinder Code (MCC) for fingerprint matching [54]. The local minutiae model is based on 3D data structure (cylinder) which is

enclosed inside a cuboid that is divided into cells. Each minutia is represented by its location and its direction and the cylinders are bit oriented. The cylinders are divided into six sectors during the verification process for the purpose of comparing the query and the reference templates. The algorithm offers the advantages of speed, dealing with noisy fingerprint regions, local distortion tolerance, and missing spurious minutiae tolerance. The problem that can occur with MCC is that there are chances that the nearest minutiae might be exchanged due to the absence or spurious minutiae. The MCC approach makes use of the convex hull enlarged with the addition of the three offset variances in order to select only the important portions of the fingerprint. This approach defines the similarity between the two cylinders by making use of the vector correlation measure. It is said that this algorithm can be implemented inside smart cards and it can make use of XOR and the AND operations to compare the cylinders [24].

Table 3.1 Comparison of fingerprint MoC and WSoC algorithms

MoC algorithms	**MoC Algorithms attributes**				
	Database used	*Smart card environment*	*Average EER/ TAR*	*Advantages*	*Disadvantages*
Fast MoC techniques using in-matcher with ISO minutiae template [32]	Fingerprint Verification Competition (FVC) FVC2000 DB1,BD2, DB3, BD4 FVC2002 DB1,BD2, DB3, BD4 FVC2004 DB1,BD2, DB3, BD4 FVC2006 DB1,BD2, DB3, BD4	8bit Microcontroller Unit (MCU),6 kilobyte, 78 kilobyte Electrically Erasable Programmable Read-Only Memory (EEPROM)	<=5.1979 EER%	The method used large data (12 databases) to test the matching accuracy of the algorithm. Large data allows the researcher to test how the matcher performs when it encounters different kinds of fingerprint matching problems (rotation, translation, partial prints and deformation)	The performance of the this method is affected by false minutiae especially in low quality images

ISO/IEC standards for on-card biometric compariso n [33].	FVC2000 DB1, DB2 and DB3 FVC2002 DB1 and DB2	8 bit java smart card	4.3 EER%	The method obtained a good recognition rate and the matching accuracy of this method is comparable to the fingerprint minutiae-based matching algorithm running on the PC	The method computes an off-card template alignment
Embedded fingerprint matching on snart card [21]	FVC2002 DB1-a and DB2-a	64 kilobyte EEPROM and 1kilobyte Random Access Memory (RAM)	5.385 EER%	The method offers the advantages of good recognition rate, simplicity, and speed	The method does precisely compute the core reference minutia in poor quality images
An asymmetri c fingerprint matching algorithm for java card TM [52]	FVC2002-a FVC2002 DB2-b FVC2002 DB1 International DB Hybrid DB	72 EEPROM and 4 kilobyte RAM	3.4 EER%	The method obtained a good recognition rate and exits the matching process as soon as the good average similarity score is found	The method requires lots of computations due to multiple reference minutiae in the template

Fingerprint pre-alighment for hybrid MoC systems [50]	FVC2000 DB1,BD2, and DB3 FVC2002 DB1,BD2, and DB3 FVC2004 DB1,BD2, and DB3	This method does not disclose the smart card environment	83 TAR%	The method can still align the query template even if the core reference point is not found without sacrificing any speed and accuracy. This method also relieves the burned of the smart card in aligning the templates as fingerprint alignment is time and resource consuming	The method is less tolerant to large amount of distortion. This method does not disclose the smart card environment.
An ultra-low fingerprint matching algorithm and implementation on a 32-bit smart card [45]	The database is not disclosed	32 bit processor ,64 kilobyte Read Only Memory (ROM), 8 RAM, and 32 kilobyte EEPROM	This algorithm did not disclose the matching accuracy	This method offers the advantage of speed	This method use an 32-bit smart card which is too expensive for the market

3.6 Summary

This chapter presented different approaches to fingerprint matching. Furthermore, the shortcomings of the existing algorithms were highlighted. Most of the literature reviewed for this study revealed that alignment techniques are not suitable for MoC algorithms because they are computationally intensive and require a lot of memory. The chapter also presented how other researchers filled the gaps in the existing algorithms. The next chapter is going to present the methodology of this research, which is derived from the work of F. Benhammadi and K. B. Bey [21]. F. Benhammadi and K. B. Bey make use of finger codes to compare the fingerprints. Finger codes (minutiae information) make use of binary representation, which is the primary language for the computers, and a distance matrix which leads to matching

speedup and the usage of less memory in a constrained environment of a smart card. The next chapter makes use of multiple reference minutiae to eliminate the use of a core reference point. This is due to inaccuracy of the location of the core reference point in noisy images and absence of a core in arch fingerprint classification [21].

Chapter 4

4. Methodology

In the literature review, different techniques for fingerprint MoC algorithms have been presented. The literature review revealed that the usage of finger codes requires less processing than using the direct information of the minutia (minutia position (x, y) and minutia direction (θ)). This study uses finger codes to implement a proposed fingerprint MoC algorithm. This chapter provides a detailed methodology that is used in this study to achieve its aims and objectives. It includes research instruments and the data evaluation method employed for accessing the accuracy of the proposed fingerprint MoC algorithm.

4.1 Research design

In order to achieve all the objectives which were discussed in section 1.5 of Chapter 1, this research used a quantitative and experimental approach. Quantitative research explains events by collecting analysed numerical data using mathematically based methods [55]. Experimental studies take place in artificial settings, enabling researchers to access the relationship between one variable to another [56]. The experiments will be conducted using code implementation in MATLAB and the fingerprints which are collected from a public database.

MATLAB offers the following advantages:

- it has very good documentation and it is continuously being improved [57].
- MATLAB is an interpreted programming language with various toolboxes which allows the manipulation of complex problems [57].
- MATLAB is reliable and it is optimized to be fast when performing matrix operations [58].

Disadvantage of MATLAB

- Execution of MATLAB programs is slow [59].

Experimental studies offer the following advantages:

- It has control over variables. Hence, it eliminates unwanted variables.

- Manipulate independent variables to simply determine the cause and effect of the problem.
- Tests/Experiments can be repeated with modified algorithms until accurate results are obtained.

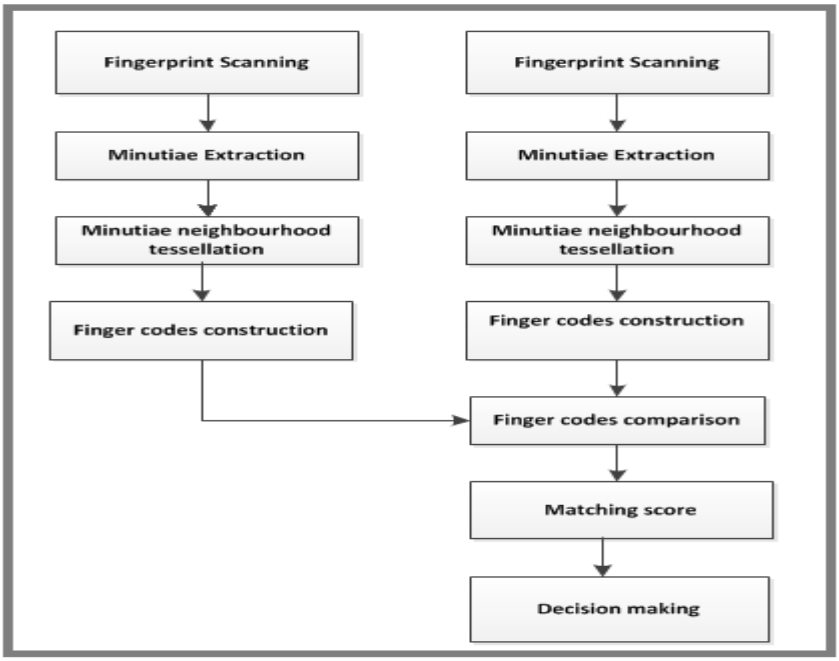
However, the experimental studies have the disadvantage of producing the results from the limited samples or variables that have been used. This is not always as accurate as the real world applications.

A quantitative study has the advantage of dealing with large amounts of data (samples), which can lead to higher probability of accurate experimental results. It also provides exact, quantitative, numerical data [60].

This research design evaluates the proposed algorithm performance with regards to how it handles the rotation, translation, distortion, deformation present in templates, as well as missing and additional minutiae when two fingerprint images from the same finger are compared. Furthermore, the research design involves the process of collecting, analysing, interpreting and obtaining the matching accuracy of the proposed algorithm. The experimental studies were conducted using the following feature measures:

- MoC matching accuracy – how often does the algorithm correctly and incorrectly compare the fingerprints. The algorithm used EER to evaluate or measure the matching accuracy. In addition, the accuracy of the fingerprint matching is often affected by rotation, translation, distortion and deformation in fingerprints.
- Memory allocation – The memory usage of the proposed algorithm to determine whether the proposed algorithm fit into the smart card.
- Time complexity – how long it takes for the algorithm to execute.

Figure 4.1 illustrates all the steps which are used in the proposed methodology.



.

Figure 4.1: Flow diagram of the proposed algorithm

4.2 Research instruments

This section explains all the steps in detail which are used in figure 4.1

4.2.1. Fingerprint scanning

Fingerprint scanning is the first step in Figure 4.1. It explains how the fingerprints were scanned. The experiments were carried out using fingerprint images from the FVC2002 database1-a [61]. FVC2002 is the second international fingerprint verification competition which is designed for fingerprint verification algorithms. It constitutes four fingerprint databases which are collected by using three commercially available scanners. The fourth database was synthetically generated by using SFinGE software. All these databases were used to collect fingerprints.

Table 4.1 illustrates the types of scanners, technology and resolution which were used to collect the four databases (DB).

Table 4-1: Scanners, image size and technologies used for the collection of FVC2002 databases [61]

Databases	Technology	Scanner	Image size - Resolution
Database 1	Optical	Identix touch view	388*374-500dpi
Database 2	Optical	Biometrika	296*560-259dpi
Database 3	Capacitive	Precise biometrics 100SC	300*300-500dpi
Database 4	Synthetic	SFinGE v2.51	288*384-500dpi

Database-1a from FVC2002 contains a total of 800 fingerprints of 100 different fingers, with each finger containing eight impressions of the same finger.

The fingerprints were collected from twenty volunteers. The volunteers were randomly divided into three groups (30 volunteers each). Each group was collected using a different fingerprint scanner from each DB. A total of four fingers which include two index fingers and two middle fingers from both hands were collected in each DB to maximize the difference in fingerprint displacement. No efforts were made to control image quality and the sensor platens were not systematically cleaned. The fingerprints were captured from the volunteer in three distinct sections. Four impressions were captured from four fingers of each volunteer in each session. Throughout the second session were requested to exaggerate displacement (impression 1 and 2) and rotation (impression 3 and 4) of the finger, not to exceed 35 degrees. Throughout the third session, fingers were alternatively dried (impression 1 and 2) and moistened (impression 3 and 4). The fingerprints in each DB were sorted according to the quality index [61] .

4.2.2. Minutiae extraction

The minutiae were extracted using Minutiae Cylinder Code (MCC) feature extraction. This method follows the ridge lines on the grey-scale image, by taking the direction of the fingerprint. The method superimposes a square meshed grid on a grey-scale image to determine a set of starting points. These ridges are kept until they terminate

(a ridge termination is found) or until the ridge divides to form two ridges (ridge bifurcation) [62].

Figure 4.2 below illustrates minutiae points (ridge bifurcation and ridge ending).

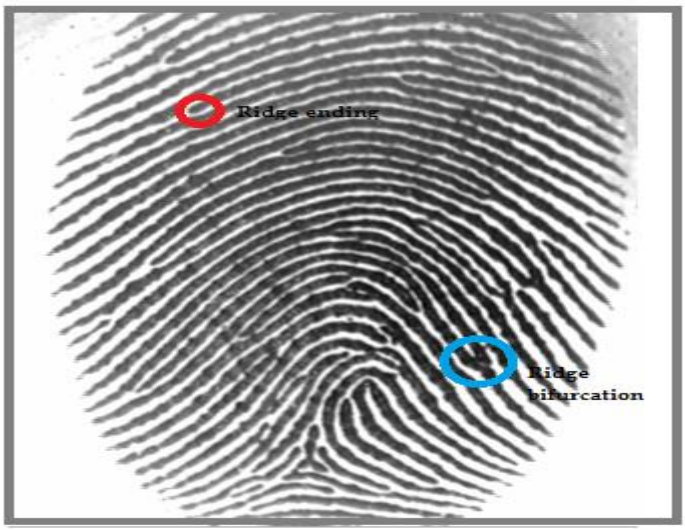


Figure 4.2 : Ridge ending (red) and ridge bifurcation (blue) [63]

4.2.3. Minutiae neighbourhood circular tessellation

Different minutiae neighbourhood circular tessellations were applied to investigate which circular tessellation offers the lowest EER. This was done by using different number of sectors and ridge counts in circular tessellations. The area of each cell in a circular tessellation is the same. The area of each cell in the circular tessellation was obtained using equation 4.1:

$$\boldsymbol{A} = \frac{\boldsymbol{\pi r^2}}{\boldsymbol{s * Rc}} \tag{4.1}$$

where A, r, s, and Rc represent the area of a sector, radius of a circular tessellation, number of sectors, and the number of ridge counts in a circular tessellation respectively. Ridge counts are circular bands in a circular tessellation. In order to evaluate which circular tessellation performs better, EER was calculated using 25 fingers, each finger has 8 impressions. This implies that 200 fingerprints were used to calculate the value of EER. The radius of a circular tessellation is 80 pixels. This is because when a radius of a circular tessellation becomes bigger than 80 pixels it get affected by distortion and when a circular tessellation is smaller than 80 pixels it can only include fewer minutiae which are not distinctive enough to represent

neighbourhood minutiae [64]. Table 4.2 illustrates different circular tessellations using different numbers of sectors and circular tessellation.

Table 4.1 : Different circular tessellations using different numbers of sectors and circular tessellation

Number of sectors	Number of ridge counts	Area of each cell in a circular tessellation	EER (%)
8	6	628.32	13.33
6	8	628.32	9.89
6	4	837.76	7.21
4	6	837.76	9.61
8	5	502.66	4.21
5	8	502.66	7.45
6	5	670.208	3.23

The proposed model used a circular tessellation of six equally sized sectors and five ridge counts because it obtained a lower EER when compared to other circular tessellations. This tessellation obtained an EER of 3.23 using 6 sectors and 5 ridge counts and when the area of each cell in a circular tessellation is the same and 3.02 when the area of the each cell in a circular tessellation is not the same. The area of each cell in the circular tessellation of the proposed algorithm is not the same.

The proposed fingerprint MoC is based on minutia-based algorithm and it does not use the ridge patterns to compare the fingerprints. It only uses the minutiae location and direction of the fingerprint. The proposed algorithm uses the following steps to compute a finger code:

- Find the minutiae which can be used as a reference point in a template. This is illustrated in figure 4.6.
- Circularly tessellate each reference minutia.
- Convert each circular tessellation into binary codes in a template. Each binary string/row in a template is a representation of a reference minutia. This implies that the number rows in the binary code are equivalent to the number of reference minutiae.

- The binary codes are converted into binary maps to construct a finger code.

The proposed methodology makes use of circular tessellation which is constructed from six equally spaced sectors with an angle of sixty degrees in each sector. The tessellation process starts from the minutia orientation while going clockwise to other sectors. The reference minutia is located at the centre of the circular tessellation. The first sector starts at the orientation/direction of the minutia for the purpose of making the algorithm rotation invariant. Figure 4.3 below illustrates 6 sectors (S1, S2, S3, S4, S5 and S6) which are used to construct a circular tessellation, minutia orientation and the reference minutia point.

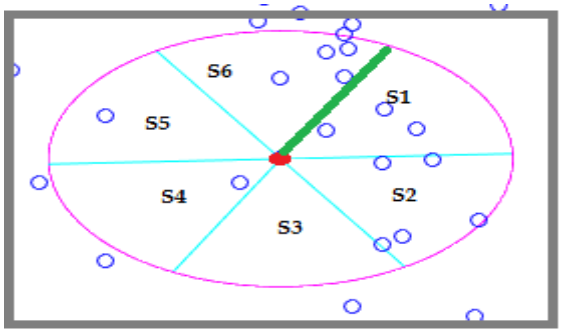


Figure 4.3: Six sectors (S1to S6) with the reference minutia point (red) and minutia orientation with minutiae points (blue)

The circular tessellation has the radius of 80 pixels and is also constructed from five ridge counts. Each minutia point consists of its position (x, y) and its minutia direction (θ). The first ridge (Rc1) count has a distance of 26 pixels from the centre, the second, third, and the fourth ridge count (Rc2, Rc3, and Rc4) have the ridge count spacing of 14.8 and the fifth ridge count (Rc5) has the ridge count spacing of 9.6 pixels. Each circular tessellation has 30 cells, which is made out of 6 sectors and 5 ridge counts (5*6=30). Figure 4.4 (a) illustrates the 5 ridge counts, reference minutia point and the minutia orientation (green).

Figure 4.4 (b) illustrates the cell numbers of the neighbourhood minutia tessellation.

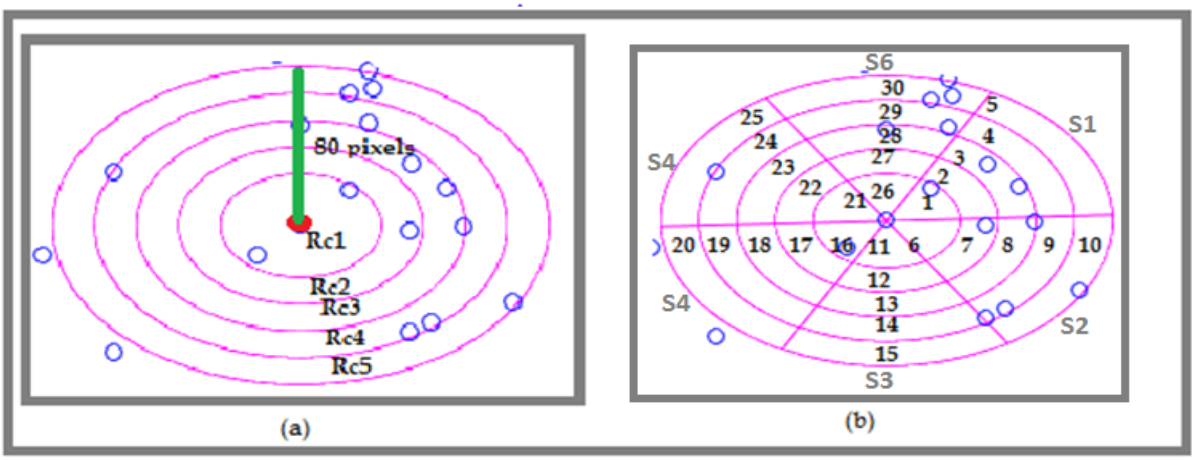


Figure 4.4 : (a) 5 ridge counts and reference minutia point and the minutia orientation (b) cell numbers of the circular tessellation

Figure 4.4 (b) illustrated above portrays a neighbourhood minutiae tessellation of the proposed algorithm. The proposed algorithm uses the radius of 80 pixels because it can cover a reasonable number of minutiae. It can be used to compare the minutiae in the query template and the reference template, as shown above in Figure 4.4 (b). The ridge counts spacing were chosen not to be equally spaced because in cases where ridge counts are equally spaced, the area of a cells of the first ridge count (Rc1) becomes smaller than the area of the cells of other ridge counts (Rc2, Rc3, Rc4 and Rc5). Figure 4.5 illustrates different neighbourhood circular structures with different numbers of sectors and ridge counts.

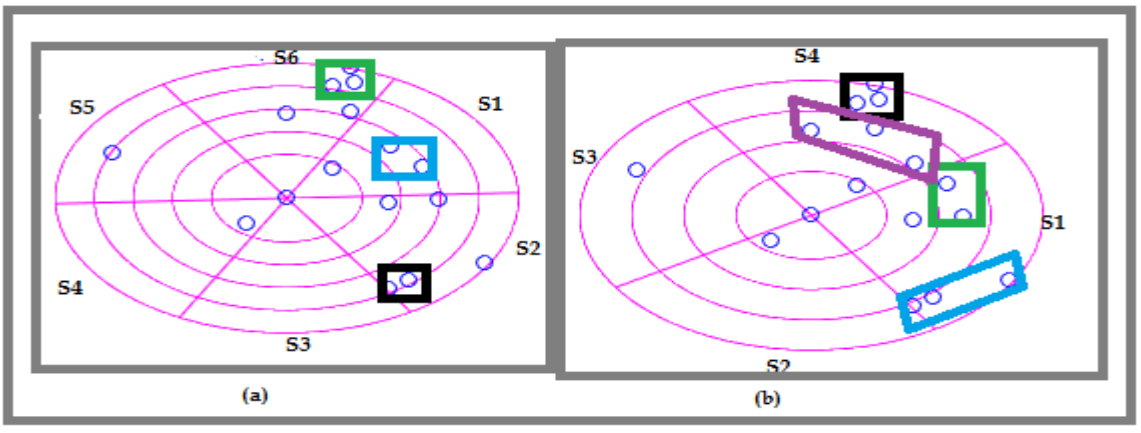


Figure 4.5 : (a) A circular neighbourhood minutiae tessellation with six sectors and five ridge counts (b) a circular neighbourhood minutiae tessellation with four sectors and four ridge counts

The reason for choosing the combination of six sectors and five ridge counts to generate the neighbourhood circular tessellation is because of the following reasons:

- The tessellation minutiae structure (Figure 4.5 (a)) reduces the possibility of obtaining many minutiae in the same cell. This implies that when Figure 4.5 (a) is compared to figure 4.5 (b), it has three cells which have more than one minutia inside the cell (S1 of Rc3 (blue), S2 of Rc4 (black), S6 of Rc5 (green) in Figure 4.5 (a))
- Using a smaller number of ridge counts and sectors produces a neighbourhood circular tessellation that is not distinctive enough to be used for comparison. This implies that figure 4.5 (a) when compared to Figure 4.5 (b) has four cells, which has more than one minutia in the same cell (S1 of Rc3 (green), S1 of Rc4 (blue), S4 of Rc3 (purple) and S4 of Rc4 (black) as in Figure 4.5 (b))

The proposed method is based on neighbourhood minutiae localization binary codes. It checks whether there is a minutia inside each cell in a neighbourhood minutiae tessellation. The presence of the minutia inside a cell is represented by one and the absence of the minutia point inside the cell is represented by zero.

The following array represention is derived from Figure 4.4 (b). The first row is derived from the first sector (s1).

$$\text{Array S} = \begin{matrix} [\,S1 \\ S2 \\ S3 \\ S4 \\ S5 \\ S6] \end{matrix} = \begin{matrix} [\,1\ 0\ 1\ 0\ 0 \\ 0\ 1\ 1\ 1\ 1 \\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 1\ 0 \\ 0\ 0\ 1\ 1\ 1\,] \end{matrix} \qquad (4.2)$$

The refence minutia point of Figure 4.4 (b) is represented by equation 4.2 which is the combination of all the rows in an Array S.

Reference minutia =[1 0 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 1](4.3)

Every minutia which has more than five neighborhood minutiae inside the circular tesselation is used as reference minutia in a template. The reason for choosing this neighborhood minutiae is to avoid obtaining binary strings, which are not distinctive enough to be used for matching. Figure 4.6 illustrates a neighborhood minutiae tessellation with more than five minutiae (red) and a neighborhood minutiae tessellation with less than five minutiae.

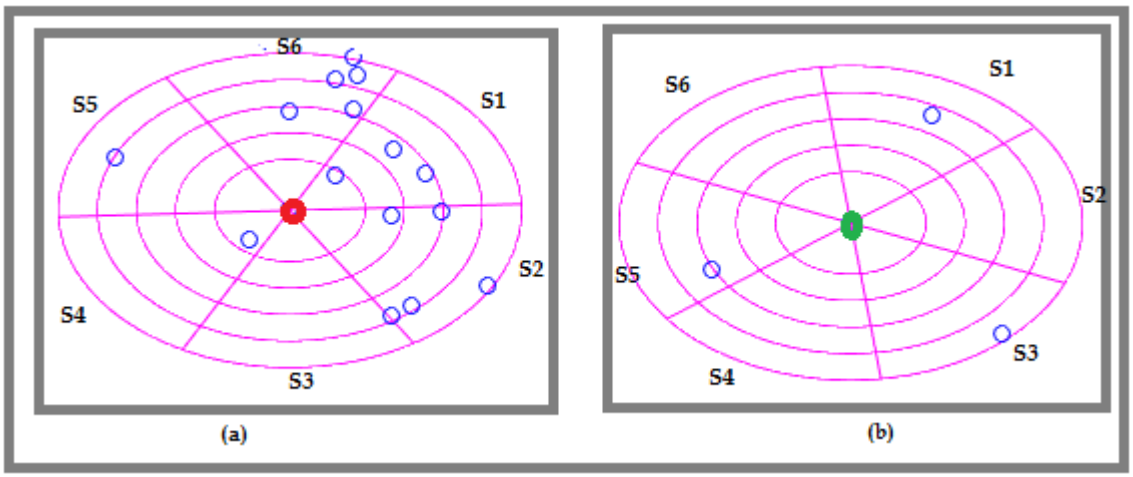


Figure 4.6: (a) A reference minutia point (b) a non-reference minutia point

In the cases when a minutia is in between the ridge counts, it is moved to the first ridge count between the two ridge counts. For example, consider sector 5 in Figure 4.5 (a), the minutia in this case lies between the fourth ridge count and fifth ridge count. This implies that the minutiae will be moved to the S5 in the fourth ridge count.

4.2.4. Finger code construction

A finger code is constructed from a neighbourhood minutiae tessellation. Each reference minutia is represented by binary codes. Figure 4.7 below illustrates the process of constructing the bit maps of a minutia according to the proposed algorithm. Bit maps are cells; each cell contains a colour value. The bit value one is represented by white and the bit value zero is represented by black.

Figure 4.7 illustrates the process of constructing a bit map.

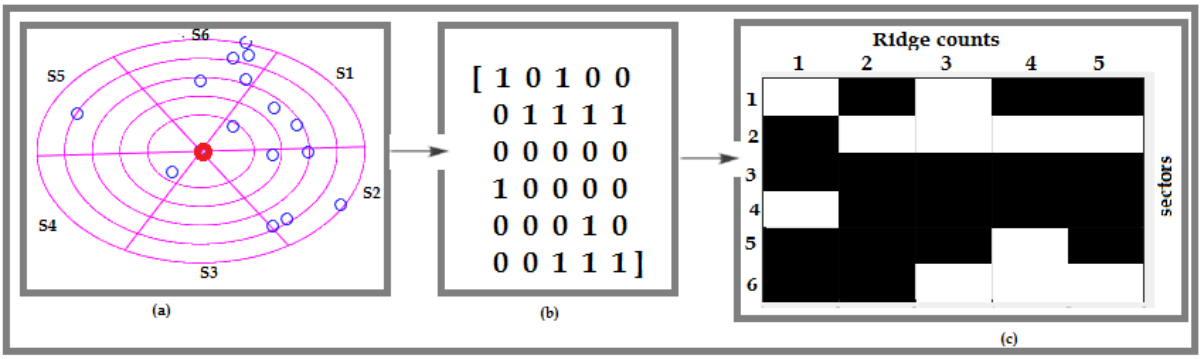


Figure 4.7 : The process of finger code construction

A minutia is represented by the concatenation of all the binary rows (sectors) in figure 4.7(b) which results in the following scalar array.

Minutiae= [1 0 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1] (4.4)

Figure 4.8 illustrates the bit maps of the reference minutiae in figure 4.7.

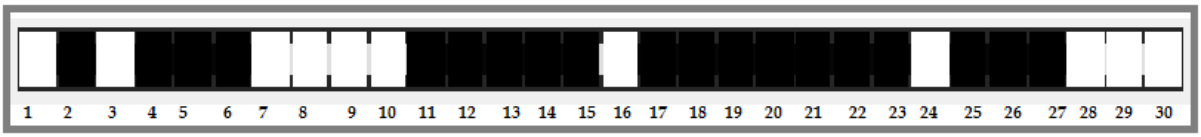


Figure 4.8 : Bit maps of neighbourhood minutiae circular tessellation in Figure 4.7

A finger code is composed of all the bit maps that are constructed from all the reference minutiae in the template. The bit maps above in Figure 4.8 represent a single reference minutia. The number of rows in a finger code is not always the same. They are equivalent to the number of the reference minutiae present in a template. The number of columns in the finger code is always 30 which is the number of the cells in a neighbourhood minutiae tessellation.

Figure 4.9 illustrates a finger code with 16 rows and 30 columns.

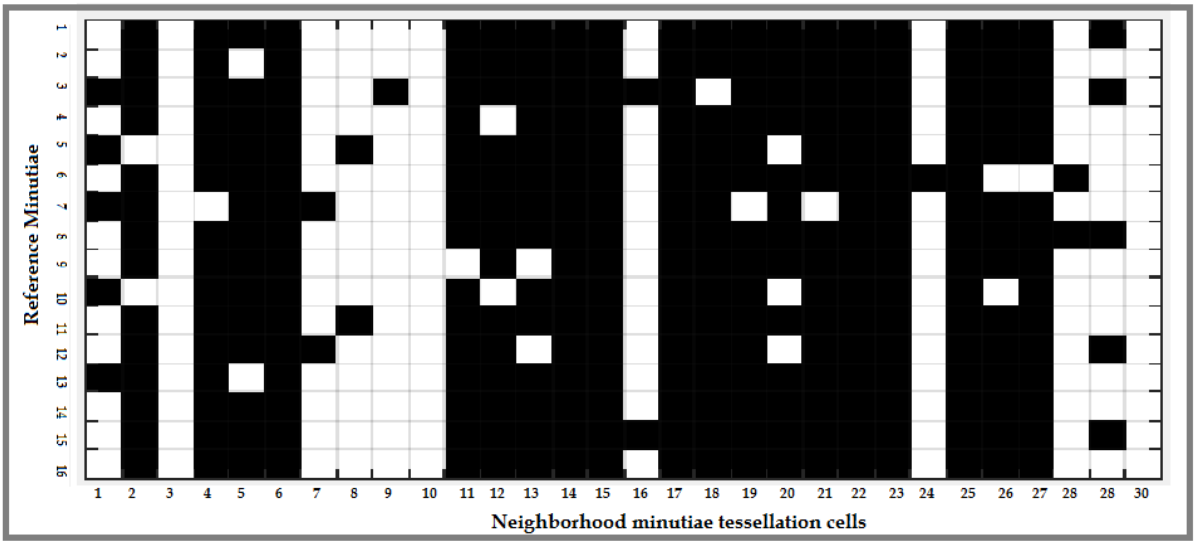


Figure 4.9 : Finger code which is made out of 16 reference minutiae points (16rows)

4.2.5. Finger code comparison

The proposed algorithm uses Hamming distance to compare the finger codes. Each row in a reference finger code is compared to every row in the query finger code. The predefined matching score for pairing the minutiae is 6. This means that, if a matching score between the compared minutiae has a Hamming distance of 6 or less than less 6, they match. Otherwise they do not match. The predefined matching score of 6 is chosen because if a Hamming distance of less than 6 is chosen the matcher tends to be too strict and rejects the genuine finger code. When a Hamming distance of greater than 6 is chosen the matcher tends to be too liberal and accept the imposter's finger code. Let $R = \{R_i, i = 1 \dots t\}$ and $Q = \{Q_j, j = 1 \dots p\}$ be reference finger code and the query finger code respectively. Where t the number of rows in the reference finger code and p is the number of rows in the query finger code. Hamming distance HD_{ij} between R and Q is calculated using equation 4.5.

$$\boldsymbol{HD_{ij}\ R_i\ Xor\ \ Q_i} \qquad (4.5)$$

Boolean exclusive-OR (Xor) measures the number of dissimilarity between the binary code Q_j and R_i.

Each row from a reference finger code is compared with every binary row from the query template.

Figure 4.10 illustrates the reference minutiae comparisons using finger codes rows.

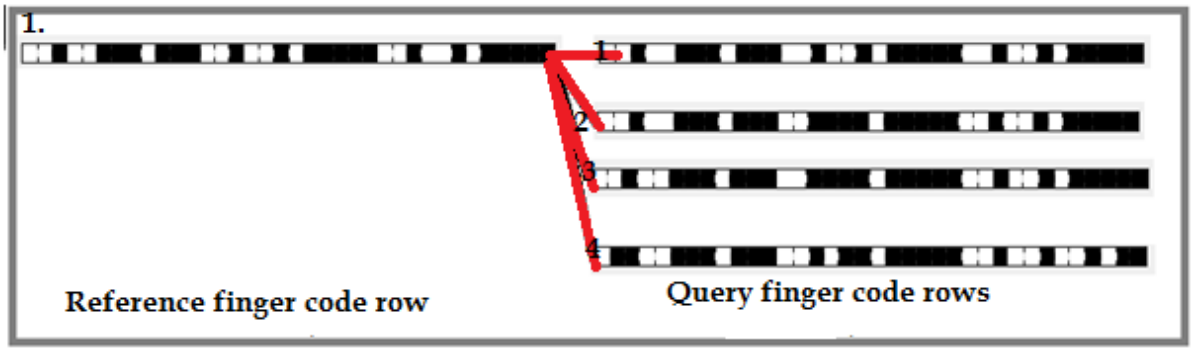


Figure 4.10 : Reference minutiae comparisons using finger codes rows

In figure 4.10 the first row of the reference finger code is compared to all the rows of query finger code to obtain the matching score. Each row in a finger code represents a reference minutia point. This process of comparing the finger code rows is done to the second row of the query finger code up until the last row of the query finger code.

4.2.6. Matching score and decision making

The proposed algorithm uses the following steps to calculate the matching score:

- Calculate the number of matched minutiae.
- Calculate the number of the reference minutiae in the reference template and in the query template using equation 4.5 and equation 4.6.

$$\mathbf{RF_i = J, J = 1, \dots \dots n} \quad (4.5)$$

$$\mathbf{RF_j = K, K = 1, \dots \dots m} \quad (4.6)$$

  Where J=number of rows in a reference finger code and K= number of rows in a query finger code. RF_i is the number of reference minutiae in a reference finger code and RF_j is the number of reference minutiae in a query finger code.
- Find the minimum number of the reference minutiae between the reference finger code and the query finger code using the following condition.

$$if \quad RF_i < RF_j$$

$$MNRF = RF_i \text{ else}$$

$$MNRF = RF_j$$

End;

Where $MNRF$ is the minimum number of the reference minutiae between the reference finger code and the query finger code.

- Divide the number of matched minutiae by the minimum number of minutiae that is obtained in the previous step in order to get the total matching score using the equation 4.6.

$$\boldsymbol{Total\ matching\ score} \frac{\boldsymbol{MM}}{\boldsymbol{MNRF}} \tag{4.7}$$

where MM represent matched minutiae. All the matching score values that are obtained between the reference minutiae in and the query template are stored in an array. The algorithm selects all the scores that are under the threshold in an array to pair the minutiae. A threshold is a value that indicates that the minutiae which are compared either match or do not match. The reference template and the query template do not always have the same number of minutiae due to the inaccuracy of the minutiae extractor. The minimum number of minutiae between the reference template and the query template divides the number of matched minutiae. This is because it is the maximum number of minutiae which can match. For example, if ten minutiae is matched to fifteen minutiae. The maximum of the minutiae that can pair/match is ten minutiae.

The proposed algorithm was derived from the work of F. Benhammadi and K. B. Bey. This relies on core reference minutia to initiate the circular tessellation. It is difficult to precisely locate the core and even when there is a small amount of distortion in the core reference minutia the overall performance of an algorithm is affected. The work of F. Benhammadi and K. B. Bey deploys a circular tessellation of 32 sectors and 16 ridge counts. Each finger code is represented by 512 bits (16*32). The proposed algorithm uses a circular tessellation of 6 sectors and 5 ridge counts in each reference minutia. The proposed algorithm uses a smaller number of sectors and ridge counts compare to the work of F. Benhammadi and K. B. Bey because it uses multiple reference minutiae. Each reference minutiae is represented by 30 bits in a finger code.

4.3 Summary

This chapter presented the methodology that was used to match the finger codes and the research design. It also described how the finger codes were circularly tessellated and how the minutiae information was presented in the finger code. The next chapter is going to present how the methodology was implemented.

Chapter 5

5. Implementations

This chapter is going to discuss the implementation environment of the proposed MoC algorithm for this dissertation. It presents the complete pseudo code for constructing a finger code and matching the finger codes, neighbourhood minutiae localization binary codes structure and time complexity of the proposed MoC algorithm.

5.1. Implementation Environment

The MATLAB code for the proposed algorithm was implemented under the following environment:

- Installed memory (RAM) of 8GB
- Windows 7 Enterprise 64-bit Operating System

5.2. Proposed algorithm

The proposed MoC algorithm creates a function to construct a finger code and a function to match the finger codes.

5.2.1. Pseudo code for constructing a finger code

Table 5.1: T**able for constructing a finger code.**

Constructing a finger code	
Input	Reference fingerprint and query fingerprint.
Output	1. Feature vectors (minutiae points). 2. Circular minutiae tessellation for each reference minutia point. 3. Binary string for each reference minutiae point. 4. Finger code

The proposed algorithm will read the fingerprint from FVC2002/DB1-a database and will extract the feature vectors from the fingerprint. Reference minutiae are used to generate circular minutiae tessellation. The binary output that is generated from each minutiae tessellation is deployed to construct a finger code.

5.2.2. Neighbourhood minutia localization binary codes

The proposed algorithm initiates the process of constructing a finger code by constructing an array A=[$s * Rc$], where s is the number of reference minutiae in the finger code and Rc is the number of cells in each circular minutia tessellation containing only zero elements. It uses every neighbourhood minutiae circular tessellation for each reference minutia point to locate bit value one (1) in the finger code otherwise leave the bit value as zero (0). Table 5.2 illustrates the starting point and the ending point of each sector in a circular minutia tessellation.

Table 5.2 : Starting and end point for each sector

Sector number	**The stating point of the sectors in degrees**	**The ending point of the sectors in degrees**
First sector	0	60
Second sector	60	120
Third sector	120	180
Fourth sector	180	240
Fifth sector	240	300
Sixth sector	300	360

The starting point of the first sector is aligned to each reference minutia orientation of its circular tessellation. This is done this way to make the proposed algorithm rotation invariant. The algorithm identifies sector number, where the neighbourhood minutiae are located by finding the angle between that neighbourhood minutia and the reference minutia point.

Figure 5.1 illustrates the angle between that neighbourhood minutia and the reference minutia point.

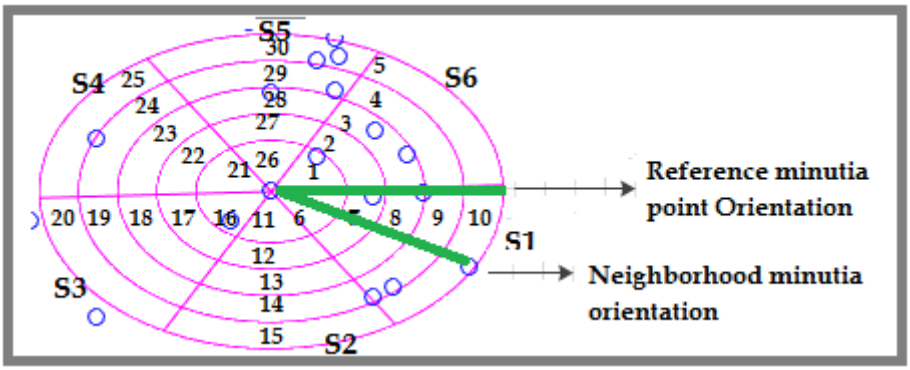


Figure 0.1: illustration of the angle between that neighbourhood minutia and the reference minutia point (green)

For example the angle between the reference minutia and the neighbourhood minutia in cell number 10 is 29 degrees, which means the neighbourhood minutia is in sector one. The proposed algorithm finds the angle between each neighbourhood minutia and the reference minutia point to find in which sectors the minutiae are located. The proposed algorithm identifies the distance between the reference minutia point and the neighbourhood minutia in order to find the ridge count number. Table 5.3 illustrates the starting and the ending point of each ridge count in each circular minutiae tessellation.

Table 5.3 : The starting and the end point of each ridge count in each circular minutiae tessellation.

Ridge count number	**The stating point of the ridge count in pixels**	**The ending point of the ridge count in pixels**
First ridge count	0	26
Second ridge count	26	40.8
Third ridge count	40.8	55.6
Fourth ridge count	55.6	70.4
Fifth ridge count	70.4	80

When the algorithm detects a minutia inside the sector, it calculates the distance of the reference minutia point to that neighbourhood minutia point in order to obtain the

ridge count number. Whenever the sector number and the ridge count number of the neighbourhood minutia are found, the algorithm places one (1) in a finger code. The algorithm places zero (0) in the cell numbers where the minutiae are not found.

5.2.3. Matching algorithm between two finger codes

After circular tessellating a neighbourhood minutiae tessellation, the finger code is constructed from the circular tessellation. Then the reference finger code is compared to a query finger code.

Table 5.4 presented below illustrates the matching algorithm conceptualized for matching finger codes.

Table 5.4 : Finger code comparison

	Finger code comparison	
	Input	Reference finger code and query finger code
Input-process-output model	Processing	1. Take the minutia i (binary string) from the reference finger code. where i=1,2,3,… and m_1, m_2, m_3, m_i is reference minutiae numbers for the reference finger code. 2. Search for the pairing minutia j in the query finger code for m_i where j=1,2,3,…and m_1, m_2, m_3, m_j is reference minutiae numbers for the query finger code. 3. If a pairing reference minutia is found: Remove the reference minutia to avoid matching the same minutiae more than once. 4. Calculate all the paired minutiae from the finger codes. 5. Calculate the percentage match.

The Hamming distance algorithm starts calculating the XOR value between the binary string m_i to the binary string m_j using equation in 4.5.

The algorithm finds the Hamming between two binary strings. This algorithm finds the Hamming distance between the first row of each binary string i in the reference template and every binary string j in the query template. When all the Hamming distances are obtained, the algorithm creates an array of all the Hamming distances which are obtained sequentially. For example the Hamming distance values between the reference minutia in the reference template and each reference minutia in the query template which has 25 reference minutiae has an array of $[1, m_j]$.

The algorithm selects the minimum Hamming distance from all the Hamming distances that are obtained. In the proposed algorithm, the first reference minutia in reference template matches with the sixth reference minutia in the query template. The maximum Hamming distance of six or less than six indicates a match. In this algorithm every binary string indicates a minutia. As soon as a match is found the corresponding binary string is cancelled so that it does not have to be matched with another minutia again.

5.3. Time complexity

Time complexity is a way to formally measure the amount of time used to execute the program. Big O Notation is going to be used to evaluate the time complexity of the proposed algorithm and it is a way of measuring how the program/algorithm scales as the amount of time increases. It characterizes function according to the growth rates**.**

- O - Represents the function/algorithm that is under evaluation.
- N - Represents the number of elements that are present in the function.

Time complexity is mainly affected by the following list of items:

- Operations (+,/,-,*)
- Comparisons (<,>,==)
- Looping (such as for loop)
- Function calls
- Variables

5.3.1. Big O notation for constructing finger codes

In the proposed algorithm the extraction of feature vectors is described by the function $f(n)$ =1. Circularly tessellating each reference minutia is described by the function of $f(n)= n$. Circular tessellating each feature vector into sectors is described by the function of $f(n)= n$. Finding all the neighbourhood angles around each reference minutia, and creating a binary string for each reference minutia point is described by the function $f(n)= n$. The entire function for constructing a finger code is described by the function $f(n)= n$. Thus the complexity time of the algorithm is linear. Therefore it gives an indication that the proposed algorithm will be fast and hence makes it suitable to be used in MoC. The algorithm for constructing the finger code is linear because it uses the following steps:

- The proposed algorithm circular tessellate each reference minutia into sectors by calculating the angle of each sector using equation 5.1, calculating the starting point of each sector using equation 5.2, calculating the ending point of each sector using equation 5.3.

$$\boldsymbol{Angle\ sector} = \frac{\boldsymbol{\pi}}{\boldsymbol{number\ of\ sectors}} \quad (0.1)$$

$$\boldsymbol{start\ sector\ in\ s_n} = \left(\frac{\boldsymbol{\pi}}{\boldsymbol{number\ of\ sectors}} * \boldsymbol{s_n}\right) - \boldsymbol{60} \quad (0.2)$$

Where s_n is sector number

$$\boldsymbol{End\ sector\ in\ s_n} = \left(\frac{\boldsymbol{\pi}}{\boldsymbol{number\ of\ sectors}} * \boldsymbol{s_n}\right) \quad (0.3)$$

s_n represents a sector number where $n = 1,2,3,4,5,6$ and number of sectors is 6.

The proposed algorithm uses a constant time complexity to tessellate and find the angle of each sector, a starting point, and the ending point for each sector.

The proposed algorithm uses code in Figure 5.2 to find the starting point, and the ending point for each sector.

```
p=6;
p=linspace(0,6,p+1);
Start_sector=(0+2*pi*(p))/6;
End_sector=(2*(p+1)*pi)/(6);
Start_sector( : ,7)=[];
End_sector( : ,7)=[];
```

Figure 0.2 : Source code for obtaining the starting and the ending point of each sector

- The proposed algorithm circularly tessellates each reference minutia into ridge counts by inputting the radius of each ridge count using the following code.

```
r=linspace(26,80,5);
teta=linspace(0,2*pi,360);
 plot(x,y,'m');
```

Figure 0.3 : Radius of each ridge count in a circular tessellation

The time complexity of this algorithm is constant.

- The proposed algorithm uses the following code to calculate Euclidean distance and the angle between the reference minutiae and their neighbourhood minutiae.

Figure 4.5 illustrates a code for generating the finger code

```
    for i=1:1:row;

            point1_x = xiiiii -  origin_X;
            point1_y = zeros(1,row);
            Point1X(1, : )=  point1_x;

            Point2_x = origin_x - origin_x(i);

            Point2_y =  origin_y - origin_y(i);
            FimalXY(i, : ) = (Point2_x);
            FimalYX(i, : )=(Point2_y);

            Magnitude_point1=sqrt( point1_x ).^2+(point1_y).^2;
            Magnitude_point2=sqrt((FimalXY).^2+(FimalYX).^2);
            Magnitude_point2(Magnitude_point2==0)=NaN;


            Norm_point1_x=point1_x/Magnitude_point1;
            Norm_point1_y=-(point1_y/Magnitude_point1);
            NP1( : , : )= Norm_point1_x;
            NP2( : , : )= Norm_point1_y;
            Norm_point2_x=FimalXY./Magnitude_point2;
            Norm_point2_y= FimalYX./Magnitude_point2;

            d=find(isnan(Norm_point2_y));
            Norm_point2_y(d)=0;

            Dot_p1p2=-Norm_point2_x;
            det=(-Norm_point2_y);
            circleAngle= mod(atan2(det,Dot_p1p2),2*pi);
    end
```

Figure 0.4: source code for calculating Euclidean distance and the angle between the reference minutiae and their neighbourhood minutiae.

The time complexity for calculating the Euclidean distance and the angle between the reference minutiae and their neighbourhood minutiae is linear.

The proposed algorithm converts the circular tessellation into binary codes. This is done by searching for minutiae in sectors and ridge counts. The proposed algorithm uses equation 5.4 to check if the minutia is inside the circular tessellation.

$$x^2+y^2 =<80 \qquad (0.4)$$

If the Equation 5.4 is satisfied, this implies that the minutia is inside the circular tessellation. Otherwise the minutia is out of the circular tessellation. The proposed algorithm uses the code in figure 5.5 to check in which cell the minutia is located in a circular tessellation.

Figure 5.5 illustrates a code which checks in which cell the minutiae is located

```
for l=1:size(circleAngle1,1)
    %findind in which sector the minutia is located
S1Angles=find(circleAngle11<=End_sector(1)&circleAngle11>Start_sector(1));
S2Angles=find(circleAngle11<=End_sector(2)&circleAngle11>Start_sector(2));
S3Angles=find(circleAngle11<=End_sector(3)&circleAngle11>Start_sector(3));
S4Angles=find(circleAngle11<=End_sector(4)&circleAngle11>Start_sector(4));
S5Angles=find(circleAngle11<=End_sector(5)&circleAngle11>Start_sector(5));
S6Angles=find(circleAngle11<=End_sector(6)&circleAngle11>Start_sector(6));
%finding in which ridge count the minutia is located
index1=Magnitude_point(S1Angles);
index2=Magnitude_point(S2Angles);
index3=Magnitude_point(S3Angles);
index4=Magnitude_point(S4Angles);
index5=Magnitude_point(S5Angles);
index6=Magnitude_point(S6Angles);
end
```

Figure 0.5: Source code for finding in which cell the minutia is located in a circular tessellation.

5.3.2. Big O notation for matching finger codes

Loading the finger codes in the matching algorithm is described by the function $f(n)$=1. Searching for the pairing minutia is described as a function of $f(n) = n$. Calculating all the paired minutiae and decision-making uses $f(n)$ =1. The entire complexity time of finger codes matching program is described by $f(n) = n$. This indicates that the functions for matching the finger codes is also fast and concludes that the time complexity proposed MoC algorithm is linear.

In order to make the matching algorithm linear the proposed algorithm avoids making the proposed algorithm quadratic by finding the maximum number of rows which can be present in a finger code ($FcodeMax_{rows}$). The proposed algorithm extends a reference finger code with rows of zeros to add up the rows in the finger code to the maximum number which can be found in a finger code. This is because the matcher uses the single loop to loop through the rows in the query finger code and the matcher does not use loop to loop through rows in the reference finger code to avoid a nested loop. Hence the matcher extends the reference finger code with rows of zeros to add up the rows in the finger code to the maximum number which can be found in a finger code.

The proposed algorithm extends the reference finger code to make all the rows in a finger code add up to $FcodeMax_{rows}$. The proposed algorithm declares each row in a finger code using the following code:

```
for i=1:size(extendedReferenceFingerCode,1)
    eval(['row' num2str(i) '= extendedReferenceFingerCode (i, : );'])
end
```

Figure 0.6: Source code for declaring each row in a finger code

The time complexity of the source code for declaring each row in a finger code is linear.

The proposed algorithm used a reference finger code with 20 rows. The last 10 rows are rows with binary strings of zeros.

The proposed algorithm uses the for loop to match the reference finger codes and the query finger code.

The proposed algorithm uses the following code to compare the reference finger code and each query finger code.

```

for  c=1:size(b,1)
    % loop through every row of a query template using the index c
   QueryFingercodeRows=QueryFingercode (c,1:size(QueryFingercode,2));
HammingDistance1=xor(Row1, QueryFingercodeRows);
HammingDistance2=xor(Row2, QueryFingercodeRows);
HammingDistance3=xor(Row3, QueryFingercodeRows);
HammingDistance4=xor(Row4, QueryFingercodeRows);
          .
          .
          .
          .
HammingDistance30=xor(Row30, QueryFingercodeRows);
end
```

Figure 0.7: The source code to compare the reference finger code and each query finger code

After obtaining the Hamming distances, the proposed algorithm eliminates all the Hamming distance which was obtained using rows with binary strings of zeros in the reference template and stores all the Hamming distance in an array. The proposed

algorithm goes through the array to select all the Hamming distances which are less than 6. Immediately the matcher finds a Hamming distance that is less than 6, it cancels the entire row and the column from where that Hamming distance was found. This is done to avoid pairing the same minutia more than once. The proposed algorithm starts by finding the Hamming distance of zero first between the reference minutiae. Then find the Hamming distance of 0 up until 6 between the reference minutiae using the following code.

```
array=[HammingDistance1;HammingDistance2,HammingDistance3,.....,HammingDistancen];
[row1,column1]=find(array==0);
for ii=1:size(row1,1)
%if the row and the column of where array  is zero is found replace that row and column by Nan
    array(row1 (ii), : )=Nan;
    array( : ,b(ii))=Nan;
end
```

Figure 0.8: Source code for finding index of where the Hamming distance is under the threshold was found

The proposed algorithm uses the same code for the Hamming distance from 1 up until 6. Then the proposed algorithm adds all the matched minutiae and divides it by the minimum number of minutiae between the reference template and the query template to get a matching score.

5.3.3. Analysis of best and worst scenario

The time taken to compare a fingerprint depends on the number of minutiae present in the fingerprint. The algorithm will experience a worst case scenario when it is matching the finger codes with more minutiae and best case scenario with when fewer minutiae are compared.

5.4. Summary

This chapter presented the implementation environment of the proposed algorithm and the pseudo code that was used to construct and to match the finger codes. It also describes how the proposed algorithm uses the circular minutiae tessellation to identify binary structure in the finger code and calculate the time complexity of the proposed algorithm using the Big O notation. The following chapter is going to describe how the proposed algorithm performs in various challenges of fingerprint

matching. It will also describe the verification and validation of the proposed algorithm.

Chapter 6

6. Verifications and validations

The purpose of this research was to develop an accurate fingerprint MoC algorithm which is fast. This chapter portrays how the algorithm deals with challenges found in fingerprint matching algorithms (Rotation, translation and distortion, as well as false and missing minutiae in the template). The chapter also verifies and validates the implementation of the proposed fingerprint MoC algorithm. This chapter will also give the overall matching accuracy of the proposed algorithm.

6.1. Accuracy evaluation and minutiae pairing

The accuracy of a matcher can be affected by the following common errors:

- Failure to correctly pair the corresponding minutiae between the query template and the reference template which leads to false non-match errors. This is because of the inconsistencies and variability in the same finger which is captured in different instances.
- Incorrectly pairing minutiae from different fingers which lead to false match errors.

False non-match errors are caused by deformation of fingerprints during fingerprint acquisition which causes the fingerprint to be inconsistent in all instances. False match errors are caused by similar features which are obtained between different fingers and false minutiae. The proposed algorithm used the threshold of 6 to pair the minutiae to compensate for fingerprint challenges which are found in fingerprint verification systems. This implies that if the Hamming distance between the reference minutiae and query minutia is 6 or less, the minutiae will pair, otherwise they do not pair. Challenges found in fingerprint matching involve rotation, translation and distortion, as well as false and missing minutiae in the template.

6.1.1 Translation between minutiae

The proposed algorithm is translation invariant because even if the location and the orientation of the minutiae change, the reference minutia will still have the same neighbourhood minutia structure. The proposed algorithm is proved to be translation

invariant based on the experimental results. Figure 6.1 illustrates how the algorithm performs when there is a translation difference between different impressions of the same finger during minutiae paring.

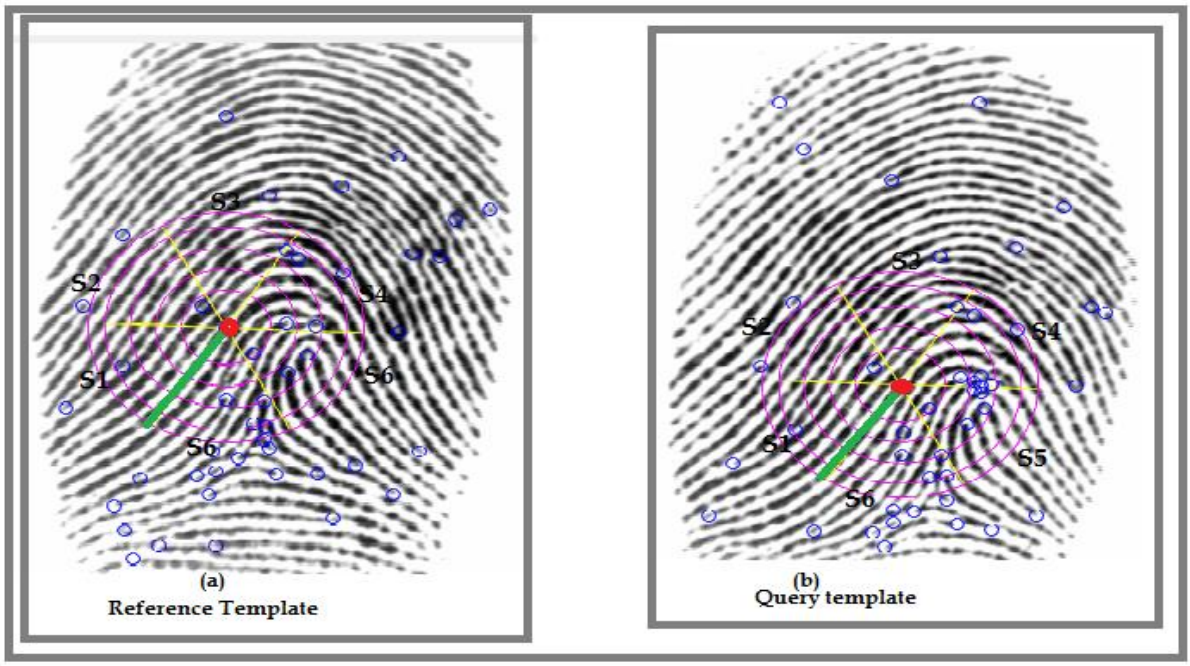


Figure 0.1: (a) Reference minutia (red) which is located at point (157,198,122.34°) (b) Reference minutia (red) which is located at point (208,244,122.34°)

The reference minutia in Figure 6.1 (a) is located at point (157,198,122.34°) and the reference minutia in Figure 6.1 (b) is located at point (208,244,122.34°). The angle of the two minutiae is the same from the minutiae file. The first sector starts from the minutia orientation in the circular minutiae tessellation (green). The translation difference between the two circular minutiae is (51, 46.0°). This algorithm is translation invariant because it uses multiple reference minutiae and each reference minutia is surrounded by the same minutiae neighbours with the same Euclidean distances between minutiae neighbours irrespective of whether the reference minutia is translated. The Euclidean distance between the neighbours can only vary when the fingerprints are distorted. Hence the translation of the minutia does not affect the accuracy of the matcher. However, distortion in fingerprints affects the matching accuracy of the proposed algorithm. The reference template and the query template from Figure 6.1 are two impressions of the same finger. The Hamming distance between reference minutia in Figure 6.1(a) and the reference minutia in Figure 6.1(b) is two. This Hamming distance value indicates that the two reference minutiae

perfectly match. The Hamming distance value is caused by deformation (distortion). In addition, the translation of the templates does not affect the matching accuracy.

6.1.2. Partial prints/incomplete prints

Matching partial prints is a challenge especially when there is a small overlap between the two templates.

Figure 6.2 illustrates the same reference minutia from different impressions of the same fingerprint with different number of neighbourhood minutiae.

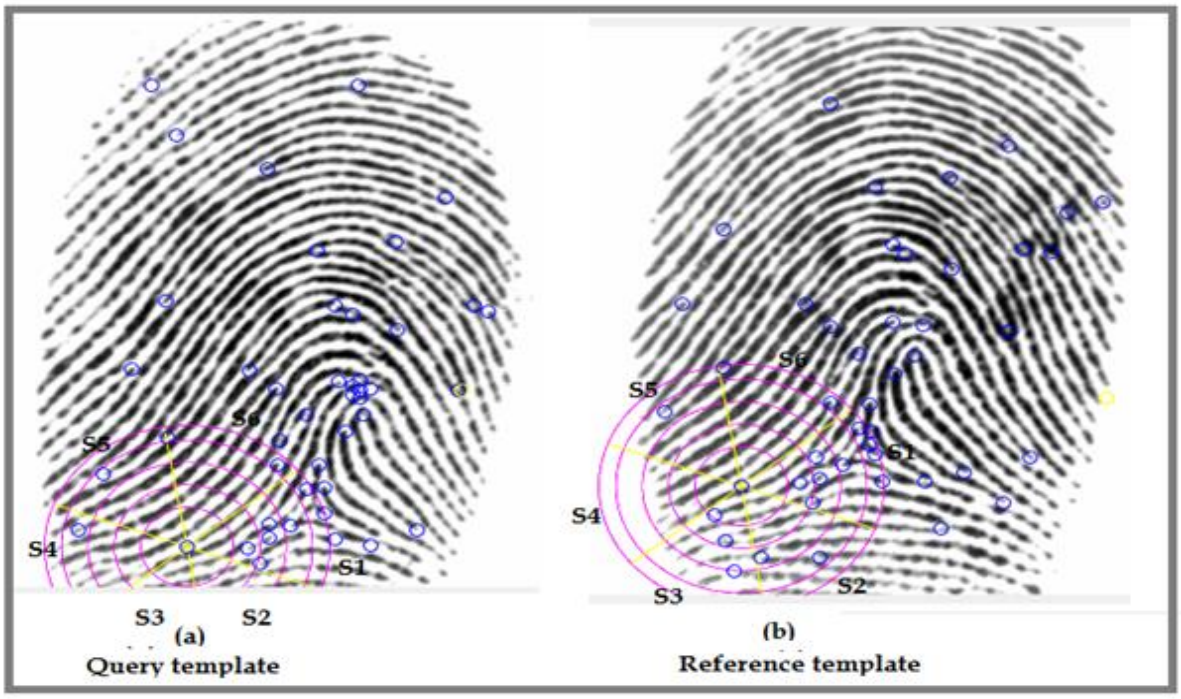


Figure 0.2: Query template with partial prints

In Figure 6.2, the reference template does not produce the same neighbourhood minutiae tessellation when comparing to the query template. This is due to partial prints found in the query template. The Hamming distance between the two templates is 5, which indicates a match between the reference minutiae. The Hamming distance between the two reference minutiae is caused by the following reasons:

- S1 and S2 in the query template do not have all the neighbourhood minutiae that are found in the reference template.

- S4 in the reference template does not have a minutia which is present in the query template. This is as a result of the ridges that are not captured by the sensor during the enrolment phase.

In addition, all the minutiae that appear in the query template do not appear in reference template and vice-versa. The reference minutiae are correctly paired because the majority of the minutiae that appear in reference template are also in the query template. The proposed algorithm can only tolerate partial prints to a certain degree. It fails to pair the minutiae when there is a small overlap between the neighbourhood minutiae circular tessellation.

6.1.3. Distortion

Distortion is the change in the impression of the template which makes the template appears different. The proposed algorithm used more than one reference point to cater for distortion. In the algorithm, the effect of distortion in the acquired template (query template) results in the minutiae being located in a different cell number as compared to the reference template in the generated neighbourhood minutiae tessellation. This may result in failure to pair with the corresponding minutiae, which leads to false non-match errors.

Figure 6.3 illustrates two different impressions of the same finger displaying the effect of distortion.

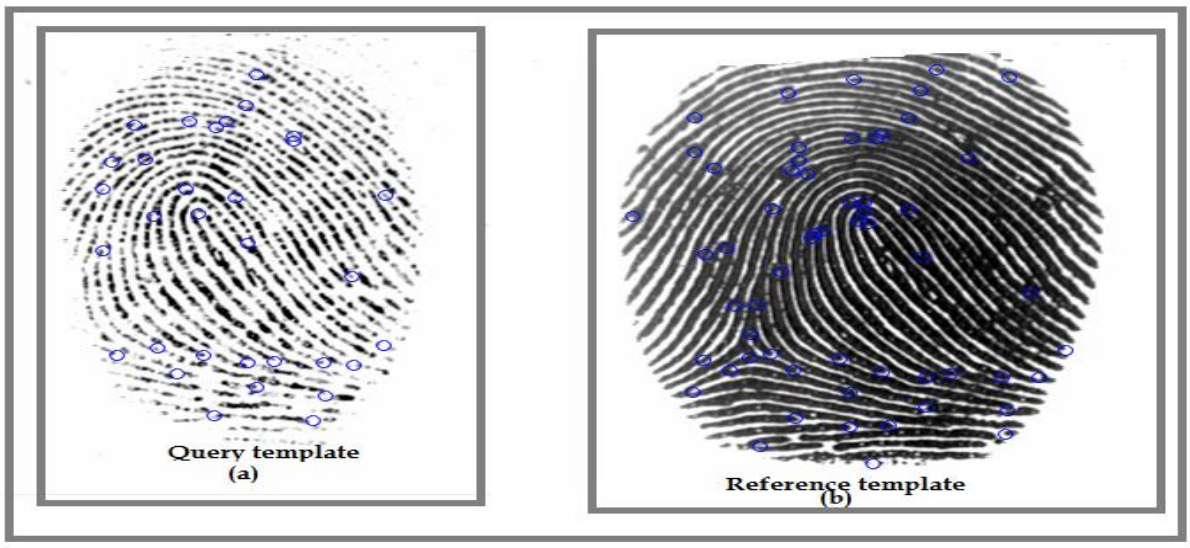


Figure 0.3: Different impressions of the same finger with the effect on distortion in the query template.

In Figure 6.3, the query template has randomly disturbed minutiae location and minutiae orientation due to the effect of distortion. In this particular case, the reference template is wider than the query template. This introduces Euclidean distance change between minutiae because of distortion.

Figure 6.4 illustrates circular tessellation of the same reference minutia using different impressions of the same finger.

Figure 0.4: Circular tessellation of the same reference minutia in different impressions of the same finger.

The Hamming distance between the circular minutiae tessellation in figure 6.4 (a) and in figure 6.4 is 5. These two circular tessellations from figure 6.4 used the same reference minutia from different impressions of the same finger to generate circular tessellations. A Hamming distance of 5 indicates that the reference minutia in the query template pairs with reference minutia in the reference template. This is because the proposed algorithm uses the threshold of hamming distance of 6 to pair the minutiae. This Hamming distance value is obtained due to the effect of distortion. The proposed algorithm can cater for distortion up to a certain degree. It fails to pair minutiae for templates with large amounts of distortion.

6.1.4. Rotation, missing and false minutiae

Comparing different impressions of the same finger with different rotation angles significantly affects the matching accuracy. In order to avoid this, the proposed

algorithm initiates the neighbourhood minutiae tessellation at the orientation of the reference minutia. This means that the first sector of the circular tessellation starts at the orientation of the reference minutia. Hence the proposed algorithm is rotation invariant.

Figure 6.5 illustrates two impressions of the same finger with different rotation angle.

Figure 0.5 : Two impressions of the same finger with rotation difference of 25.53 degrees

Figure 6.6 illustrate circular tessellation of the same reference minutia using the same finger of different impressions.

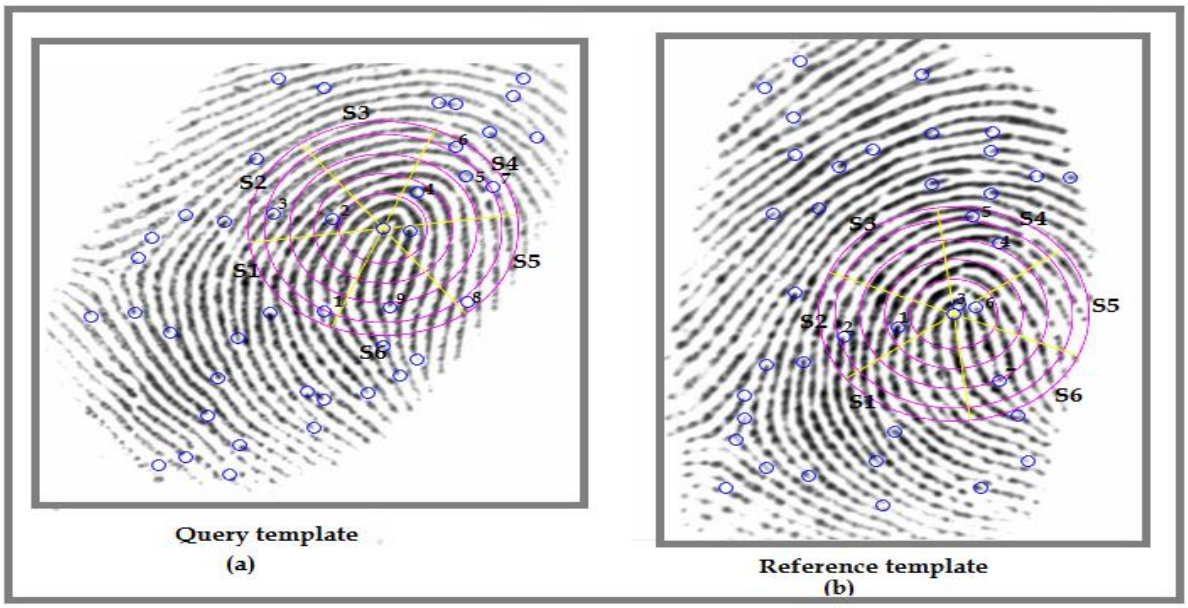


Figure 0.6 : Circular minutiae tessellations of the same reference point using the templates in Figure 6-5

In figure 6.6, the Hamming distance between the reference minutiae in the reference template and the query template is 4. This indicates that the reference minutia from the query template pairs with the reference minutia from the reference template. This Hamming distance value between the two reference minutiae is present due to the following reasons:

- Minutiae extractor failed to extract the ridge ending in S1 of Rc5 in the reference template.
- Minutiae extractor failed to extract the core reference point in S4 of Rc1 in the query template.
- Minutiae extractor failed to extract the ridge bifurcation in S4 of Rc2 in the reference template.
- The ridge line which creates ridge ending (minutia 8) in the query template does not exist in the reference template.

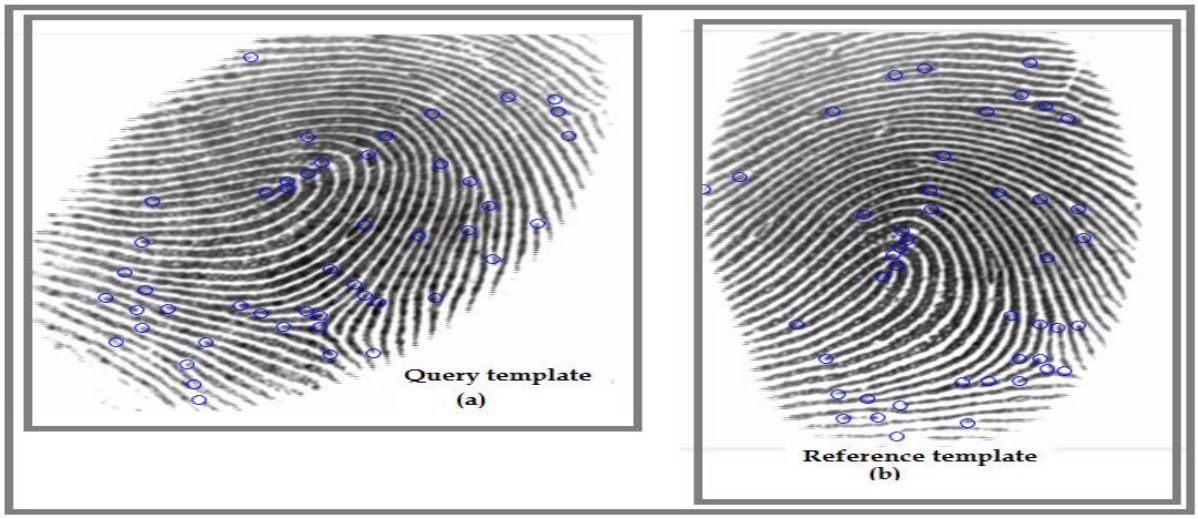


Figure 0.7 : Two impressions of the same finger with different rotation difference of 37 degrees

Figure 6.8 illustrate circular tessellation of the same reference minutia using the same finger of different impressions.

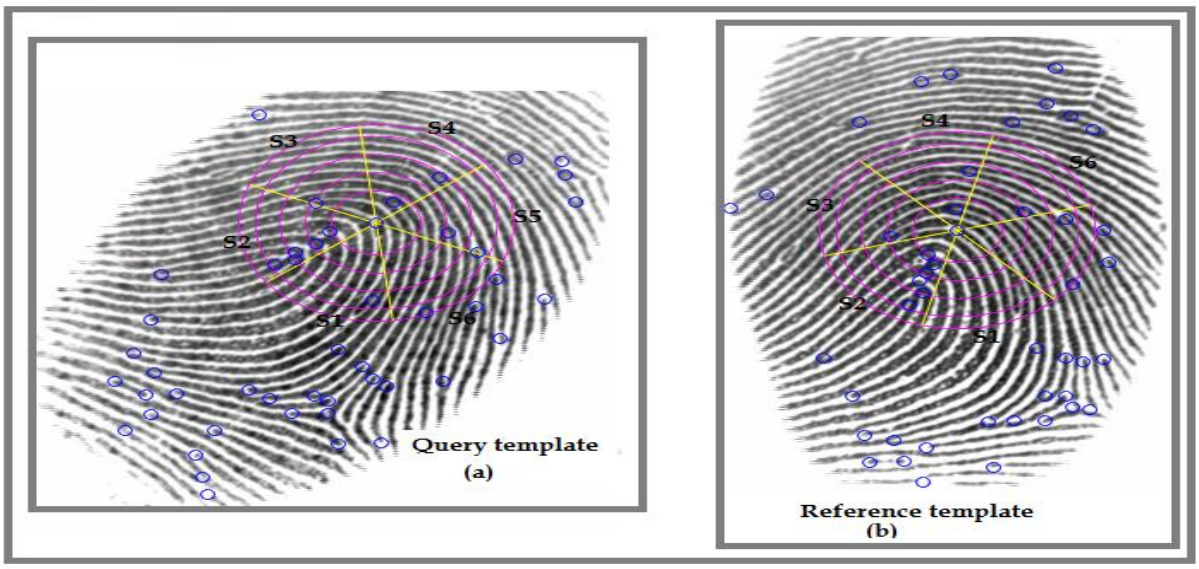


Figure 0.8 : Circular minutiae tessellations of the same reference point using the templates in figure 6.7

In Figure 6.8, the two circular minutiae tessellation of the reference minutiae from the two templates indicate that the reference minutiae correspond with a Hamming distance of 4. This Hamming distance value is present due to the following reasons:

- False minutia point is extracted in S1 of Rc4 in the query template.
- Distortion- the ridge bifurcation in S6 of Rc5 in the query template is outside the circular minutiae tessellation in the reference template.
- The distances between the neighbourhood minutiae are different due to distortion.

This concludes that minutiae extractor errors (missing and false minutiae) and distortion affect the matching accuracy of the proposed algorithm. Furthermore, it is also ascertained that the proposed algorithm rotation is invariant**.**

6.2. Speed of execution

Although the proposed algorithm uses more than one reference minutia point to generate circular minutiae tessellations, it does not compute an algorithm which selects reference minutiae. Since the proposed algorithm uses the minutiae which

are already extracted by the terminal, it only uses the minutiae that are distinctive enough to be used for matching to reduce computations.

Figure 6.9 illustrates minutiae that are discarded during minutiae comparison.

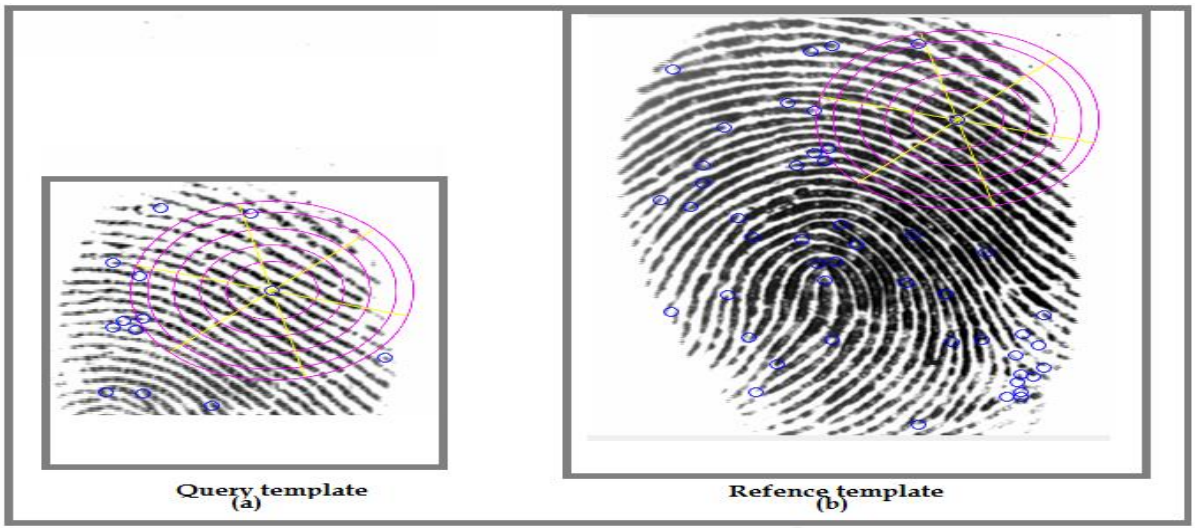


Figure 0.9 : Discarded minutiae during minutiae pairing

In Figure 6.9, some of the minutiae which appear in both templates have neighbourhood minutiae tessellations which have few minutiae. These minutiae cannot be used as reference minutiae, because this results into majority false pairing of minutiae. The proposed algorithm only uses minutiae points which have more than five neighbourhood minutiae. The algorithm is advantageous on matching speed because it reduces the computation by not developing an algorithm which has to compute a reference point before it generates a circular minutiae tessellation unlike a core-based fingerprint algorithm which was proposed by Bey.

Table 6.1 illustrates the percentage of False Non-Match Rate (FNMR) and the causes of FNMR

Table 6.1: FNMR analysis

FNMR analysis	
Fingerprints problems	**FNMR percentage**
Partial prints	0.91
Distortion	2.04
Missing and false minutiae	0.49
Partial prints and distortion	0.82
Distortion, missing and false minutiae	0.56
Partial prints, distortion, missing and false minutiae	0.67

These results show that the proposed algorithm is mainly affected by distortion in templates. The FMR is caused by the similarity in minutiae structure in the different fingerprints.

6.3. Performance evaluation

The performance of the proposed fingerprint matching algorithm is evaluated using the False Match Rate (FMR) and the False Non-Match Rate (FNMR). As explained in previous chapters, False Non-Match Rate is the probability of genuine attempts being wrongly not matched and FMR is the probability of the imposters being incorrectly matched. The proposed fingerprint matching algorithm uses EER to assess the accuracy of a matcher. EER is the error rate where FMR and the FNMR are assumed to have the same value. The FMR and the FNMR are predetermined by the threshold. The threshold[2] is defined as a pre-requisite to decide whether the fingerprint should match or not match. The matcher may prefer to have more FNMR than FMR or vice versa, depending on the type of the application the algorithm is used for. The proposed algorithm has a Failure-To-Enrol (FTE) rate of 5%. FTE is the number of templates which were not successfully enrolled due to absence of reference minutiae points or poor quality fingerprint images. The proposed algorithm uses the True Acceptance Rate (TAR) and the True Rejection Rate to obtain the FMR and FNMR. The True acceptance rate (TAR) is the percentage of the times that

[2]The threshold is a specific number that is set to indicate that a query template has to be accepted or rejected.

the system (correctly) verifies a true claim or identity. TAR is the synonym for Genuine Acceptance Rate (GAR).

True Rejection Rate (TRR) is the percentage of times a system (correctly) rejects a false claim. The proposed algorithm used ith fingers, each finger has jth impressions $i = 1,2,3 \ldots \ldots 100$, $j = 1, \ldots 8.$, where T represent a reference template and I represent the query template.

The algorithm uses the following steps to calculate the TAR and the TRR:

- TAR - Each fingerprint is compared to its remaining impression, avoiding the symmetric comparison (if the first finger of the first impression is compared to the first finger of the second impression, the first finger of the first impression is not supposed to be compared to the first finger of the second impression). Therefore, the total number of genuine recognition attempts is (100(8*7)/2) = 2800, if FTE rate is zero.
- TRR - The first template of each finger is compared to the first template of the remaining impression, avoiding the symmetric comparison. The total number of imposter recognition attempts is ((100*99)/2) = 4950, if FTE rate is zero.

In addition, 7750 comparisons were used to test the matching accuracy of the proposed algorithm. The proposed algorithm obtained the TAR and TRR of 94.5%, and FTE of 5%. The algorithm uses a threshold of 6 to pair the minutiae and a threshold of 0.179 to match the finger codes. The following equations were used to calculate FMR and FNMR

$$\mathbf{FMR} = \frac{\boldsymbol{Number\ of\ imposters\ accepted}}{\boldsymbol{Total\ Imposters}} * \mathbf{100} = \frac{\mathbf{273}}{\mathbf{4950}} = \mathbf{5.5} \qquad (0.1)$$

$$\mathbf{FNMR} = \frac{\boldsymbol{Number\ of\ genuine\ rejected}}{\boldsymbol{Total\ genuines}} * \mathbf{100} = \frac{\mathbf{125}}{\mathbf{2260}} = \mathbf{5.5} \qquad (0.2)$$

Figure 6.10 illustrates the error rates of the proposed algorithm.

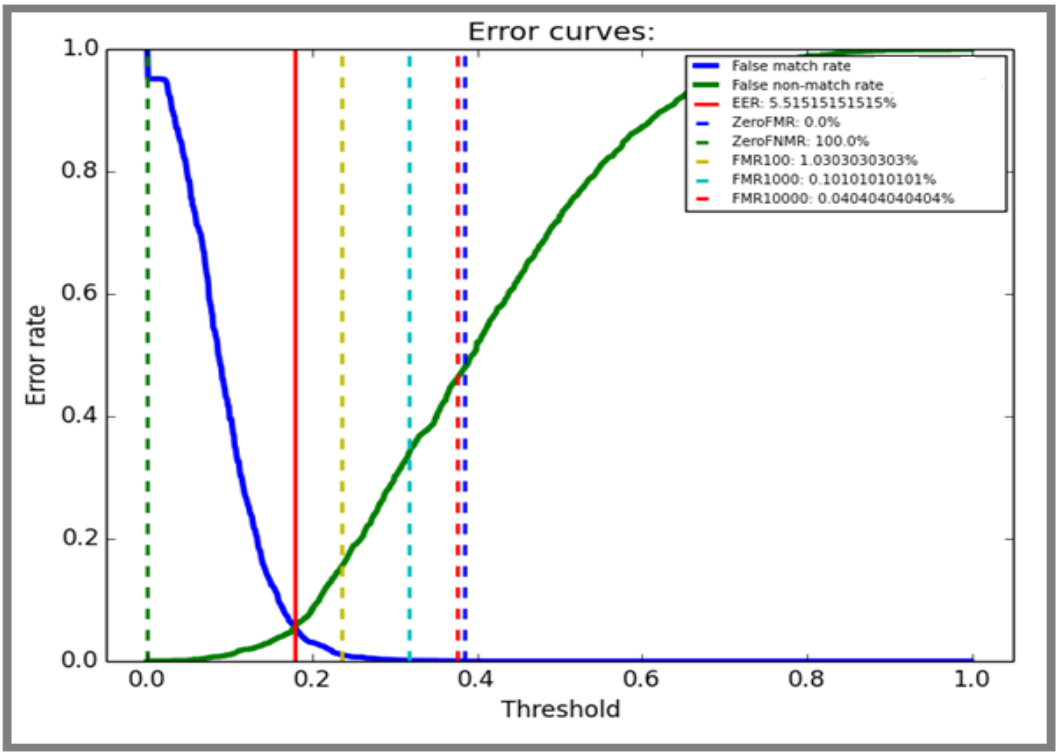


Figure 0.10 : Error rate curves for the proposed algorithm

ZeroFMR, FMR100, FMR1000, FMR10000 describes the expected value of FNMR when FMR is 0.0, 01, 0.1, 0.01 respectively. Zero FNMR describes the value of imposters in the system when all the genuine attempts are not rejected.

Table 6.1 below illustrates the comparison of the proposed algorithm with the work of [21].

Table 6.2: Comparison of the proposed algorithm on DB1-a

Criteria	Proposed algorithm	Core-based algorithm in [21]
EER (%)	5.5	6.28
Time Complexity	$f(n) = n$ or O(n)	$f(n) = n^2$ or O(n^2)

Validation is the process of assessing the proposed algorithm to check whether it meets all the requirements. Validation ensures that the model/algorithm is built correctly [65]. This research validates and verifies that the requirements of this study

have been meet. The goal of this research was to implement an accurate fingerprint MoC algorithm which can execute in a constrained smart card platform quickly (linear time complexity). Nevertheless some fingerprint algorithms compromise speed for accuracy or vice versa, especially in MoC algorithms. In contrast, the proposed algorithm focuses on both matching accuracy and matching speed. The problem statement of this research was solved due to the following reasons:

- The proposed algorithm was implemented without using a core as a reference minutia with an acceptable recognition rate.
- The algorithm provides an acceptable matching speed.
- The proposed algorithm offers the advantage of matching accuracy.

Verification ensures that the simulation or the experimental procedures were performed correctly [65]. This research verifies that the proposed algorithm can execute in a short period of time. This is because the time complexity of the proposed algorithm is linear. Time complexity is the time that it takes for the algorithm to finish.

Table 6.3 illustrates time complexity classes.

Table 6.3 : Time complexity classes

Time complexity classes		
Name	Running time (T(n))	Definition
Constant time	$O(1)$	The algorithm always takes roughly the same amount of time irrespective of the input size (n).
Linear time	$O(n)$	The time varies directly with the size of the input size.
Quadratic time	$O(n^2)$	The time varies by n^2 according to the size input.

Linear complexity algorithms are the most preferred algorithms as compared to quadratic complexity algorithms. This is because quadratic algorithms are time consuming because they take too much time to process data when compared to

linear algorithms. This implies that the proposed algorithm is faster when compared to the work of Bey as illustrated in table 6.2.

This research also verifies that the algorithm was computed without using a core reference minutia and obtained an acceptable recognition rate. This is because of the proposed algorithm obtained the TAR of 94.5% and the matcher uses extracted minutiae as reference minutiae. Equation 6.3 illustrates how the TAR was calculated.

$$\boldsymbol{TAR} = \frac{\boldsymbol{Number\ of\ genuine\ accepted}}{\boldsymbol{Total\ genuines}} * \mathbf{100} = \frac{\mathbf{2136}}{\mathbf{2260}} = \mathbf{94.5} \qquad (0.3)$$

Figure 6.11 illustrates a reference point of the proposed algorithm.

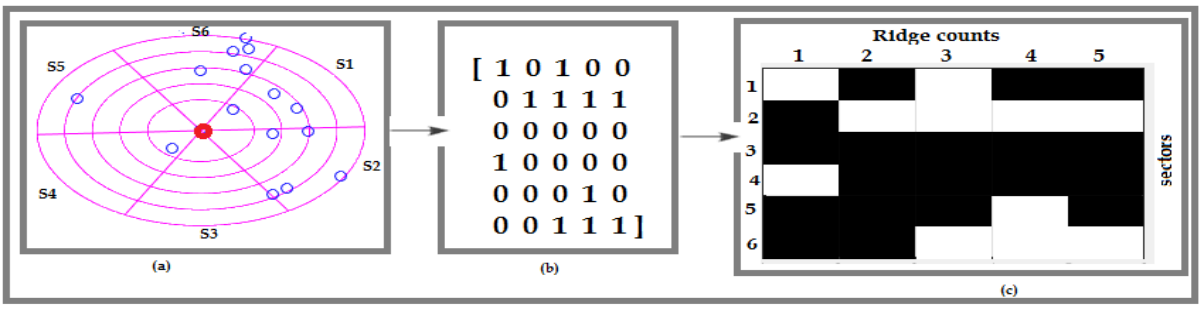


Figure 0.11 : Reference point (red) in a neighbourhood minutiae tessellation

According to the proposed algorithm, every minutia in a template which has more than five closest minutiae neighbours which are not further than 80 pixels from that minutia is a reference minutia. The algorithm outputs the expected binary codes and bit maps out of the circular minutiae tessellation.

This research also verified that the proposed algorithm offered a good matching accuracy. The algorithm performed 7210 comparisons, with 4950 imposter attempts and 2260 genuine attempts. The algorithm obtained TAR and TRR of 94.5%, with an EER of 5.5% using DB1-a as illustrated in Figure 6.10. This implies that the proposed algorithm offers a good recognition rate as compared to the work of Bey (2013).

6.4. Summary

This chapter presented the matching results, which evaluated how the proposed algorithm performed with fingerprint challenges (rotation, translation and distortion as well as missing and additional minutiae) during minutiae pairing. It also explained

how these challenges affect the proposed algorithm. The experimental results illustrated that the matching errors which are found in the algorithm are caused by partial prints, distortion and minutiae extractor errors. However the algorithm caters for these challenges but only up to a certain degree. This chapter also presented the performance evaluation of the proposed MoC verification algorithm in terms of accuracy and compared the proposed approach with the work of Bey (2013). The following chapter is going to give a conclusion and the possible future work.

Chapter 7

7. Conclusion

This chapter presents the conclusion drawn from the previous chapters of this research dissertation. This chapter presents a brief summary, future work and conclusion of this study.

7.1. Summary

The goal of this research was to develop and implement a fingerprint MoC algorithm which can accurately compare the fingerprints very rapidly (less than a second). Fingerprint MoC algorithms provide a superior secure authentication system when compared to PIN codes, ToC algorithms, and Work Sharing on-Card algorithms due to the following reasons:

- PIN codes can easily be guessed, detected or stolen via fraudulent means;
- ToC algorithms introduce security vulnerabilities due to the reference template that is sent to the terminal via communication channels during the verification process. This approach allows the imposter to steal the reference template along the communication channel;
- Although work sharing on-card techniques relieve the computational load in the smart card by performing computationally intensive processes such as template alignment at the terminal's site, this approach also introduces security vulnerabilities due to the template information that is sent out of the smart card. This is because even if the communication channel between the smart card and the terminal is encrypted, it is not guaranteed that the template cannot be stolen;
- In MoC technology, the minutiae extraction and the data acquisition is done at the terminal's site and the template comparison is done inside the secure environment of a smart card, the template never leaves the secure environment of a smart card. MoC technology makes it very difficult for the imposter to read the user's fingerprint template.

In this study the smart card was considered as a secure environment because according to ISO/IEC 24787 (2010) on-card biometric comparison standard, there is no software that can be used to download the stored template(s) inside the smart card. This means that even if the user loses the smart card, it will be very difficult for the imposter to extract the reference template stored on the smart card as all the data in the environment of a smart card is encrypted [32].

MoC technology provides the highest degree of security and privacy protection to the cardholders. However, it is challenging to implement an accurate and fast matching MoC algorithm inside the restricted environment of a smart cart. This is because smart cards offer limited memory and processing speed. Implementing a fingerprint MoC algorithm inside a smart card requires a light weight matching algorithm. This research reviewed that the matching accuracy and the speed of PC-based fingerprint algorithms outperforms MoC algorithms. This is because PCs are not resource restricted (have less memory and processing speed) like smart cards.

The methodology of this research was derived from the work of Bey (2013). The difference between the work of Bey (2013) and the proposed algorithm is the reference minutia in a template. The work of Bey (2013) uses a core as a reference minutia, instead of using the core reference point which is not always accurately computed, the proposed algorithm uses every minutia which has more than 5 closest minutiae surrounding the minutiae with an increasing distance of 80 pixels as reference point. The proposed method uses minutiae based-matching to find the similarity between the reference template and the query template. Minutiae-based matching technique requires two steps namely: minutiae extraction and template comparison. Although the accuracy of a minutiae extractor affects the accuracy of a matcher, minutiae extraction is a fairly complex process which is not investigated or analysed in this study. The proposed algorithm used the previously extracted template from FVC2002 DB1-a to compare the templates. The templates were extracted using Minutiae Cylinder Code (MCC). This method uses more than one reference minutia to cater for distortion. The proposed matching algorithm is based on neighbourhood minutiae. Neighbourhood minutiae represent the nearest minutiae from a reference point, provided that the minutiae are inside the circular tessellation. The neighbourhood minutiae structure is characterized by attributes that are rotation and translation invariant.

Each reference minutia is circularly tessellated in every neighbourhood minutiae. This method uses the tessellated neighbourhood minutiae to construct a finger code. The number of rows in each finger code depends on the number of reference minutiae present in a template. The proposed approach offers the advantages of speed due to the binary codes that it utilizes to construct the finger codes.

7.2. Significance of the research

This research fills a gap in data security systems by developing and implementing an accurate and fast MoC algorithm. The proposed algorithm offers the following advantages:

- It avoids the computationally intensive procedure of template alignment by using neighbourhood minutiae structure.
- It does not depend on a core reference minutia to initiate the matching process. A core reference point is not always accurately detected.
- This method is rotation and translation invariant.
- It uses binary representation for finger codes which results in speeded up matching and a small template size.
- This method offers a good recognition rate, speed and simplicity (does not use complex mathematics for calculations).

7.3. Specific Contributions

This work offers an efficient MoC algorithm which can be used for identity verification. The proposed algorithm can also be used as secure application software in different devices such as mobiles, Automated Teller Machine (ATM), laptop, PCs and smart cards for different applications such as banking, National ID, and door access control. The proposed approach can also be used as a stimulus for further research involving finger code MoC algorithm matching accuracy and speed. This study addresses the current MoC shortfalls which can further be used to implement various MoC algorithms in the future. It also opens up the following research questions regarding finger codes matching algorithms:

- How to construct a finger code that cannot be revised to obtain the information of the minutiae?

- Which circular tessellation offers a higher matching accuracy regarding the circular tessellation size, number of sectors and number of circular bands?
- How to order the finger code rows in order to speed-up the matching process?
- How to construct the circular tessellation in order to obtain the finger code which has less effect of distortion?
- How to identify finger code rows with partial neighbourhood minutiae (circular tessellation which contains the information about the entire tessellation due to partial prints)

7.4. Feasibility of implementation

Implementing a fast and accurate fingerprint matching algorithm inside a smart card requires careful monitoring of processing and memory usage. A matching algorithm has to be sufficiently light-weight to be implemented in a smart card. The proposed fingerprint MoC algorithm was implemented on a PC using MATLAB. The current smart cards contain 8, 16, or 32 bit processors with the memory size of 2-16 Kbytes of RAM, 64-300 Kbytes of ROM and 32-150 Kbytes of EEPROM. The 32 bit smart cards are classified as high end smart card mainly based on their price and 8 bit smart cards are classified as low cost smart cards. This research calculates the memory used to allocate the variables in the proposed algorithm to measure the feasibility of implementation of the proposed algorithm inside the smart card. The reference template is stored in the EEPROM memory and the query template is stored in the RAM memory. Using a template with 20 reference minutiae, the MATLAB code requires 4.8 Kbyte to store the reference template. The code requires 4.8 Kbytes because it treats each bit as a character. The template with 20 reference minutiae in a smart card can require a memory of approximately 0.6 Kbytes. All the static variables in the smart card are stored in the EEPROM and the dynamic variables in the smart card are stored in the RAM. The memory allocation of the variables in the RAM is approximately 15 Kbyte using MATLAB code on a PC. The memory allocation of the variables in the RAM is approximately 2.5 Kbyte using a smart card. The memory allocation of the variables in the EEPROM is approximately 20 Kbyte using a MATLAB code on a PC. The memory allocation of the variables in the EEPROM is approximately 1.875 Kbyte using a smart card. This implies that it is

feasible to implement the proposed algorithm inside the smart card. It is feasible to implement the proposed algorithm in the smart card due to the following reasons:

- The implementation of the proposed algorithm is simple, it uses four mathematical operations (*, /, +, - and XOR operator). It does not include complex mathematics such as trigonometry.
- The time complexity of the proposed algorithm is linear.
- The memory used in the proposed algorithm is less than the memory available on the smart card.

The proposed algorithm used linear time complexity to match the finger codes. It uses the following steps to match the finger codes:

- Load the reference template and the query template. Loading the whole files from start to the end uses the time complexity of O(n) where n is the file size.
- If a number of rows in the query template are less than 50, fill the query template with rows of zeros until the query template has 50 rows. Fifty is the maximum number of rows that a template can contain. This is done to make the size of the all the query template the same and also to avoid using a nested loop to count the number of reference template and the number of the query template during the matching process. This is step uses make use of constant time complexity.
- Create a loop which is going to take each row in a query template and make it a variablev_q. v_q=$\{v_1, v_2, v_3, \ldots . v_l\}$,$v_q$ is the number of variables and l represent the number of a variable. This step makes use of linear time complexity.
- Create a for loop to compare the each row in the reference template with the all v_q. This process makes use of linear time complexity
- Calculate the number of matched rows. This step is down inside the loop of the previous step.
- Decline or accept the query template. This step makes use of linear time complexity.

7.5. Conclusion

This work concludes that the accuracy of the proposed algorithm is affected by the accuracy of the minutiae extractor, distortion and partial prints. In contrast the algorithm performs well when it encounters challenges such as rotation and translation. This research develops a fingerprint MoC algorithm without a computation of a core. The proposed algorithm uses circular neighbourhood minutiae tessellation to construct the finger codes. The proposed algorithm uses multiple reference minutiae to cater for distortion present in templates during fingerprint matching. The experimental results showed that the proposed algorithm offers the advantages of speed and good recognition rate. The experimental results also showed the proposed algorithm is comparable with existing MoC algorithms and that it can fit into a smart card. The proposed algorithm obtained a better matching accuracy and speed when compared to the work of Bey (2013).

7.6. Future work

As future work, properties of fingerprint image distortion and partial/incomplete fingerprints will be studied. In order to use a different minutiae extractor that is more accurate than the one which is used for the proposed algorithm to improve the accuracy of the matcher, further study on different minutiae extractors will be done. The proposed algorithm will also be implemented inside the platform of the smart card.

Bibliography

[1] M. Yildiz and M. Gokturk, "Combining Biometric ID Cards and Online Credit Card Transactions," in *Digital Society, 2010. ICDS'10. Fourth International Conference on*, 2010, pp. 20-24.

[2] N. S. Udoh, O. T. Eluwole, and A. O. Ologunde, "Ethical responsibilities: The smart card engineer," in *Ethics in Science, Technology and Engineering, 2014 IEEE International Symposium on*, 2014, pp. 1-5.

[3] K. Markantonakis, *Smart cards, tokens, security and applications*: Springer, 2007.

[4] O. Henniger and K. Franke, "Biometric user authentication on smart cards by means of handwritten signatures," in *Biometric Authentication*, ed: Springer, 2004, pp. 547-554.

[5] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar, *Handbook of fingerprint recognition*: Springer Science & Business Media, 2009.

[6] A. K. Jain and A. Kumar, "Biometrics of next generation: An overview," *Second Generation Biometrics,* 2010.

[7] R. A. Rahim, N. Ali, M. I. Idris, D. F. Yap, and M. M. Ismail, "Fingerprint Enhancement for Minutiae Matching."

[8] C. Fanglin, Z. Jie, and Y. Chunyu, "Reconstructing Orientation Field From Fingerprint Minutiae to Improve Minutiae-Matching Accuracy," *Image Processing, IEEE Transactions on,* vol. 18, pp. 1665-1670, 2009.

[9] J. Feng, Z. Ouyang, and A. Cai, "Fingerprint matching using ridges," *Pattern Recognition,* vol. 39, pp. 2131-2140, 2006.

[10] S. Pankanti, S. Prabhakar, and A. K. Jain, "On the individuality of fingerprints," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 24, pp. 1010-1025, 2002.

[11] A. Ross, J. Shah, and A. K. Jain, "From Template to Image: Reconstructing Fingerprints from Minutiae Points," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 29, pp. 544-560, 2007.

[12] P. Arora, "Fingerprint Recognition Using Minutiae Score Matching."

[13] Ł. Więcław, "A minutiae-based matching algorithms in fingerprint recognition systems," *Journal of Medical Informatics & Technologies,* vol. 13, pp. 65-71, 2009.

[14] J. Zhou, F. Chen, and J. Gu, "A novel algorithm for detecting singular points from fingerprint images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 31, pp. 1239-1250, 2009.

[15] W. Zhang and Y. Wang, "Core-based structure matching algorithm of fingerprint verification," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, 2002, pp. 70-74.

[16] Q. Zhang and H. Yan, "Fingerprint classification based on extraction and analysis of singularities and pseudo ridges," *Pattern Recognition,* vol. 37, pp. 2233-2243, 2004.

[17] B. Vibert, C. Rosenberger, and A. Ninassi, "Security and performance evaluation platform of biometric match on card," in *Computer and Information Technology (WCCIT), 2013 World Congress on*, 2013, pp. 1-6.

[18] C. T. Pang, Y. W. Yun, and X. Jiang, "On-Card Matching," in *Encyclopedia of Biometrics*, ed: Springer, 2009, pp. 1014-1021.

[19] C. T. pang, "On-card matching," vol. 14, p. 8, 2009.

[20] Y. H. Yahaya, M. R. M. Isa, and M. I. Aziz, "Fingerprint Biometrics Authentication on Smart Card," in *Computer and Electrical Engineering, 2009. ICCEE'09. Second International Conference on*, 2009, pp. 671-673.

[21] F. Benhammadi and K. B. Bey, "EMBEDDED FINGERPRINT MATCHING ON SMART CARD," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 27, 2013.

[22] P. Grother, W. Salamon, and R. Chandramouli, "Biometric Specifications for Personal Identity Verification," *NIST Special Publication,* vol. 800, pp. 76-2, 2013.

[23] J. Bringer, H. Chabanne, T. A. Kevenaar, and B. Kindarji, "Extending match-on-card to local biometric identification," in *Biometric ID Management and Multimodal Communication*, ed: Springer, 2009, pp. 178-186.

[24] T. M. a. J. v. d. M. M.B. Shabalala, "Fingerprint Match-On-Card: Review and Outlook," 24 – 25 March 2015.

[25] A. S. Patrick, "Fingerprint Concerns: Performance, Usability, and Acceptance of Fingerprint Biometric Systems," *National Research Council of Canada.[Online]. Available http://www. andrewpatrick. ca/essays/fingerprint-concerns-performanceusability-and-acceptance-of-fingerprint-biometric-systems (Accessed July 24, 2014),* 2008.

[26] E. Zhu, J. Yin, and G. Zhang, "Fingerprint matching based on global alignment of multiple reference minutiae," *Pattern Recognition,* vol. 38, pp. 1685-1694, 2005.

[27] K. M. Shelfer and J. D. Procaccino, "Smart card evolution," *Communications of the ACM,* vol. 45, pp. 83-88, 2002.

[28] A. Boorghany and R. Jalili, "Implementation and Comparison of Lattice-based Identification Protocols on Smart Cards and Microcontrollers," *IACR Cryptology ePrint Archive,* vol. 2014, p. 78, 2014.

[29] M. W. Webster, "Utilizing destructive features as RAM code for a storage device," ed: Google Patents, 2016.

[30] K. Mayes and K. Markantonakis, *Smart cards, tokens, security and applications*: Springer Science & Business Media, 2007.

[31] U. Hansmann, M. S. Nicklous, T. Schäck, A. Schneider, and F. Seliger, *Smart card application development using Java*: Springer Science & Business Media, 2012.

[32] T.-P. Chen, W.-Y. Yau, and X. Jiang, "Fast match-on-card technique using in-matcher clustering with ISO minutia template," *International Journal of Biometrics,* vol. 7, pp. 119-146, 2015.

[33] T.-p. C. a. W.-y. Yau, "ISO/IEC standards for On-card biometric comparison," *Int .J. Biometrics,* vol. 5, pp. 30-52, 2013.

[34] J. C. Yang and D. S. Park, "A fingerprint verification algorithm using tessellated invariant moment features," *Neurocomputing,* vol. 71, pp. 1939-1946, 2008.

[35] A. S. Rikin, L. Dongju, T. Isshiki, and H. Kunieda, "A fingerprint matching using minutia ridge shape for low cost match-on-card systems," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences,* vol. 88, pp. 1305-1312, 2005.

[36] D. Brown and K. Bradshaw, "Improved Fingercode alignment for accurate and compact fingerprint recognition," in *Technologies for Homeland Security (HST), 2016 IEEE Symposium on*, 2016, pp. 1-6.

[37] D. A. Kumar and T. U. S. Begum, "A Comparative Study on Fingerprint Matching Algorithms for EVM," *Journal of Computer Sciences and Applications,* vol. 1, pp. 55-60, 2013.

[38] M. Shabalala, T. Moabalobelo, and J. van der Merwe, "Fingerprint Match-on-Card: Review and Outlook," in *Iccws 2015-The Proceedings of the 10th International Conference on Cyber Warfare and Security*, 2015, p. 286.

[39] T.-P. Chen, W.-Y. Yau, and X. Jiang, "ISO/IEC standards for on-card biometric comparison," *International Journal of Biometrics,* vol. 5, pp. 30-52, 2013.

[40] J. F. A. K. Jain, and K. Nandakumar, "Fingerprint matching," *Computer (Long. Beach. Calif),* vol. 43, pp. 36-44, 2010.

[41] A. Jain, A. Ross, and S. Prabhakar, "Fingerprint matching using minutiae and texture features," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, 2001, pp. 282-285.

[42] J. Feng, "Combining minutiae descriptors for fingerprint matching," *Pattern Recognition,* vol. 41, pp. 342-352, 2008.

[43] R. Sanchez-Reillo and C. Sanchez-Avila, "Fingerprint verification using smart cards for access control systems," in *Security Technology, 2001 IEEE 35th International Carnahan Conference on*, 2001, pp. 250-253.

[44] R. Sanchez-Reillo, L. Mengibar-Pozo, and C. Sanchez-Avila, "Microprocessor smart cards with fingerprint user authentication," *Aerospace and Electronic Systems Magazine, IEEE,* vol. 18, pp. 22-24, 2003.

[45] S. B. Pan, D. Moon, Y. Gil, D. Ahn, and Y. Chung, "An ultra-low memory fingerprint matching algorithm and its implementation on a 32-bit smart card," *Consumer Electronics, IEEE Transactions on,* vol. 49, pp. 453-459, 2003.

[46] C. Liu, T. Xia, and H. Li, "A hierarchical hough transform for fingerprint matching," in *Biometric Authentication*, ed: Springer, 2004, pp. 373-379.

[47] T. Cucinotta, M. Di Natale, and R. Brigo, "A fingerprint matching algorithm for programmable smart cards," in *Information Security Bulletin Journal*, 2005.

[48] Y. Moon, H. Ho, K. Ng, S. Wan, and S. Wong, "Collaborative fingerprint authentication by smart card and a trusted host," in *Electrical and Computer Engineering, 2000 Canadian Conference on*, 2000, pp. 108-112.

[49] H. W. Yeung, Y. S. Moon, J. Chen, F. Chan, Y. M. Ng, H. S. Chung*, et al.*, "A comprehensive and real-time fingerprint verification system for embedded devices," in *Defense and Security*, 2005, pp. 438-446.

[50] H. Lam, W. Yau, T. Chen, Z. Hou, and H. Wang, "Fingerprint pre-alignment for hybrid match-on-card system," in *Information, Communications & Signal Processing, 2007 6th International Conference on*, 2007, pp. 1-4.

[51] V. T. De Zhi and S. A. Suandi, "Fingercode for identity verification using fingerprint and smart card," in *Control Conference (ASCC), 2015 10th Asian*, 2015, pp. 1-6.

[52] S. Bistarelli, F. Santini, and A. Vaccarelli, "An asymmetric fingerprint matching algorithm for Java Card TM," *Pattern analysis and applications,* vol. 9, pp. 359-376, 2006.

[53] M. Govan and T. Buggy, "A computationally efficient fingerprint matching algorithm for implementation on smartcards," in *Biometrics: Theory, Applications, and Systems, 2007. BTAS 2007. First IEEE International Conference on*, 2007, pp. 1-6.

[54] R. Cappelli, M. Ferrara, and D. Maltoni, "Minutia Cylinder-Code: A New Representation and Matching Technique for Fingerprint Recognition," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 32, pp. 2128-2141, 2010.

[55] J. W. Creswell, *Research design: Qualitative, quantitative, and mixed methods approaches*: Sage publications, 2013.

[56] D. E. Gray, *Doing research in the real world*: Sage, 2013.

[57] B. R. Hunt, R. L. Lipsman, and J. M. Rosenberg, *A guide to MATLAB: for beginners and experienced users*: Cambridge University Press, 2014.

[58] J. Kiusalaas, *Numerical methods in engineering with MATLAB®*: Cambridge University Press, 2010.

[59] R. Grepl, "Real-Time Control Prototyping in MATLAB/Simulink: Review of tools for research and education in mechatronics," in *Mechatronics (ICM), 2011 IEEE International Conference on*, 2011, pp. 881-886.

[60] D. Madrigal and B. McClain, "Strengths and weaknesses of quantitative and qualitative research," *UX Matters,* 2012.

[61] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain, "FVC2002: Second fingerprint verification competition," in *Pattern recognition, 2002. Proceedings. 16th international conference on*, 2002, pp. 811-814.

[62] M. C.-C. SDK, "Biometric System Laboratory, DISI–University of Bologna. 2014," ed.

[63] E. Marasco and A. Ross, "A survey on antispoofing schemes for fingerprint recognition systems," *ACM Computing Surveys (CSUR),* vol. 47, p. 28, 2015.

[64] Z. Jin, A. B. J. Teoh, T. S. Ong, and C. Tee, "Fingerprint template protection with minutiae-based bit-string for security and privacy preserving," *Expert systems with applications,* vol. 39, pp. 6157-6167, 2012.

[65] A. Tolk, "Verification and Validation‖," *Engineering Principles of Combat Modeling and Distributed Simulation, John Wiley & Sons, Hoboken NJ,* pp. 263-294, 2012.