Honours Project
(CSI 4900)

Cancelable Biometrics: Analysis and Implementation of a Fingerprint Template Transformation Method

By

Arman Kompany Zare

Project Supervisor: Dr. Carlisle Adams

Faculty of Engineering

School of Electrical Engineering and Computer Science (EECS)

University of Ottawa

Fall 2023

Abstract

In this report, we analyze, implement, and verify a specific cancelable biometric method from a 2018 research paper (Yang, Hu, Wang, & Wu, 2018), which is used for transforming fingerprint templates. Our implementation of this method has been solely developed on Python over the course of three months with the help of some open source python libraries. The implementation closely follows the given algorithms on the paper and bears certain advantages in comparison such as improved runtime performance using GPU utilization (powered by NVIDIA CUDA), and all tools involved being fully open source and free to use. The verification of the paper's results was done by running the implementation on the same FVC fingerprint datasets used on the paper and other datasets, using the same parameter configurations as indicated in their different test cases. Similar outcomes to those on the paper were achieved as a result, which further proved the security and accuracy of this method for fingerprint matching.

Keywords:
Fingerprint, Cancelable Biometrics

Implementation Software and Tools:
Python 3, Anaconda, Numba, TensorFlow, Nvidia CUDA

Acknowledgement

First and foremost, I would like to thank Professor Carlisle Adams for suggesting this particular research subject, and his continued support and supervision on it. I also appreciate Professor Wencheng Yang, and the other authors of the 2018 paper, for responding to my inquiries regarding their work with helpful explanations and answers. Full credits and acknowledgement goes to the developers and maintainers of the Python libraries used within this project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's world, biometrics play a greater role than ever in preserving our digital security and privacy. Some of the reasons that most people and organizations choose biometrics over static or variable string-based passwords are: Uniqueness, Convenience, and Complexity. However, there is one specific weakness involved with biometric methods, which will be discussed further ahead (under Section 1.2). There exist numerous methods for resolving this weakness which generally fall under the term: Cancelable Biometrics. In this report a specific method (Yang et al., 2018) which is used for fingerprint biometric data, will be discussed and analyzed, along with a documentation on its implementation, yielded results and performance comparison.

1.1 Advantages of Biometric Data

The Uniqueness advantage comes from the fact that when it comes to biometrics, especially fingerprints, almost no two individuals possess the same biometric information. So there are no overlaps in case of comparisons. On the other hand, there is a slim chance that two people in the world would coincidentally share the same static password for their applications or accounts, which would lead to security issues if one of their passwords gets compromised.

Regarding Convenience, it is obvious that biometric information do not necessarily need to be remembered by a person, and are perfectly portable since they are literal parts of that person's own body. On the other hand, complex passwords may be difficult to remember and

misinputs could occur while entering them in. Even sometimes, said complex passwords or hashes are so large in data size that they need to be carried around by flash or external drives.

Naturally, the data complexity of a single fingerprint or piece of biometric data surpasses the complexity of an average static password by a great margin. As it will be discussed further later in this report, approximately 2.5 Megabytes worth of data is required for representing a single encoded fingerprint template. Meanwhile an average static password (32 characters) is represented by a mere 32 Bytes of data. At first glance, there may be a space cost disadvantage, but in return the exponentially increased complexity will make it near impossible to reconstruct biometric data through exhaustive search.

1.2 The Problem: Irrevocability

In spite of the advantages mentioned above, static passwords hold at least one important advantage over biometric data, and that is their revocability. In case of a credential leak, the exposed user can easily revoke their password and have it replaced by a new one, and as a result, the leaked password will hold no leverage in future compromises on the same user.

On the other hand, if biometric data, such as a fingerprint get compromised, it will be forever exposed and there will be no way to have it revoked and replaced like a static password. Additionally, the risk for the biometric data to be used in future attacks will remain indefinitely.

1.3 The Solution: Cancelable Biometrics

Fortunately for us, there are a plethora of methods which cover the irrevocability weakness. These solutions fall under the category of Cancelable Biometrics. In general, the method converts a fingerprint into a certain data structure referred to as a template, which represents the fingerprint's minutiae data with great accuracy. Then the method transforms the template using a public key into a transformed template. Afterwards, the transformed template and its respective public key are saved to a database. In case the database gets hacked into, the attacker wouldn't be able to retrieve the original template through the public key and the transformed

template. In addition, the transformed template and its respective public key are revocable. Meaning that the exposed data could be easily removed from the database, and be replaced by a new public key from which another completely different transformed template could be generated.

1.3.1 Solution Outline

The process behind Cancelable Biometrics methods usually involves two generic stages: Enrolment and Verification. The following is a rough explanation of the whole process and its stages.

Enrolment

In the Enrolment stage, the Target fingerprint which is supposed to be used as reference for future comparisons gets scanned, and we get a grayscale picture of the fingerprint scan as a result. Next, we need to detect all the valid minutiae points on the fingerprint image, and create an Initial Fingerprint Template (IFT) using said minutiae. The IFT is composed of extracted minutiae data under a number of different schemes. From this point on, is where the most crucial process begins to occur. Using a set of algorithms designed for this specific Cancelable Biometric method, we will transform the IFT into an Transformed Fingerprint Template (TFT) using a psudeo-randomly generated Public Key. Finally, the TFT along with its respective Public Key which was used to generate it, will be stored in memory.

Verification

Afterwards, in the Verification stage, our Query fingerprint will be scanned and converted into another IFT using the same schemes and standards based on its minutiae data. Then using the same public key, the IFT will be transformed into a TFT. The Query fingerprint's TFT will then be compared with the Target fingerprint's TFT using a series of comparison methods. Ultimately, using a final score derived from the comparison methods and a predetermined score threshold, we will determine whether the two fingerprints match or not.

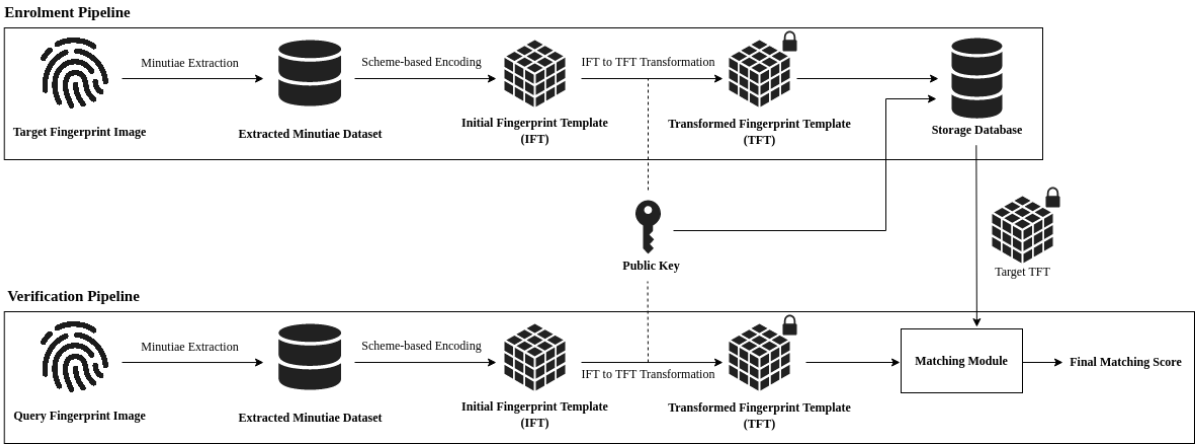

Figure 1.1: A simplified diagram of the Enrolment and Verification pipelines

1.3.2 Solution Properties

Ofcourse, in order for both security and accuracy to be guaranteed, two main requisites must be met in the design a Cancelable Biometric method. One is the irreversibility of the process, in which the IFT gets transformed into the TFT. The other is the comparability of the Target fingerprint's TFT with the Query fingerprint's TFT, hence the differences in the Target and Query IFT must be somewhat preserved in their respective TFTs.

Irreversibility

In the case of an intrusion, the worst case scenario would be the compromise of the TFT and Public Key pair. In this case, the infiltrator should not be able to obtain the IFT from the TFT and Public Key by any means other than brute force. For a function to guarantee this attribute, it must preferably be a non-injective function (not one to one), and non-invertible. The input element of this function must be also large enough for exhaustive search to be near impossible. For instance a hash function such as SHA-512, is a good example of an irrevertible function.

Comparability

An irrevertible function such as a hash function may seem to be a good candidate for our purpose at first glance. But when it comes to biometric input, the smallest differences in input data could result in very different and incomparable outputs after being put through something such as a hash function.

Therefore we need a function or transformation that could preserve the small initial differences and render the results comparable. In Chapter 2, a few of these functions, which are relevant to the main algorithm of our study, will be discussed.

Chapter 2

Preliminaries

Before we get to the full analysis of the Yang method (Yang et al., 2018), a few basic definitions and concepts will be briefly covered. After a brief explanation of the Yang method, some of its exclusive advantages over other methods in Cancelable Biometrics will be listed. Moreover at the end, the irreversibility and comparability criterias of the IFT to TFT transformations will be demonstrated.

2.1 Concepts and Definitions

2.1.1 Minutiae Points

The exterior of a human fingertip is a pattern of interleaved ridges and valleys. At its local level, important features called minutiae, can be found in fingerprint patterns. In this context, minutiae refers to the various ways that said ridges can be discontinuous. (Maltoni, Maio, Jain, & Feng, 2022)

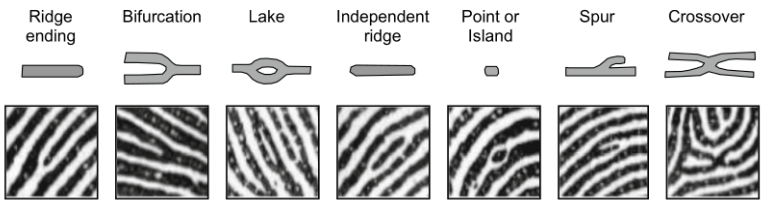

Figure 2.1: Seven most common minutiae types

The points in which these discontinuities or changes in ridge pattern occur, are referred to as minutiae points. Minutiae points possess several inherent attributes, but we will only need a few of them for our purposes. Each minutiae point has the following primary attributes include: X-Y cartesian coordinates, minutiae orientation, minutiae type, and a confidence score.

	x	y	angle	score	class
0	126.0	161.0	2.422032	0.838129	2.0
1	198.0	322.0	2.099404	0.799058	4.0
2	148.0	204.0	1.143339	0.803224	0.0
3	178.0	261.0	1.534922	0.807595	2.0
4	147.0	180.0	1.919628	0.811289	2.0
5	261.0	316.0	1.653534	0.790637	4.0

Figure 2.2: A list of detected minutiae points and their attributes

The cartesian coordinates are the pixel coordinates of the minutiae point's location on the fingerprint input image. The orientation of a minutiae, is the direction it is heading towards, or in other words the direction of the line tangent to the ridge at that point. The minutiae type is an integer enumeration of the seven most common types of minutiae as referred to in Figure 2.1. The confidence score, is the probability of that minutiae point being a legitimate minutiae point, according to the FingerFlow framework, which we will discuss in the next chapter.

2.1.2 Delaunay Triangulation

Considering a set of discrete points P on a euclidean plane, a three-point subset:

$$\{p_1, p_2, p_3\} \subset P$$

is considered a Delaunay Triangle, if and only if no other point in P is inside the circumcircle of the three-point subset. (Delaunay, 1934)

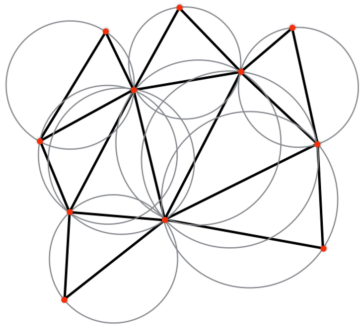

Figure 2.3: A set of points subjected to Delaunay Triangulation

After deriving all feasible Delaunay Triangles from the points in P, we end up with a connected graph of neighboring triangles with no intersecting edges. There are many algorithms with which to perform Delaunay Triangulation with:

Voronoi Algorithm

Given a set of n discrete points $P = \{p_1, p_2, ..., p_n\}$ on a Euclidean plane, the plane is segmented into n non-overlapping partitions dedicated to each point in P. Each of these partitions is referred to as a Voronoi cell. Every Voronoi cell C_k, consists of every point on the plane for which its respective point p_k is the closest site. Meaning the distance of any arbitrary point within C_k is minimal from itself to p_k than to any other point $p_i \neq p_k$. The set of all Voronoi cells, make up a Voronoi diagram.

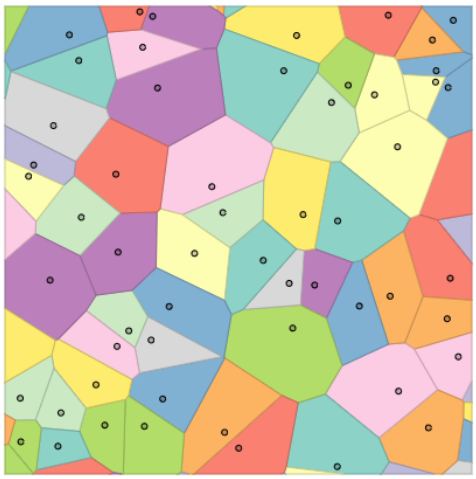

Figure 2.4: A Voronoi diagram consisting of Voronoi cells and their respective points

The borders of these Voronoi cells, refered to as Voronoi edges, are constructed by creating the perpendicular bisector of the line between two points respectively and blending them. Finally, the Delaunay Triangulation of P is then created by connecting the points of all neighboring Voronoi cells. The Yang method originally uses the Voronoi algorithm to derive the Delaunay Triangles from the set of minutiae points on a fingerprint. But there is another easier to implement algorithm which was used in our implementation instead.

The Bowyer-Watson Algorithm

In computational geometry, the Bowyer-Watson algorithm is an incremental algorithm for deriving all possible Delaunay Triangles from a set of points in N-dimensional Euclidean space. The pseudo-code is fully described in A.1. The complexity for this algorithm is $O(nlog(n))$ if implemented efficiently, and $O(n^2)$ in a normal sub-optimal implementation. The Python implementation of the Bowyer-Watson algorithm was readily available from a Python package named `delaunay-triangulation` (Henkel, 2021), which was imported and used in our code.

Figure 2.5: Minutiae points on a fingerprint and its Delaunay Triangulation

2.2 The Yang Method

2.2.1 Brief Outline

The Yang method's outline follows from general outline described in 1.3.1, and includes the mentioned Enrolment and Verification phases. The generation of the IFT from the fingerprint image, is done separately in parallel under two different schemes: The Polar Coordinate-based

scheme and the Delaunay Triangulation-based scheme.

The Polar IFT, which is an array of vectors with zero and one entries, is then transformed using the Feature Decorrelation Algorithm (FDA) (which will be discussed in Chapter 3), and then projected into a lower dimension using a projection matrix pseudo-randomly generated from a seed. The seed used to generate the pseudo-random projection matrix, is a part of the Public Key associated with the TFT. Every vector in the Polar IFT, is essentially based around a reference minutiae point and its polar coordinate-based relations with other minutiae points within a certain predefined radius. Therefore within the Polar IFT, there's a vector dedicated to every minutiae point, since every vector is created by setting a minutiae point as the origin point.

The Delaunay IFT, is essentially an array of all Delaunay Triangles encoded and quantized into binary numbers, based on attributes such as side length, the angle between two sides of the triangle, the minutiae type of the triangle vertices, and orientation differences between two vertices. Thereafter, the Delaunay IFT is permutated into the Delaunay TFT by converting each encoded triangle in the IFT from binary to integer, and adding it to another integer which is generated based on a predetermined integer ϕ. The ϕ number is another part of the Public Key associated with the TFT.

After the two IFTs of the target fingerprint are obtained, we obtain the two IFTs of the query fingerprint using the same Public Key (same projection matrix and ϕ). Then the similarity scores of the respective IFTs are calculated (`SC_MAX` for the Polar IFTs, and `SD` for the Delaunay IFTs), normalized, and put into a Final Score formula for giving the finalized similarity score of the two fingerprints.

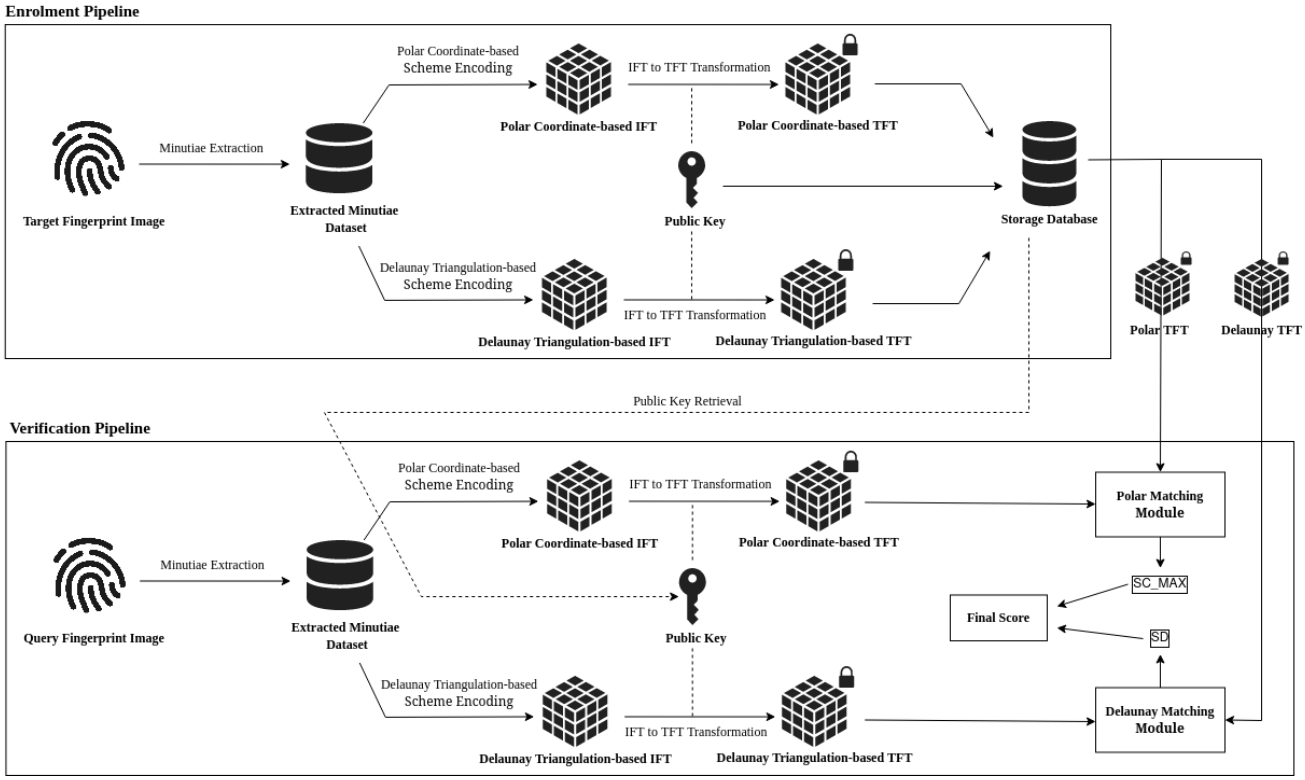

Figure 2.6: Full Enrolment and Verification pipeline diagram of the Yang method

2.2.2 Advantages

The main contributions of the Yang method compared to the previously mentioned works, involve the following advantages:

Reduced Non-Linear Distortion

Since the Yang method's IFT and final result depend on a combination of the Polar Coordinate-based and Delaunay Triangulation-based schemes, the impact of non-linear distortion, which would lead to recognition inaccuracy being reduced. Roughly speaking, non-linear distortion results from systems in which the output signal is not exactly proportional to the input signal.

For instance a cause of non-linear distortion would be to have a system which solely relies on the Polar Coordinate-based scheme for the generation of its IFT, which would result in the system not taking into account the minutiae that are located far away from the reference minutiae. That is because, in the Polar Coordinate-based scheme, only minutiae that are

placed within a certain limited radius of the reference minutiae are taken into account and used to generate the IFT. Therefore, compared with the system that solely uses a Polar Coordinate-based scheme, higher recognition accuracy is achieved.

ARM Attack Invulnerability

The ARM attack, otherwise known as Attack via Record Multiplicity, is an exploit which results from the compromise of several TFTs and their respective generative parameters (the Public Key in our case), and leads to the full or partial generation of the original IFT. A main reason that a lot of Cancelable Biometrics methods suffer from the ARM attack is due to feature correlation. To propagate this issue, a FDA is implemented within the method, so that in the process of the IFT to TFT transformation, the feature vectors within the IFT become uncorrelated in regards to another. Without any correlation between said features in the IFT, the attacker wouldn't have enough data to determine the original IFT.

2.2.3 Guarantee of Irreversibility

Polar Coordinate-based Scheme

The irreversibility of the IFT to TFT transformation is guaranteed through the random projection F performed on the IFT vectors.

$$F : \mathbb{C}^n \to \mathbb{C}^m \text{ where } n \geq m\text{: } F(x) = Ax$$

In our case, the projection matrix A is composed of pseudo-randomly generated entries based on a seed. According to basic linear algebra fundamentals, since the projection matrix A is not a square matrix, it is therefore not invertible. Therefore it is not possible to derive F^{-1} except through exhaustive search. Hence irreversibility is guaranteed due to the involvement of random projection within the IFT to TFT transformation within the Polar Coordinate-based scheme.

2.2.4 Guarantee of Comparability

Polar Coordinate-based Scheme

The irreversibility of the IFT to TFT transformation is guaranteed, but what about its comparability? Will the characteristics of the IFT vectors in regards to each other be preserved after performing random projection on them? Fortunately, the answer is yes, thanks to the Johnson-Lindenstrauss (JL) Lemma (Lindenstrauss & Johnson, 2018):

Lemma: For any $0 \leq \epsilon \leq 1$ and any integer p, let n be a positive integer such that:

$$n \geq \frac{4ln(p)}{\epsilon^2/2 - \epsilon^3/3}.$$

Then for any set $\mathbb{S}$ of $p = |\mathbb{S}|$ data points in $\mathbb{R}^N$, there is a map $f : \mathbb{R}^N \to \mathbb{R}^n (N \geq n)$, such that for all $x, y \in \mathbb{S}$:

$$(1 - \epsilon)\|x - y\|^2 \leq \|f(x) - f(y)\|^2 \leq (1 + \epsilon)\|x - y\|^2 \tag{2.1}$$

Overall, the JL lemma states that our IFT vectors in $\mathbb{R}^N$ vector space can be embedded into a lower dimension vector space $\mathbb{R}^n$, such that the difference between any two pair of vectors in the IFT will also be approximately maintained in between their respective pair in the TFT after the random projection is performed on the vectors.

Chapter 3

Algorithm and Implementation

This chapter will include the full description of the Yang method and the generations of its IFTs and TFTs. In addition, we will go over the list of tools and frameworks utilized for our implmentation and the reason behind the selection of said tools. Finally, the whole development process and the challenges involved in it will be provided in a detailed manner.

3.1 The Algorithm in Full Detail

First we begin by obtaining all minutiae and their attributes from the fingerprint image. As a result we will obtain a set of N minutiae $M = \{m_0, m_1, ..., m_{N-1}\}$ within that image. Every minutiae $m_i \in M$ can be represented by its respective vector $m_i = (x_i, y_i, \theta_i, t_i)$, where x_i and y_i respectively represent the x and y coordinates of the minutiae m_i on the fingerprint picture, θ_i represents the minutiae orientation, and finally t_i represents the minutiae type. θ_i ranges between $(0, 2\pi)$, and t_i's value is represented by single bit, signifying whether it is a ridge ending minutiae type or not. We choose to base t_i around the ridge ending minutiae type, because it's the most commonly seen type of minutiae by a great margin compared to other minutiae types.

3.1.1 Generation of the IFTs

With the minutiae data on hand, we begin generating the IFTs both under the Polar Coordinate-based scheme and the Delaunay Triangulation-based scheme.

Polar Coordinate-based IFT

Under this scheme, the final IFT ($C = \{P(m_i)\}_{i=0}^{N-1}$) is an array of vectors with 0 and 1 entries. Each IFT vector $P(m_i)$ is created based on a minutiae point m_i which is used as an origin point. Now let's assume we want to derive $P(m_0)$, we would first have to take minutiae point m_0 as reference and calculate the three attributes $(\rho_i, \alpha_i, \beta_i)$ of the other minutiae with respect to m_0 as the origin point.

First off, only minutiae points that fall within a radius R (= 300 pixels) of point m_0 are taken into account. Moreover, every other minutiae point's data relative to m_0 consists of three attributes. The radial distance ρ_i, which implies the distance from selected minutiae m_i to the reference minutiae m_0, which ranges from: $0 \leq \rho_i \leq R = 300$.

Next, we have the radial angle α_i, which is the polar angle of minutiae point m_i with respect to m_0 as the origin point. The third and last attribute, is the absolute minutiae orientation difference β_i, between between m_0 and m_i. Both α_i and β_i range from: $0 \leq \alpha_i, \beta_i \leq 2\pi$.

After the $(\rho_i, \alpha_i, \beta_i)$ triplet for every minutia $m_i \neq m_0$ is obtained, it's time to perform quantization on the three attributes. We assume that the step sizes for ρ_i, α_i, and β_i are s_ρ, s_α, and s_β, where $5 \leq s_\rho \leq 20$ and $\pi/12 \leq s_\alpha, s_\beta \leq 2\pi/9$. Then we quantize the polar space around m_0 into a 3D cube containing $l_C = L \times S \times H$ cells, where $L = \lfloor R/s_\rho \rfloor$, $S = \lfloor 2\pi/s_\alpha \rfloor$, and $H = \lfloor 2\pi/s_\beta \rfloor$. The cell in which minutia point m_i is located in the 3D cube is $(\rho_i^q, \alpha_i^q, \beta_i^q)$, where $\rho_i^q = \lfloor \rho_i/s_\rho \rfloor$, $\alpha_i^q = \lfloor \alpha_i/s_\alpha \rfloor$, and $\beta_i^q = \lfloor \beta_i/s_\beta \rfloor$. By doing so, we eventually get the vector $P(m_0)$ of length l_C, which consists of '1' and '0' entries, which is obtained by concatenating all the rows of cube cell values together in one line. The '1' indicates that one or more quantized minutiae points belong in that corresponding cube cell.

The size of any vector $P(m_i) \in C$ depends on the value of l_C, which itself depends on the step sizes for the triplet attributes. The smaller the step sizes, the bigger l_C will get and the more accurate our method becomes. And with our defined boundaries for step sizes, the largest l_C can get, is 34560. To finish this section up, we repeat the same steps mentioned above for all other m_i, until we get all other $P(m_i)$ and obtain the full Polar Coordinate-based IFT ($C = \{P(m_i)\}_{i=0}^{N-1}$).

Delaunay Triangulation-based IFT

First we start by performing Delaunay Triangulation on all minutiae points in M, through either the Voronoi or the Bowyer-Watson algorithm. After all triangles are extracted, we need to go over every triangle $\triangle m_1 m_2 m_3$ and extract certain attributes from it for further processing. The IFT (D) under this scheme, is also similar to the Polar Coordinate-based IFT, in the fact that it is also an array of vectors with zero and one entries, with each vector dedicated to a triangle.

Suppose we're processing the $\triangle m_1 m_2 m_3$ triangle. There are four attributes $(\alpha_i, l_i, h_i, t_i)$ we need to take into account. First one is α_i, which is the size of the largest angle of the triangle, its value is ranged between $0 \leq \alpha_i \leq 2\pi$. Second and third one are l_i and h_i, which are respectively the lengths of the larger and smaller edges of the α_i angle. Both are ranged between $0 \leq l_i, h_i \leq 300$. Finally, there's t_i which is supposed to represent the minutiae types of the three vertices of the triangle using three bits. First bit is dedicated to the minutiae type of the largest angle's vertice, second and third are dedicated to the minutiae type of the vertices positioned on the larger and smaller edges of the triangle.

At this point, all attributes other than t_i (since it is already a three bit binary vector) need to be quantized. Similar to what we did in the Polar Coordinate-based IFT, the quantization step sizes are set to be s_α, s_l, and s_h, which are respectively ranged at $\pi/12 \leq s_\alpha \leq \pi/9$, and $15 \leq s_l, s_h \leq 25$. To each attribute, the minimum amount of bits to cover all possible quantization values will be dedicated. For instance, if the value of α_i^q ranges between 0 and 9, then 4 bits will be dedicated to α_i^q. The quantized value of each attribute is calculated as follows: $\alpha_i^q = \lfloor \alpha_i / s_\alpha \rfloor$, $l_i^q = \lfloor l_i / s_l \rfloor$, and $s_i^q = \lfloor h_i / s_h \rfloor$. Finally we convert each quantized attribute into binary, and concatanate every binarized attribute into a single list which we will refer to as $f^q_{m_1 m_2 m_3}$. The same should be done with every other triangle to achieve the entirety of our IFT set (D).

Example:

$$[\alpha_i^q, l_i^q, h_i^q, t_i] = [10,7,3,[0,0,1]] \rightarrow [[1,0,1,0],\ [0,1,1,1],\ [0,0,1,1],\ [0,0,1]] = f^q_{m_1 m_2 m_3}$$

3.1.2 Transformation of the IFTs into TFTs

With both Polar (C) and Delaunay (D) IFTs being generated, we move on to transforming them into their respective TFTs.

Polar Coordinate-based TFT

The transformation of the Polar IFT (C), involves subjecting each of its vectors to Feature Decorrelation and Random Projection. By the end of the process, we will have an array of complex valued vectors which will be stored for further use.

First we need to put each $P(m_i) \in C$, through the Feature Decorrelation Algorithm, or optionally, its extension which is referred to as the Enhanced Feature Decorrelation Algorithm. Overall, the enhanced version allows for tighter security at the expense of performance and matching accuracy. A larger value of L_S in the enhanced algorithm makes it more difficult to revert from S'_j to S_j, but at the same time would lower matching accuracy.

INPUT: the original feature $P(m_i)$
Process:
Step 1: Each position V_p of value 1 that is located in the binary string $P(m_i)$, is input into a folding function $f_P = \text{mod}(V_p, L_C)$, where L_C is an application-specific parameter and set to be smaller than l_C. mod(·) is a modulo operation, for example, mod(7,5) = 2 and mod(12,5) = 2.
Step 2: With each value of 1 in $P(m_i)$ relocated to a new position depending on f_P, a new binary string $\mathscr{P}(m_i)$ of length L_C is generated.
Step 3: $\mathscr{P}(m_i)$ is converted into a real-value vector through the Discrete Fourier Transform (DFT) as $\mathbb{P}(m_i) = DFT(\mathscr{P}(m_i))$.
OUTPUT: decorrelated feature $\mathbb{P}(m_i)$

INPUT: the original feature $P(m_i)$
Process:
Step 1: $P(m_i)$ is divided into two parts, P_1 and P_2, where P_1 contains the first L_C elements of $P(m_i)$, while P_2 contains the remaining elements.
Step 2: P_1 is evenly divided into N_S segments, so $P_1 = \{S_j\}_{j\in 1}^{Ns}$ and each segment S_j is of length $L_S = L_C/N_S$.
Step 3: For each segment S_j, calculate $\phi_j = sum(k * S_j(k))$, where $1 \le k \le L_S$. ϕ_j is a weighted sum of position k and its binary value for segment S_j. Then all the elements in S_j are circularly shifted left by ϕ_j bits to generate a permutated segment S'_j.
Step 4: All the permutated segments $\{S'_j\}_{j\in 1}^{Ns}$ are concatenated into a new binary vector P'_1, which is further concatenated with P_2 to create a binary vector $P'(m_i) = P'_1 \parallel P_2$
Step 5: Perform Steps 1 to 3 in Algorithm 1 with input $P'(m_i)$.
OUTPUT: decorrelated feature $\mathbb{P}(m_i)$

Figure 3.1: The Feature Decorrelation Algorithm (Left) and its Enhanced version (Right)

After acquiring the decorrelated complex valued vector $\mathbb{P}(m_i)$ from either of the Feature Decorrelation Algorithms, we will have to project it to a lower dimension Y using a random projection matrix $\mathbb{M}$:

$$\widehat{P}(m_i) = \mathbb{M} \times \mathbb{P}(m_i)$$

The random projection matrix itself is deterministically generated by a predetermined seed, which is a part of the public key used for the IFT to TFT transformation. The lower Y is, the stronger the irreversibility gets. But in return, we will end up with poorer matching accuracy. After we repeat the same procedure with all $P(m_i) \in C$, we obtain our Polar Coordinate-based TFT: $\widehat{C} = \{\widehat{P}(m_i)\}_{i=0}^{N-1}$.

Delaunay Triangulation-based TFT

Given the Delaunay TFT D and each of its vectors $f^q_{m_i m_j m_k}$, we aim to permute each of them, and place them each in a lengthy binary vector by their resulting integer representation. First, we convert each quantized attribute in $f^q_{m_i m_j m_k}$, which are α^q_i, l^q_i, h^q_i, and t_i into integers using the $int(\cdot)$ function, and obtain the largest integer out of them: $\mathbb{B} = max(f^q_{m_i m_j m_k} = [int(\alpha^q_i), int(l^q_i), int(h^q_i), int(t_i)])$. Next, we generate the feature code $f^c_{m_i m_j m_k}$ using the following formula, and another part of our public key ϕ which is a predetermined positive integer:

$$\mathbb{Y} = \mathbb{B} + \phi \tag{3.1}$$

$$f^c_{m_i m_j m_k} = \mathbb{Y}^3 int(\alpha^q_i) + \mathbb{Y}^2 int(l^q_i) + \mathbb{Y}^1 int(h^q_i) + \mathbb{Y}^0 int(t_i) \tag{3.2}$$

After obtaining the integer feature code $f^c_{m_i m_j m_k}$, we concatenate the entirety of $f^q_{m_i m_j m_k}$ into one binary string, convert it to integer, and add it to $f^c_{m_i m_j m_k}$, which we will refer to as the Index Number of $f^q_{m_i m_j m_k}$. After obtaining all the Index Numbers of each existing $f^q_{m_i m_j m_k} \in D$, we then create a zero vector $\widehat{D} = 0, 0, ..., 0$ that has great length. In our implementation we used a length of 2^{20}. Next, based on each Index Number we've retrieved from each $f^q_{m_i m_j m_k} \in D$, we replace a '0' by '1' in $\widehat{D}$.

For example if an Index Number is 12763, we will replace the 12763th element in $\widehat{D}$ by '1'. In case of a collision, we shall let the '1' remain. In case the Index Number is larger than than the length of $\widehat{D}$, we will use $mod(IndexNumber, length(\widehat{D}))$ to determine a feasible Index Number. Thus, $\widehat{D}$ on its own will become our Delaunay TFT.

3.1.3 Obtaining the Polar and Delaunay Matching Scores

Given two TFT sets, one belonging to the Target Fingerprint $(\widehat{C}_T, \widehat{D}_T)$ conceived during the Enrolment stage, and the other belonging to a Query Fingerprint $(\widehat{C}_Q, \widehat{D}_Q)$, we can compare them together and derive definitive scores which will be used for Fingerprint matching down the line.

The matching score resulting from the comparison of the Polar IFTs $\widehat{C}_T$ and $\widehat{C}_Q$, is referred to as `SC_MAX`. The other matching score derived from the comparison of the Delaunay IFTs $\widehat{D}_T$ and $\widehat{D}_Q$, is referred to as `SD`. Both `SC_MAX` and `SD` are ranged between 0 and 1. We will need both to calculate the `Final_Score` which is also ranged between 0 and 1.

Calculating `SC_MAX`

In order to derive `SC_MAX`, we plug in every possible pair of vectors $\widehat{P}_T(m_i) \in \widehat{C}_T$ and $\widehat{P}_Q(m_j) \in \widehat{C}_Q$, into the following formula, in order to derive a list of `SC` values:

$$\texttt{SC} = 1 - \frac{\|\widehat{P}_T(m_i) - \widehat{P}_Q(m_j)\|_2}{\|\widehat{P}_T(m_i)\|_2 + \|\widehat{P}_Q(m_j)\|_2} \tag{3.3}$$

Finally, we obtain `SC_MAX`, by finding the maximum value in the list of all derived `SC`s

Calculating `SD`

To obtain `SD`, we calculate the correlation coefficient of the $\widehat{D}_T$ and $\widehat{D}_Q$ by using the following formula:

$$\texttt{SD} = \frac{\sum_{k=1}^{L}(\widehat{D}_{Q,k} - \overline{\widehat{D}_Q})(\widehat{D}_{T,k} - \overline{\widehat{D}_T})}{\sqrt{\sum_{k=1}^{L}(\widehat{D}_{Q,k} - \widehat{D}_Q)^2 \sum_{k=1}^{L}(\widehat{D}_{T,k} - \widehat{D}_T)^2}} \tag{3.4}$$

In which L is the length of both $\widehat{D}$ arrays ($=2^{20}$ in our implementation), and $\overline{\widehat{D}}$ represents the mean value of said array. Also, the equivalent of the formula above can also be found in the MATLAB programming language as a function referred to as the `corr2()` function (MathWorks, 2023).

Calculating `Final_Score`

After calculating `SC_MAX` and `SD`, we can finally attribute a matching score to a pair of Target and Query fingerprints. The basic formula for `Final_Score` is as follows:

$$\texttt{Final\_Score} = (\rho_C \times \texttt{SC\_MAX}) + ((1 - \rho_C) \times \texttt{SD}) \tag{3.5}$$

The variable ρ_C signifies a score multiplier between 0 and 1, which varies depending on whether we'd want to rely more on the Polar or Delaunay scheme for matching the fingerprints. More emphasis is put on the Delaunay scheme the closer it is to 0, and vice versa more emphasis it put on the Polar scheme the closer it is to 1.

However, we would use a more elaborate formula when there are more than one comparisons involved, which is usually the case when we're working with a database of fingerprints, in which we can have many choices for picking a pair of fingerprints:

$$\texttt{Final\_Score} = (\rho_C \times norm(\texttt{SC\_MAX})) + ((1 - \rho_C) \times norm(\texttt{SD})) \tag{3.6}$$

Where $norm(\cdot)$ resembles a Min-Max normalization function. Before we use the more elaborate formula, we need to normalize the list of all derived `SC_MAX` and `SD` with the normalization function and then plug in the respective normalized `SC_MAX` and `SD` in the formula. We will discuss the uses of `Final_Score` for deriving fingerprint matching data in Chapter 4.

3.2 Tools and Implementation

One of the main challenges of implementing the Yang method was having to implement the whole method from scratch. Because unfortunately, the authors of the article under analysis did not put any links or references to their own implementation on their article. Upon further investigation, it was discovered that they've used paid frameworks such as VeriFinger for minutiae extraction and MATLAB for processing the extracted data. Due to budget limitations, it was not possible for us to use said proprietary software. Therefore we would have to resort to free open source solutions.

3.2.1 Python

For this project and our implementation needs, Python (version 3.9+) was our number one choice. Mainly because Python, due to its simple syntax and numerous open source easy to use libraries, is meant for quick prototyping and implementation of any concept or method. Had any other language been chosen, it would have taken far more than four months to implement the whole method and derive results from it.

Anaconda

Anaconda itself, is a Python distribution which is mainly meant for scientific programming purposes such as Data Science or Machine Learning. One specific feature of Anaconda which was vital to introducing GPU utilization to our implementation, is the ability to create isolated Python environments. In an environment it's possible to have Anaconda use a specific version of Python and Python packages so that compatibility between them for their intended purposes is guaranteed.

FingerFlow

FingerFlow is a python framework for extracting fingerprint minutiae and their respective properties directly from images as input, and returns the minutiae data as a Pandas dataframe (Arendáč, 2021). FingerFlow itself is based around two other Machine Learning and Deep Learning frameworks named TensorFlow and Keras respectively. It is also capable of utilizing CUDA drivers via TensorFlow's capabilities to speed up the minutiae detection.

For it to work, we need to supply it with a few prebuilt .H5 formatted model files, which are provided in the FingerFlow's Git repository. FingerFlow is also capable of matching fingerprints and providing us with a matching score. In Chapter 4, we will analyze FingerFlow's performance when it's utilizing the CPU versus when it utilizes the GPU.

NumPy

The NumPy library is our go to for mathematical operations used in our application, such as matrix multiplications, normalization, calculating norms, and so on. Another use for it, is for it to encode and save our calculated TFT data to a file using its `save()` method. The purpose of this is to be able to quickly load the TFT data for extracting statistical results, without having to go through the previous steps again. So as a result we will save a great amount of time in our experiments, at the cost of some disk space.

Numba

Numba is an open source Just-In-Time (JIT) compiler that translates a subset of standard Python code and NumPy functions into fast and efficient machine code. Numba's changes in performance have been significant when it comes to performing cumbersome mathematical operations at a high scale. Numba also has the capability to utilize CUDA for our calculations. As long as our function only consists of standard Python operations and supported NumPy functions, we can simply just add the following decorator `@jit(target_backend='cuda')` to the beginning of our qualified function definitions and we'll be able to benefit from its efficiency.

delaunay-triangulation

This library provides us with a ready to use Bowyer-Watson algorithm for deriving Delaunay Triangles from our minutiae set (Henkel, 2021). A certain class within this package, named `Vertex` is also used for representing our minutiae in an organized manner. To serve our purposes for minutiae representation better, the `Vertex` class is slightly modified at its source code to hold more attributes and methods.

3.2.2 Nvidia CUDA

CUDA (or Compute Unified Device Architecture) is a driver and API (Application Programming Interface) exclusive to Nvidia graphics cards that gives the developers direct access to parallel computational elements and the GPU's virtual instruction set. CUDA's parallel com-

putation capabilities is utilized by both FingerFlow and Numba for faster computation. So far in our implementation, the performance has only been raised linearly, and there's little room for optimization due to project time constraints and a lack of hands-on CUDA programming experience.

3.2.3 Hardware

Our implementation has been successfully tested on two separate computer platforms, and runs on both platforms without flaw. One being a home desktop (GPU only) running on the Windows 10 Pro operating system. The other being a Lenovo ThinkPad model T430s laptop (CPU only) running on the Fedora 37 operating system. So far, mostly the home desktop has been used for conducting test cases, due to the significant runtime speed that its GPU provides. The full specifications of both platforms are provided in the figure below.

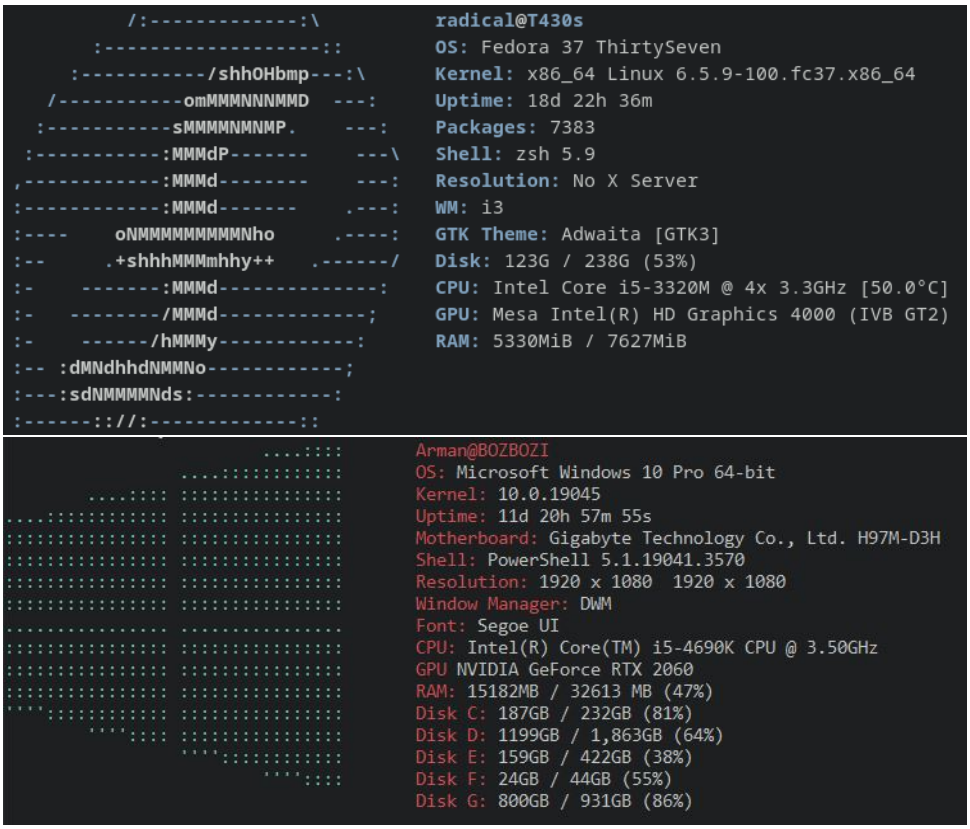

Figure 3.2: Full system specifications of both Linux and Windows platforms

Chapter 4

Evaluation

In this chapter, our sole focus will be on the algorithm's fingerprint matching accuracy, runtime performance, and memory usage. We will use the same exact fingerprint datasets used in the article for conducting our matching accuracy tests. Unfortunately, due to time constraints, it was not possible to conduct security related tests, such as testing ARM attack invulnerability, revocability, and unlinkability.

4.1 Brief Introduction to Matching Metrics

To determine the matching accuracy of any method, we need to first of all process a fingerprint dataset and derive all the matching scores between all possible pairs of fingerprints. Next, we'll declare a variable named the Acceptance Threshold t_A which is ranged between 0 and 1. The use for this variable is to declare whether two fingerprints match each other and belong to the same finger or not. If a fingerprint pair's matching score is higher than t_A, then the fingerprint pair are considered to be matching each other and accepted as a result. If on the other hand, the matching score is lower than the threshold, the fingerprint pair is deemed as rejected.

4.1.1 GAR, FAR, FRR, and EER

A few important factors for deriving the matching accuracy of our method are the Genuine Acceptance Rate (GAR), False Acceptance Rate (FAR), False Rejection Rate (FRR), and Equal

Error Rate (EER).

Under a certain Acceptance Threshold t_A, the GAR is the ratio of the number of accepted fingerprint pairs to the number of all fingerprint pairs taken from the same finger, otherwise known as genuine tests. The FAR is the ratio of the number of accepted fingerprints to the number of all fingerprint pairs taken from different fingers. The FRR is basically equal to 1-GAR.

As we can see, we can get different values for the GAR, FAR, and FRR under different acceptance threshold values. Therefore we can deduce that the GAR, FAR and FRR are essentially functions parametrized by t_A. Finally the EER's definition is as following:

$$\text{If } t_A \text{ is such that: } \mathrm{FAR}(t_A) = \mathrm{FRR}(t_A) \text{ then: } \mathrm{EER} = \mathrm{FAR}(t_A) = \mathrm{FRR}(t_A)$$

By starting at 0 and incrementing t_A by 0.001, we can eventually get to the point where the FRR and FAR meet. The lower the EER value, the more accurate our matching system is.

4.2 Reliability of FingerFlow

First we need to look into FingerFlow itself and see whether it works as intended and gives us the right minutiae to work with or not. One way to confirm this, would be to give FingerFlow a dataset of fingerprints, and then have it calculate the matching scores for every pair using its own built in method for matching. Then an ROC curve will have to be drawn based on its GAR and FAR.

Unfortunately, we haven't been able to utilize FingerFlow's matching functionality due to a lack of documentation and not having enough time to reverse engineer the source code. So the only thing we have to work with, is the developer's own ROC curve plot which is based off of an unknown fingerprint dataset, and multiple matching modules referred to as VerifyNet which come in different versions.

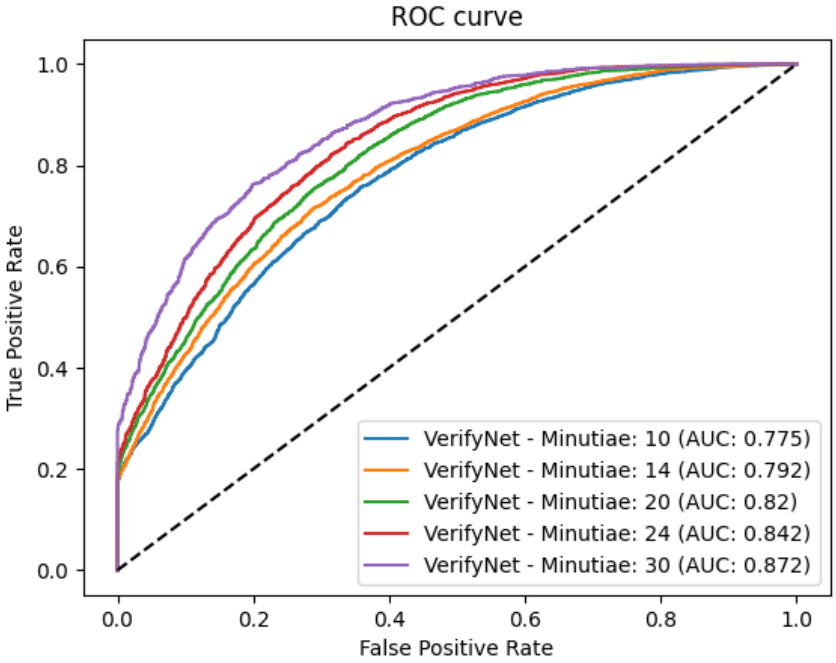

Figure 4.1: FingerFlow's ROC curve generated by different VerifyNet modules

4.3 Matching Accuracy of the Yang Method

The fingerprint datasets used for this section are: FVC 2002DB1, FVC 2002DB2, FVC 2002DB3, FVC 2004DB4. Each of which include 80 fingerprint images in total, which are 8 different impressions of 10 different fingerprints. We also have two different protocols involved for the selection of fingerprint pairs. The default protocol considers the set of all pair selections from the whole dataset, but the 1V1 protocol only considers the set of all pair selections from the first two impressions of each fingerprint, resulting in a smaller dataset to work with.

4.3.1 Case 1: Both Schemes vs. Delaunay Only vs. Polar Only

In this section, under the 1V1 protocol performed on the FVC 2002 DB2 dataset, using the regular Feature Decorrelation Algorithm, and a projection matrix dimension Y of 300, we are going to derive three ROC curves from three instances where both schemes, only the delaunay scheme, and only the polar scheme was used for obtaining the final score. Below are the ROC

curve figures on the paper compared to the one we derived from our implementation:

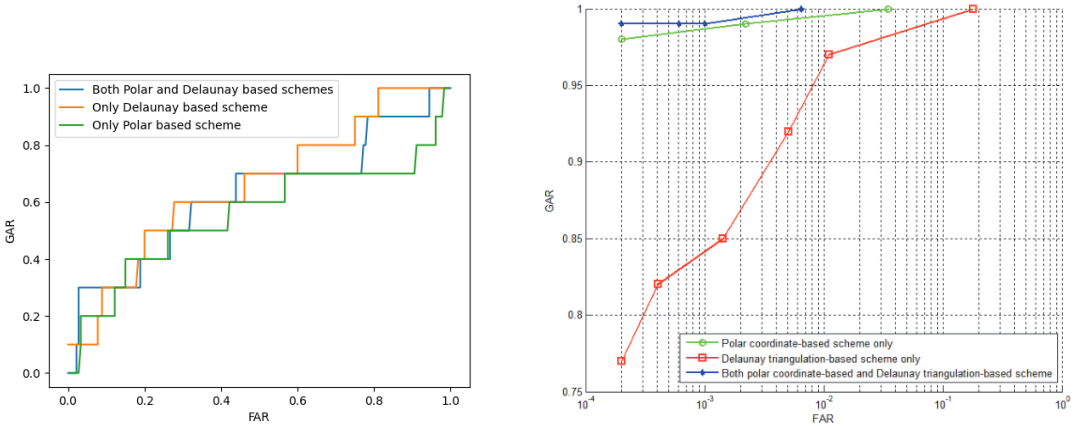

Figure 4.2: The three ROC curves derived from our and the author's implementation

The resulting EERs we got from the three instances are as follows: Delaunay only EER = 0.3986, Polar only EER = 0.434, Normal only EER = 0.3986.

4.3.2 Case 2: Feature Decorrelation Algorithm Parameters

In this test case we're going to see the effects of changing the parameters within both the enhanced and normal versions of the Feature Decorrelation Algorithm, namely L_C and L_S

4.4 Runtime Performance

Chapter 5

Conclusion

5.1 What has been achieved

5.2 What is left to be achieved

References

Arendáč, J. (2021). Fingerflow. https://github.com/jakubarendac/fingerflow.

Delaunay, B. (1934). Sur la sphere vide. *Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles*, *6*, 1934, 793–800.

Henkel, L. (2021). Python delaunay triangulation. https://gitlab.com/LittleLeonie/delaunay-triangulation-py.

Lindenstrauss, J., & Johnson, W. (2018). Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, , 2018, 189–206.

Maltoni, D., Maio, D., Jain, A. K., & Feng, J. (2022). *Handbook of fingerprint recognition.* Springer Nature Switzerland AG.

MathWorks (2023). Matlab documentation - 2d correlation coefficient. https://www.mathworks.com/help/images/ref/corr2.html.

Yang, W., Hu, J., Wang, S., & Wu, Q. (2018). Biometrics based privacy-preserving authentication and mobile template protection. *Hindawi Wireless Communications and Mobile Computing*, *2018*, June, 2018.

Appendix A

Code

A.1 Bowyer-Watson Pseudo-Code

```
def BowyerWatson (pointList):
    #pointList is a set of coordinates defining the points to be triangulated
    triangulation := empty triangle mesh data structure
    # must be large enough to completely contain all the points in pointList
    add super-triangle to triangulation
    # add all the points one at a time to the triangulation
    for each point in pointList:
        badTriangles := empty set
        # first find all the triangles that are no longer valid due to the
            insertion
        for each triangle in triangulation:
            if point is inside circumcircle of triangle:
                add triangle to badTriangles
        polygon := empty set
        # find the boundary of the polygonal hole
        for each triangle in badTriangles:
            for each edge in triangle:
                if edge is not shared by any other triangles in badTriangles:
                    add edge to polygon
```

```
        # remove them from the data structure
        for each triangle in badTriangles:
            remove triangle from triangulation
        # re-triangulate the polygonal hole
        for each edge in polygon:
            newTri := form a triangle from edge to point
            add newTri to triangulation
    # done inserting points, now clean up
    for each triangle in triangulation:
        if triangle contains a vertex from original super-triangle:
            remove triangle from triangulation
    return triangulation
```