BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

EVALUATION OF FINGERPRINT MORPHING METHODS COMPARED TO A REAL SYSTEM

ZHODNOCENÍ METOD MORPHINGU OTISKŮ PRSTŮ OPROTI REÁLNÉMU SYSTÉMU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. DANIEL KOLÍNEK**
AUTOR PRÁCE

SUPERVISOR **Prof. Ing., Dipl.-Ing. MARTIN DRAHANSKÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Kolínek Daniel, Bc.**
Program: Informační technologie
Obor: Počítačové vidění
Název: **Zhodnocení metod morphingu otisků prstů oproti reálnému systému**
Evaluation of Fingerprint Morphing Methods Compared to a Real System
Kategorie: Umělá inteligence

Zadání:

1. Prostudujte literaturu týkající se zpracování snímků otisků prstů, zejména extrakce vlastností otisku prstu z papilárních linií.
2. Navrhněte algoritmus pro morphing otisků prstů na základě extrahovaných vlastností z papilárních linií.
3. Výše navržený algoritmus implementujte, otestujte na zadaném datasetu a dostupném softwaru od společnosti Innovatrics a porovnejte s již existujícím řešením vytvořeným v rámci projektové praxe.
4. Zhodnoťte dosažené výsledky a diskutujte možná rozšíření.

Literatura:

- ZAERI, Naser. Minutiae-based fingerprint extraction and recognition. *Biometrics*, 2011.
- FERRARA, Matteo; CAPPELLI, Raffaele; MALTONI, Davide. On the feasibility of creating double-identity fingerprints. *IEEE Transactions on Information Forensics and Security*, 2016, 12.4: 892-900.
- MSIZA, Ishmael S., et al. On the introduction of secondary fingerprint classification. *State of the art in Biometrics*, 2011, 105.
- PATRICIU, Victor-Valeriu; SPINU, Stelian. Fingerprint Ridge Frequency Estimation in the Fourier Domain. *Advances in Electrical and Computer Engineering*, 2014, 14.4: 95-98.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 30. července 2021
Datum schválení: 11. listopadu 2020

Abstract

The aim of this work is to design methods of fingerprint morphing, subsequently to implement the proposed methods and to test their effectiveness by breaking into real verification systems made by Innovatrics and Neurotechnology, two prominent companies in biometrics. Methods for morphing were designed mainly based on the article On the Feasibility of Creating Double-Identity Fingerprints where the part of the preparation of fingerprints on the input was modified and an adjustment of the output was made so that it is able to work with fingerprint images taken under different conditions. Fingerprints created by morphing were recognized as genuine in 41.72 % of cases when compared to the default threshold. The main benefit of this work is the created application for generating fingerprints using morphing and to point out that this method is a threat to existing systems used for fingerprint recognition.

Abstrakt

Cílem této práce je návrh metod morphingu otisků prstů, následná implementace navržených metod a otestování jejich účinnosti prolomení do reálných systémů verifikace od dvou významných společnostmi v biometrii Innovatrics a Neurotechnology. Při návrhu metod jsem převážně čerpal z článku On the Feasibility of Creating Double-Identity Fingerprints, kde byla upravena část přípravy otisků na vstupu a byla přidána úprava výstupu tak, aby byl schopný pracovat se snímky otisků prstů, které jsou pořízeny za rozdílných podmínek. Otisky vytvořené morfingem byly ve 41.72 % rozpoznány jako pravé při porovnání oproti výchozímu prahu. Přínosem této práce tedy bylo vytvoření aplikace pro generování otisků prstů pomocí morfingu a poukázání na fakt, že tato metoda je hrozbou pro stávající systémy používané pro rozpoznání osob pomocí otisků prstů.

Keywords

Fingerprint, fingerprint classes, morphing, extraction, generation, sensors, comparison

Klíčová slova

Otisk prstu, třídy otisků, morphing, extrakce, generování, senzory, porovnání

Reference

KOLÍNEK, Daniel. *Evaluation of Fingerprint Morphing Methods Compared to a Real System.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D.

Rozšířený abstrakt

Diplomová práce se zabývá tématem generování otisku prstu pomocí metod morphingu a následného ohodnocení těchto metod oproti reálným systémům od společnosti Innovatrics a Neurotechnology.

Otisky prstů se nám mohou jevit jako ideální biometrická vlastnost pro identifikaci osob. Je velice akceptovatelná (lidem nijak nevadí přiložit svůj prst na senzor), jednoduše získatelná a zůstává prakticky neměnná po celou dobu života, pokud nepočítáme hluboká zranění v oblasti otisku prstu. Z tohoto důvodu se stala tato biometrická vlastnost velice populární a její využití sahá od kriminalistiky až po potvrzení platby chytrým telefonem. Samostatný otisk prstu je tvořený papilárními liniemi a údolími mezi nimi na vnitřní straně konečků prstů. Jak jsou papilární linie s údolími formovány je dáno kombinací genetického kódů a výsledku náhodných vlivů na dítě v matčině lůně. Díky tomuto faktu se jedná o unikátní biometrii, a to i při porovnání mezi jednovaječnými dvojčaty.

Teoretická část obsahuje informace potřebné pro porozumění návrhu a řešení diplomové práce. Kapitola začíná obecným popisem biometrických vlastností člověka. Dále je zde vysvětlen rozbor kůže a popis vlastností otisků prstů jako jsou třídy, markanty, pole lokálních orientací papilárních linií nebo frekvenční charakteristika. Následuje popis možností snímání otisků prstů pomocí různých technologií, včetně extrakce dříve popsaných vlastností, a porovnání otisků vůči šabloně s popisem možných chyb, které mohou při porovnání nastat.

Dále jsou popsány tři navržené metody generování otisků prstů založené na morphingu, které pracují podobným způsobem.Prvně je obraz normalizován, je použito metody CLAHE pro vyrovnání kontrastu pomocí histogramu a následně je oříznuto pozadí. Otisky musí mít stejné rozlišení, proto otisk s menším rozlišením je zmenšen na rozlišení druhého se zachování pomeru stran. Následně nastává nasazení jednoho otisku na druhý, kde první metoda pracuje s nasazením pomocí podobnosti orientovaných polí a druhá metoda s překrytím masek otisků. Poté je jako maska výsledného otisku určena jako průnik nasazení. Dále jsou předpočítány markanty, orientovaná pole a frekvenční charakteristiky nasazeného otisku. Následně je určeno barycentrum pro dělící přímku. První metoda určí barycentrum jako jádro otisku prstu a druhá metoda jako střed otisku prstu. Pro výpočet optimálního natočení dělící přímky je přímka rotována okolo barycentra. Poté je určena taková pozice, která maximalizuje počet markantů, podobnost orientovaných polí a podobnost frekvenční charakteristiky daných otisků. Následně pod dělící přímkou se nachází vlastnosti z prvního otisku prstu a nad ní z druhého otisku prstu. Pro první a druhou metodu, které provádí morphing otisků prstů na základě obrázku, započne generování tak, že čím dále se picel nachází od dělící přímky, tím více je obsaženo původního otisku a čím blíže tím více se otisky prolínají. Třetí metoda, která zůstala pouze u návrhu z důvodu dřívějších horších výsledků a rozdílného téma, rozdělí výsledný otisk stejně, ovšem pouze bere vlastnosti otisku (markanty, pole lokálních orientací a frekvenční charakteristiku) a provádí generování otisku prstu ze šablony.

Kapitola implementace popisuje aplikace, které byly implementovány autorem v rámci této diplomové práce. Začíná konzolovou aplikací morphinu, která byla implementována v jazyce Python pro demonstraci prvních dvou navržených metod a vy. Další dvě aplikace jsou implementovány v jazyce C++ za pomoci nástroje od společnosti Innovatrics. První aplikace slouží k rozřazení otisků do tříd a druhá slouží k porovnání otisků. Poslední dvě implementované aplikace slouží k verifikaci pomocí nástroje Verifinger od společnosti Neurotechnology. První aplikace je implementována v jazyce C s použitím Verifinger v6 a

následně druhá v jazyce C++ s využitím Verifinger v12.1. Samozřejmě nechybí ani popis volaného API pro nástroje jak od společnosti Innovatrics tak Neurotechnology.

Poslední kapitola Solution evaluation před závěrem začíná popisem použitých databází otisků prstů EBD a FVC2002. Jedním z výsledků je i nově vygenerované databáze morphovaných otisků pomocí dvou implementovaných metod s použitím databáze otisků prstů EBD. Databáze obsahuje celkem 9 982 morphovaných otisků prstů.

V prvním kroku pro vyhodnocení bylo vygenerováno 1 315 otisků prstů první morphovací metodou s použitím databáze otisků prstů FVC2002. Takto vygenerované otisky byly porovnány oproti VeriFingeru v6 s úspěšností 76.3 %, přičemž tenhle výsledek sloužil k porovnání oproti článku, kde dosáhli úspěšnosti 81.1 %. K mírně horšímu výsledku mohlo dojít výběrem jiných otisků pro morphing (autor uvádí pouze, že byly vybrány náhodně) a následné mírné úpravy, jelikož ne vše bylo dostatečně v článku vysvětleno. Následuje ohodnocení výsledků oproti aplikaci porovnání implementované pomocí Innovatrics a Verifinger v 12.1. Pro vyhodnocení je použita nově vygenerovaná databáze morphovaných otisků prstů. První metoda morphingu zde dosahuje úspěšnosti 24.7 % oproti Innovatrics a 27.9 % oproti Verifingeru. Druhá metoda je úspěšná v 41.7 % případů v porovnání oproti oběma aplikacím. Ve vyhodnocení jsou porovnány jednotlivé třídy, přičemž došlo ke zjištění, že některé třídy mají tendenci být úspěšnější než jiné a to nejen z důvodu charakterisk dané třídy, ale také díře v daných systémech.

Evaluation of Fingerprint Morphing Methods Compared to a Real System

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Prof. Ing., Dipl.-Ing. Martin Drahanský. Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Daniel Kolínek
July 26, 2021

Acknowledgements

I would like to thank Prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D. for leading this work. I would also like to thank Ing. Ondřej Kanich, Ph.D. for guiding me during implementation and providing a database needed for testing the final application. And last but not least, to my family for their support and background in creating this work.

Contents

Chapter 1

Introduction

Technology is evolving faster every year which also applies to biometric systems. Fingerprint-based biometric systems are gaining acceptance faster than ever as one of the most effective technology for authentication. Nowadays we use fingerprint-based biometric features not only to unlock our smartphones but even to log into our bank accounts or as a proof of identity on borders in some countries. But as always in technology, when something moves forward the attackers do as well. In the case of bank accounts or proof of identity, nobody wants to be cracked.

There is a lot of effort and work put into making sensors and the whole process more robust. In the case of sensors liveness detection can be used as an example but this feature, as many others, can not be used in every situation mostly because of its size. As an example, it can be used in smartphones. And the second part, that is instantly improving, is the algorithm itself. For improving the algorithm there is a lot of testing needed, which comes with the need for a big database of fingerprints. The database should not only have a lot of fingerprints from one finger under different angles but also a lot of fingerprints from different people. Therefore there are a lot of algorithms for generating fingerprints used just to make larger databases rather than for attack systems.

This work is mainly focused on known fingerprint generation by morphing. This field of fingerprint generation was not much researched yet. Where basically only one paper was made public in 2017 and it includes a lot of uncertainties. This work is trying to describe these uncertainties and to find improvements in the already described process of morphing and lastly to test the solution against the used biometric systems for fingerprint matching made by two prominent companies in biometrics.

Chapter two describes the theory needed before getting familiar with the algorithm itself and understanding its principles. Included in this Chapter is not only the theory needed for the implementation but also the basics about fingerprints and capturing fingerprints with sensors. The third Chapter describes the proposal of all stages of an improved algorithm based on the paper mentioned before. The fourth Chapter describes the implementation of a morphing application and also the second part of this work which is the implementation of fingerprint matching scripts with the use of tools from companies Innovatrics and Neurotechnology. Lastly in the fifth Chapter, there are the results of these two scripts compared against implemented algorithm with the use of a database provided by the Faculty of Information Technology at the Brno University of Technology.

Chapter 2

Biometrics

This Chapter describes all the theory needed to understand the solution of this thesis, which is the application for morphing fingerprints and applications for matching fingerprints. This thesis deals with fingerprint generation but also includes the comparison of results against real working solutions. The comparison includes an understanding of fingerprint recognition techniques. Both problems fall into the field of biometrics.

2.1 Introduction to Biometrics

It is necessary to explain from the beginning, what do we imagine under the word biometrics in the field of information technology. The term biometrics in our context means the automatic recognition of a person on the basis of their characteristic physiological and behavioral properties. Three basic methods are defined for recognizing people: a) what a person knows, b) what a person has, c) what a person is (in biometrics, for example, just a fingerprint). While recognition methods based on the knowledge (eg. password) or on the ownership (eg key) can be forgotten or lost, guessed or stolen, shared in some way. [30] Since we still have the biometric properties of our body with us, they are mostly not possible to lose and can't be forgotten at all. However, this is also the biggest weakness of biometric systems, which basically means once some biometric feature is discovered, we no longer have the opportunity to change it. [39]

There are many important terms in the field of biometrics. Interclass and intraclass variability are among them. Interclass variability says how big is the difference between individual classes (in our case the classes are people and we want this feature to be as different as it can be). On the other hand, there is the intraclass variability, which sets the differences between the same class (in our case, it is the same person). There are seven basic features for comparison of biometric characteristics: [39]

- Universality = Each person should have this feature,
- Uniqueness = The feature should differ for each person,
- Permanence = The feature should stay the same over the time,
- Measurability = Biometrics is easily acquired,
- Performance = Recognition accuracy, speed, resource requirements, etc.,
- Acceptability = Willingness to capture the feature,

- Circumvention = Difficulty to create working fake feature. [39] [29] [15]

There is no expectation at all that one particular biometrics will be perfect in all these characteristics. This means that no biometrics is ideal but most of them are acceptable. When choosing which biometrics to use the decision needs to be made based on the requirements of an application and the features of that biometrics.

2.2 Fingerprint

The fingerprint is made by the representation of the epidermis, where the pattern is made by interleaving ridges and valleys. The epidermis itself is in the highest level creating skin together with the dermis, called real skin, and subcutaneous (fat) layer make up skin (see in Figure 2.1). [29] [23]

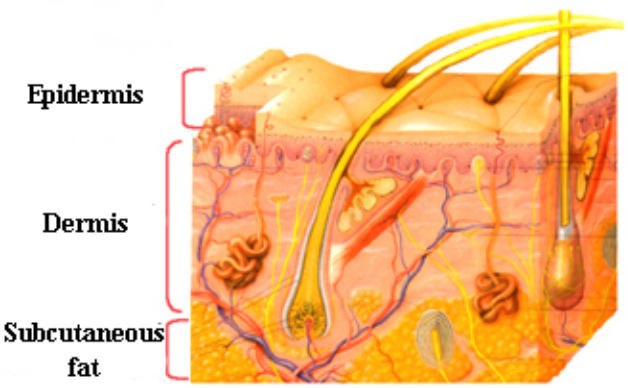

Figure 2.1: Skin structure [6]

A combination of genetic and environmental factors create the pattern of the ridges in the process of growing capillaries and blood vessels in angiogenesis. Basically, the forming of skin is made by genetic code but the specific way that skin will look like is the result of random acts of mother's womb during infancy. This is the reason why there is no identical biometrics even in the case of identical twins. The final look of the fingerprint is made approximately in the first seven months of fetal development and then it is the final form. That means that the configuration of papillary lines stays the same throughout a life of a person except for some injuries. And this makes our fingerprints a very attractive biometrics feature for identification. [39] [23] [8]

Typically the fingerprints are taken in the form that the ridges are dark lines in contrast with the valleys which are light (Figure 2.3). Ridges are usually from 100 µm to 300 µm wide and the valleys are around 500 µm wide. Injuries are usually only small scratches or so, and these small injuries usually do not go to the underlying structure of the ridge so the pattern stays unchanged after regeneration. [39] [23]

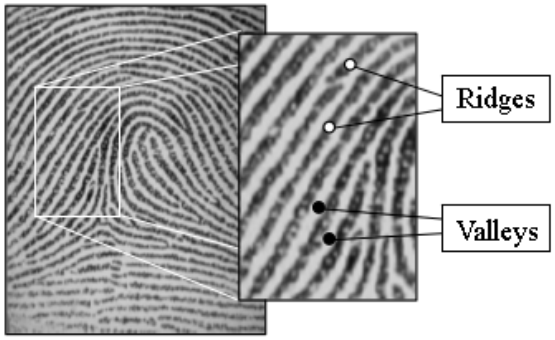

Figure 2.2: Fingerprint [39]

2.2.1 Improvements of Fingerprint Image and Binarization

The quality of input images plays a big role in algorithms that implements minutiae extraction and fingerprint recognition. Actually, in real applications even bad conditions like too wet/dry skin, small injuries, bruises, not enough pressure on the fingerprint sensor or bad fingerprints (older people or hand working people) result in a high probability (around 10 %) that the fingerprint is not acceptable. [33]

Therefore the algorithm for improving fingerprints images is used. It improves brightness, the clarity of ridges in places where it is still possible and it marks places where it is not possible as too noisy for next processing. The most common way how to do the improvement is based on context filters. Characteristics of the filter are change based on the local neighborhood in the context filtering. The change is mostly based on local ridge orientation and local ridge frequency. The unwanted noise and preservation of the true structure of the ridge and valley can be achieved by choosing the right filter. [29] [11]

Figure 2.3: The result after image binarization is shown on the right. [39]

2.2.2 Fingerprint Analysis

Ridges contain a lot of information and there are three basic levels that describe the amount of extracted information. The first level extracts only the global pattern of ridge flow. The

second contains information about minutiae and the third is the most complex and includes information about pores, the local shape of ridge edge, and even more. [13] [39]

When we look at ridges globally (on the 1st level), they usually run smoothly and parallel but there can be found at least one area where some changes can be found like high curvature of ridges or frequent ridge endings. We call these areas singularities or singular areas and they can be divided into three topologies which are *whorl*, *delta*, *loop*. Areas that contain these three topologies are usually marked with these shapes: ∩, Δ and O. [41] [39]

The alignment of fingerprints can be done in some applications by aligning two fingerprints *core* on the core. The core of fingerprint was defined by Edward Henry as „The northernmost point of the innermost ridgeline“. In the end, everything depends on what is used in the applications and it is the point of the center of the singularity of the northernmost loop type (e.g. those that belong to the arch class in the Figure 2.5). Sometimes it is hard to detect core because some fingerprints simply do not contain whorl or arch, like those which fit into the class of the arch 2.5. In these cases, the core is identified as a point with maximal curvature of the ridge. Because of the high variability of fingerprints, it is hard to identify the core in all the images of fingerprints. Singular points that are used for classification can be seen in the Figure 2.5 and it is used in the way to accelerate the classification by working only over fingers with the same class. [13]

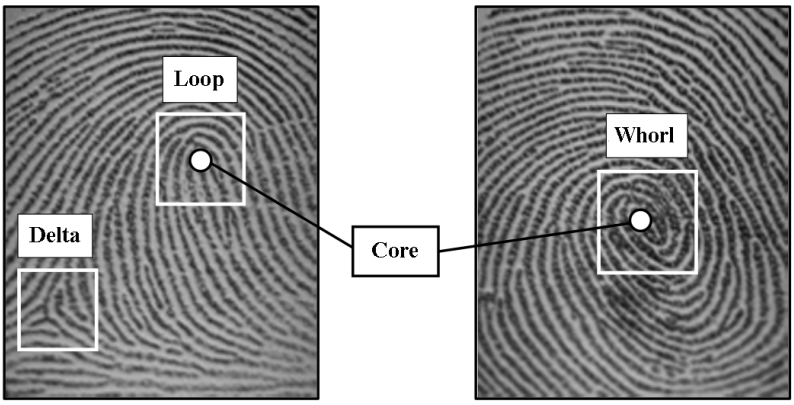

Figure 2.4: Singularities of a fingerprint [39]

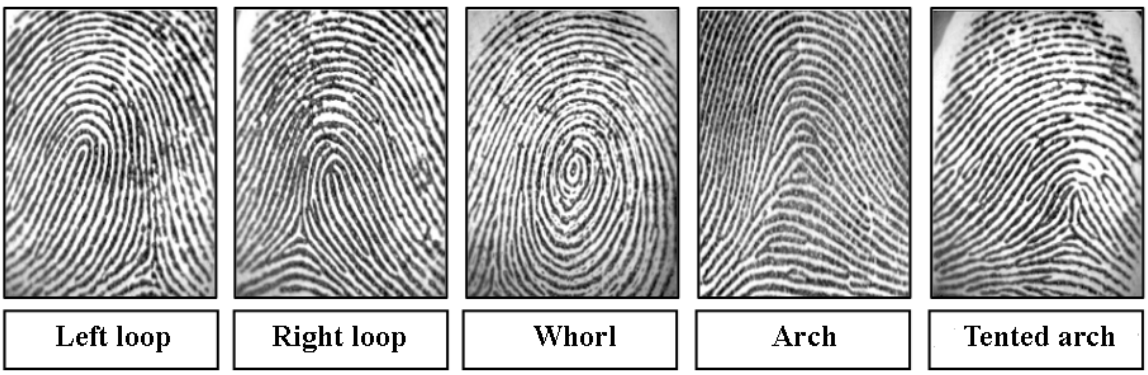

Figure 2.5: Each fingerprint is representative of each of the five classes [39] [22].

Important points, called minutiae, can be found in the pattern on a local level (on the 2nd level). Most of the known and used methods for minutia extraction work with binarized images for which the image needs to be converted to a grayscale image. After the binarization, the process called thinning is applied which results in ridges of only 1px in width. At the end of the scanning of the image those pixels which are known as minutiae can be detected. There are many minutiae (some of them can be seen in the Figure 2.6), but most implementations use just two which are: 1) ridge ending (termination) or ridge division (bifurcation). Most applications use just these two because of their higher accuracy. [29] [59]

American National Standards Institute (ANSI) has designed a taxonomy of minutiae-based on four classes: termination, bifurcation, trifurcation, and not known. But even the model used by FBI works only with termination and bifurcation. [39]

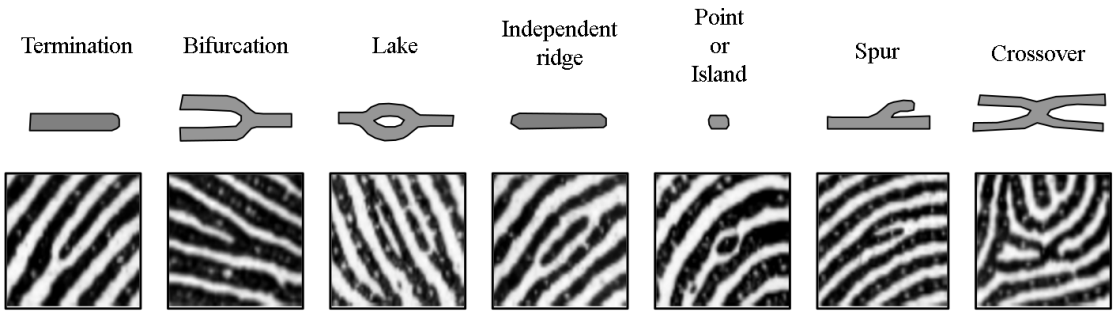

Figure 2.6: Seven most common minutiae [39]

As it was written before there can be taken even more information from the fingerprints, for example by pores detection (on the 3rd level). The problem of this level is that pictures of fingerprints need to be taken at very high resolution, which means basically 1000 dpi or more. But better results are achieved in the result of using just detection of pores and with a combination of these two methods, significant improvements are reached in terms of both. As already stated the problem is the high resolution of images, so in spite of the achieved improvements this method is not used very often. [29] [59]

2.2.3 Local Ridge Orientation

Local ridge orientation can be counted for every arbitrary neighborhood that some ridge comes through. Then the local orientation of the actual ridge at pixel $[x, y]$ is the angle θ_{xy} in which ridge is coming through the arbitrary big neighborhood with a center in $[x, y]$ and arbitrarily forms an angle with the horizontal axis. Because ridges are not coming in specific direction, angle θ_{xy} is assigned as unoriented direction in $[0...180°]$. Because of the complexity of the computation. Many algorithms work only on discrete positions, which brings even more computing efficiency, and orientation for other pixels is computed by interpolation. [25] [24]

This local ridge orientation is matrix D whose elements are coding local orientation of papillary lines from fingerprints (see in Figure 2.7). Each element θij which corresponds to the element $[i, j]$ in a matrix corresponds to pixel $[x, y]$ then indicates the average orientation of the local neighborhood. Value r_{ij} is often computed for every element $theta_{ij}$. This value r_{ij} stands for reliability or consistency of computed orientation in each block. The high

value of r_{ij} means that the picture of the fingerprint is in very good resolution and no big damage is contained in the neighborhood. [39]

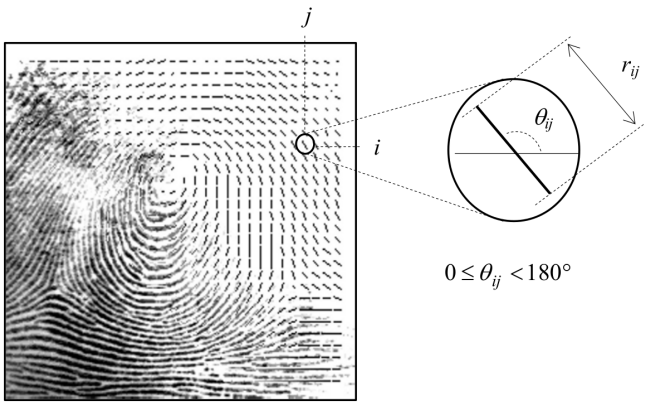

Figure 2.7: The transition of a fingerprint image to the corresponding orientation image is calculated over a $x \times x$ square block grid. Each element indicates the local orientation of the fingerprint papillary lines. [39]

2.2.4 Local Frequency of Ridges

Local frequency of ridges f_{xy} in pixel $[x, y]$ basically means the number of ridges in some window of given width placed over the hypothetic segment with the center in $[x, y]$. Two rules for the window have to be satisfied. The first, is that it has to be placed perpendicular to the local orientation of the ridges θxy. And the second, is that if the frequency is estimated over discrete positions, the orientation field has to be computed over discrete positions as well. The frequency differs in every fingerprint and can even differ in different areas of the same fingerprint. [39]

The local ridges frequency estimation can be computed as the average count of pixels between two consecutive peaks of gray in a grayscale image in the perpendicular direction to the local orientation of papillary line (Figure 2.8). For this purpose, the fingerprint is cut by a plane parallel to the z axis and perpendicular to the local ridge orientation. [26]

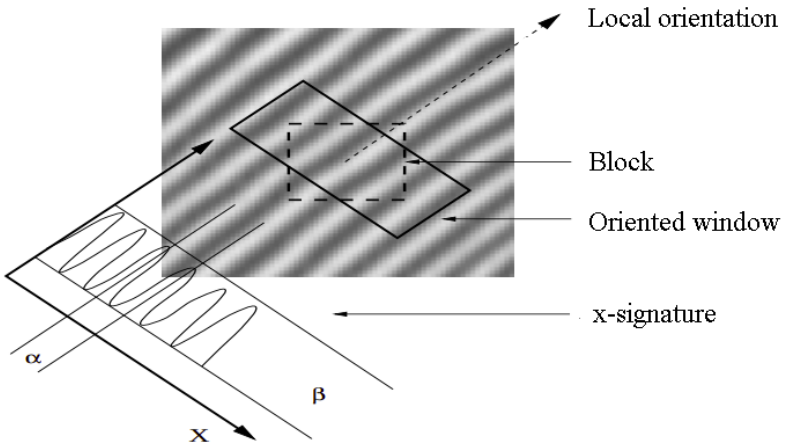

Figure 2.8: Oriented field and x-signature. [26]

2.3 Fingerprint Sensors

Most AFIS (Automated Fingerprint Identification Systems) for criminal and commercial use accept digital images on the input as the result of live scanning acquired by capturing fingerprints by an electronic fingerprint sensor. That means no ink is used and the scanned person has to do only one thing and that is a slight push with his finger to the flat surface of the scanner used for live scanning. The sensor is the most important thing of the whole scanner because it is the part that creates the fingerprint image. There are three basic types of sensors: [29]

1. **Optical sensors:** FTIR (Frustrated Total Internal Reflection) is the oldest and most commonly used technique to capture live scans of fingerprints. The user pushes his finger against the upper side of a scanner, which means that ridges are touching the flat glass surface and valleys are in a short distance. The left side of the prism is illuminated by diffused light. Ridges absorb the light coming through the prism to the finger and valleys reflect the light. Thanks to this feature it is easy to decide whether it is a ridge or valley. The light rays are coming from the right of the prism and are focused with the lens on a CCD or CMOS sensor. [29] [40]

2. **Solid-state sensors:** Solid-state sensors have been commercially used since the first half of the 90s. Silicon technology for fingerprint live scanning is made from a lot of small pixels and all those pixels are basically small sensors. There is no need for any optical sensor at all, a user just slightly pushes his finger against the surface of the sensor and his fingerprint is scanned. The scanner works on acquiring the physical feature and converts the feature to an electric signal. There are four most used types of silicon sensors: capacitive, piezoelectric, thermal, and electric-field. [29] [54]

3. **Ultrasound sensors:** This last type can be seen also as a type of ultrasound. The sensor operates by sensing the acoustic signal sent against the finger so by the echo signal the range of depth for fingerprint image can be counted. This method can be useful in places like hospitals or labs because these sensors are scanning subsoil of fingerprint, which means, it can detect the fingerprint over the fluids or even gloves. [29] [38]

2.3.1 Fingerprint Comparison

One fingerprint can be captured in many different ways, which makes reliable fingerprint comparison is very difficult problem. Movement, rotation, part overlay, nonlinear coloring, variable pressure, skin changes, noise, mistakes in feature extraction these are all huge factors that are responsible for intra-class variability. This means that a situation can happen, where two images of two different fingerprints will have way more in common, than two images of fingerprints of the same person. [9] [56]

This is the reason why professional dactyloscopy has to work with many factors before the final verdict if it is decided that the two fingerprints are from the same person. The factors are: 1. a fingerprint has to be from the same class, 2. quantitatively the corresponding features of minutiae must be identical, 3. quantitatively a certain number of minutiae must be the same for the examined fingerprints (for example in the USA the minimum is 12 same minutiae), 4. the corresponding features of the compared minutiae must be identical. Special protocols were defined for comparison and there was even a special diagram created

to help the detectives with dactyloscopy if they ever need to compare fingerprints manually. [9] [56]

The same rules don't have to be followed in the automatic (commercial) comparison of fingerprints. Actually many new methods were created in the last 50 years which are used only for automation although the basics for detecting minutia and comparison of two fingerprints are based on these rules. It is caused by the need for faster detection but many times at the price of worse results. There are three main ways how to compare fingerprints: [29] [39]

- **Comparison based on correlation similarity:** It is called correlation because it computes the correlation of two fingerprints that are overlaid. The correlation is computed over some rotation and movement. [29] [39]
- **Comparison based on minutiae:** First, all minutiae from both fingerprints are extracted and saved into a two-dimensional array as a set of points. And then the matching score of the minutiae basically consists of finding an alignment between the template and the sets of input minutiae, which leads to the maximum number of pairings of minutiae. [29] [39]
- **Comparison based on functions working without minutiae:** As mentioned before the extraction of fingerprint minutiae from images of fingerprints with bad quality or resolution is almost impossible although other features, like frequency of ridges, shape, or some information of the structure, is still possible to extract in pretty good shape even though their matching capability is not that impressive. Algorithms that are working with this approach are using features extracted from the pattern of ridges. [29] [39]

When somebody says that the biometrics system works in identification mode, it means that the system works in comparison one to many. That means that the system has N templates and now it wants to compare our one fingerprint with the whole database. If the comparison is made only against the users that are registered in the database then this comparison is called closed identification. Closed identification always returns an array of possible candidates that is not empty. In the opposite situation, when we are talking about open identification, where the identification of a non-registered person may occur, the system can return an empty array. [39]

2.3.2 Comparison Errors

The *similarity match score* is typically the result of matching of two fingerprints. The value of the similarity score is usually normalized into interval $[0, 1]$, which indicates the similarity between the presented set of features and the registered template. Closer to 1 the score is, the more certain system is that the present set of features comes from the same fingerprint as the registered template. Then the *threshold* value t is set for decision making if features are coming from the same fingerprint. Which means that if the value of the score is lower than t, non-match result is decided (Features on the input are coming from different fingerprint than the compared template). Otherwise, the match is made true. [49]

For matching one to one fingerprint (matching features from a fingerprint on an input against the features from a fingerprint on the output) is the output value of the matching system true for matched or false for non-matched. Two errors of this system that works on

matching one to one fingerprint can be made: 1)the decision about the true match of two sets of features that are coming from different fingerprints. This error is called *false-match*, 2) the decision about the false match of two sets of features that are coming from the same fingerprint. This error is called *false non-match*. [49] [39]

It is important to understand the difference between errors false match, false non-match, and more likely used *false acceptance* and *false rejection*. False match or false non-match are used in matching mode one to one while false acceptance or false rejection are errors used in processes of identification and their real meaning actually depends on the type of identity declaration made for the user. Like in the applications for positive identity statement (access control system) for example the false acceptance means false acceptance of impostor to the system and false rejection means rejection of the valid user registered into the system. On the other hand, in the application of negative identity claims (for example the application for preventing users from obtaining social benefits under a false identity), false acceptance means rejection of real request that should be served and false rejection means acceptance of impostor request. The application can of course use different metrics from a decision about acceptance/rejection but false acceptance (false acceptance rate **FAR**) and false rejection (false rejection rate **FRR**) have become popular metrics in the commercial sector. [53] [39]

Verification Error Rate

For verification, false match and false non-match rate, can be used. Now let's say that the set of features from the input will be marked as N (as not know) and the saved template saved in the database will be U (as a user). Then the null and alternative hypotheses can be defined as: [39] [35]

> $H_0 : N \neq U$ is the set of features from input that does **not** match the template from the database.
>
> $H_1 : N = U$ is the set of features from input that does match the template from the database.

So the decisions described before can be defined as: D_0 : non-match, D_1 : match. [39] [35]

Similarity measure $s(U, N)$ is included in the verification process when matching U and N. So D_0 is assigned as the result if the similarity score is lower then t and D_1 is assigned as the result otherwise. Now the two errors mentioned before can be defined as: [39]

> Error 1: false match (H_0 is true and D_1 is assigned as result)
>
> Error 2: false non-match (H_1 is true and D_0 is assigned as the result).

The probability of error 1 is called *false match rate* **FMR** and the probability of error 2 is called *false non-match rate* **FNMR**: [39]

$$FMR = P(D_1|H_0) \tag{2.1}$$

$$FNMR = P(D_0|H_1) \tag{2.2}$$

The power test $(1 - FNMR)$ is sometimes used. [39]

A large dataset of fingerprints is needed to evaluate the biometric system. After obtaining a large dataset, evaluation of the system is made by running the matching algorithm over the dataset. Then the score, called *genuine distribution*, is obtained by matching features from the same finger of the same person from the same hand with the probability $p(s|H_1)$. And the score called *impostor distribution* can be obtained as well by matching features from two fingers that are not the same (different person / different finger) by $p(s|H_0)$. The common graph as a result of computed and compared FMR and FNMR over threshold t is illustrated in the Figure 2.9. [39] [35]

$$\text{FNMR} = \int_0^t p\,(s \mid H_1)\,ds \tag{2.3}$$

$$\text{FMR} = \int_t^1 p\,(s \mid H_0)\,ds \tag{2.4}$$

As it can be seen in the graph or in the definition, the FMR and FNMR are functions of the value t and there should be a compromise made between these two. The system itself will not output always non-noisy or perfect images of fingerprint, therefore t should be lowered, but not too much because the lower the t is, the higher FMR is and more impostors are welcomed in our system. The system itself is not usually developed just for one purpose therefore graph of *Receiver Operating Characteristic* ROC or *Detection-Error Tradeoff* DET should be as an output of testing as well. The ROC or DET is not dependent on t which can be useful for the comparison of different matching systems. The ROC is used for plotting one function against the other, for example, FMR against (1-FNMR) can be plotted. A DET graph is a plot of error rates for binary classification systems (in our system 1 = match 0 = non-match). So with the DET graph, we can plot FMR against FNMR. [39]

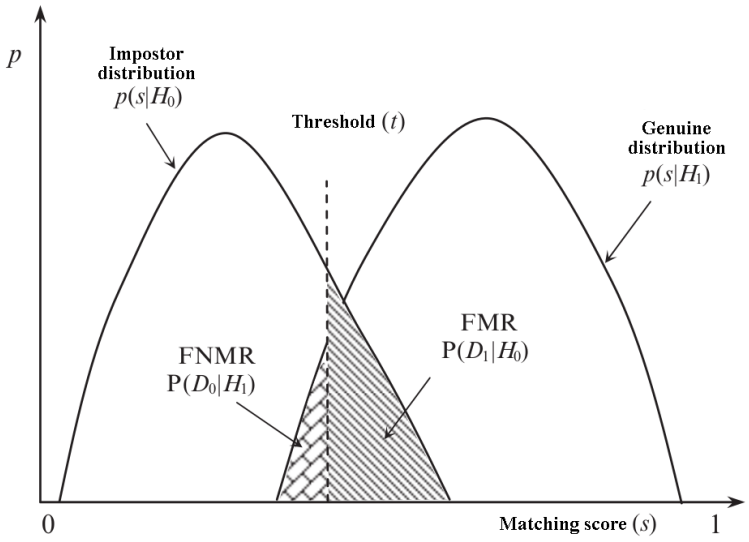

Figure 2.9: Graph FMR and FNMR. [39]

2.3.3 Attacks on Biometric Fingerprint Systems

Every biometric system has to consist of at least four modules which are a sensor, a feature extractor, a matcher, and a decision module. [28] The schema of basic system can be seen in the Figure 2.10.

- Sensors are described in Section 2.3.
- Feature extractor generates template based on extracted important data from raw data on input.
- Matcher is a module producing matching score, which stands for similarity of input biometric and biometric from the database generated by the matching algorithm.
- The decision module compares the matching score with the threshold and decides on either accepting or rejecting biometric on an input. [28]

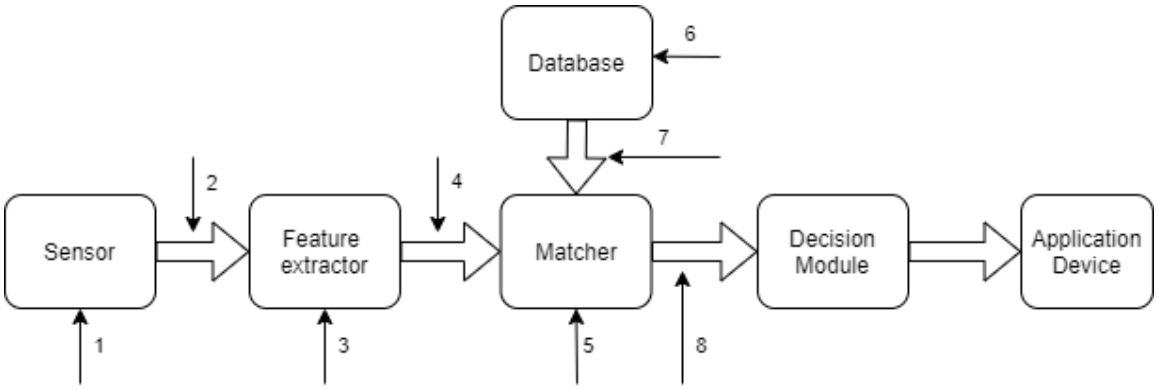

Figure 2.10: Possible attacks on the biometric system (numbered arrows shows points for possible attacks)

There exist attack points not only in every module of a biometric system but also in every connection. There are 8 attack points shown in Figure 2.10. The attacks are divided into two basic groups: direct and indirect attacks. [42] [31]

Direct attacks do not require any knowledge about any other part of the system than the sensor. Attack on the sensor can be done with any fake biometric (for example fake fingerprint made from gel, wax, ..) or just by damaging the sensor itself. [31]

Indirect attacks are the attacks where the impostor needs to know some information about the inner parts of the recognition system, which includes any of the parts (arrows with numbers) from Figure 2.10 except number 1, which is an attack on the sensor. [31] [18]

While the sensor is sending raw data through a communication channel to a feature extractor for pre-processing it is possible to steal the biometric (point 2). The stolen biometric is later replayed to a feature extractor to bypass the sensor. To attack the feature extractor (point 3) the impostor pushes on output feature values except for the original ones. Attack on the communication channel between feature extractor and matcher (point 4) is similar to the attack on point 2 with the exception of stolen values. [7] [31]

Attack on matcher module is simply pushing on the output the desired value which can be the high score to pass the threshold or lower the score for the opposite result as same

as just saying match or no match in cases where the decision is done already in a matching module. [7] [31]

Attack on the database (point 6) is as simple as pushing new or removing or modifying the existing templates in the existing database used by the biometric system. This attack is not as easy as it seems because saved templates are usually protected by steganography, watermarking and other methods, which require huge knowledge about the system. Another attack that can be done only while transmitting data from the database to the matcher is on point 7. This attack involves stealing, replacing, or altering biometric templates. [43] [31]

The last point where the impostor can be successful is on the communication channel between the matcher and the decision module (point 8). Although all the steps before were successful and well secured, the result can be still overridden, which causes the same result as attacking any point before. [31] [18]

2.4 Mathematical Morphology

In the last sixty years, there has been made a huge progress in the area of image processing. The innovation is mostly in the meaning of transforming an image into a better-represented form for further processing which can be image analysis or pattern recognition. The innovation never ends and very big attention is given to the field of mathematics morphology, because of its properties. The valued properties are quantitative descriptions of geometric structure and shape as well as a mathematical description of algebra, topology, probability, and integral geometry. [20] [52]

Mathematical morphology is having a huge impact on a lot of industries. Even in biology where it can be used for the extraction of forms or structures of animals or plants. Of course, it is used in the computer industry as well where it is mostly used in computer vision for the extraction of representative components that can describe the whole topology or shape. There is a lot of mathematics used in morphology, the analysis is based on set theory, topology, lattice algebra, functions, and many more. [20] [52]

An image is basically a multidimensional signal. The morphology was designed as complex as it can be. The morphological operations are generally designed, that they can be used for analyzing any signal or even multidimensional geometric pattern. [20] [52]

2.4.1 Binary Morphology

As stated before the mathematical morphology helps to obtain or even obtains by itself some information. There are three described transformations: unary (one image on the input and one image on the output), dyadic (merging result to one image from two input images), and information extraction (one image on the input and information on the output). The binary stands for the binary image which means that every pixel can contain only 2 values (in the binary 0 and 1). It is usually the rule that objects in front have the value 1 and objects in the back have the value 0. The advantage of binary images is that they store only pixels with the value 1 because the other pixels are 0. So we need to store only two values and that is the pixel value in its position (x, y). The advantage can be seen against the gray-scale image, where every element needs to store three values (x, y, z) where the z is the intensity of black color. On the other hand, gray-scale images store more information but in the extracted ridges this information is useless. [52]

2.4.2 Binary Dilation

Minkowsky came first with the idea of dilation and it was the reason why it is sometimes called the Minkowsky complement. At the beginning, we have our binary image on the input which is a set of positions. Then we combine our set with a template by use of the sum of the elements of the set. The definition is: Let K and L be two sets in E^N with elements k and l in this order, where $k = (k_1, k_2, ..., k_N)$ and $l = (l_1, l_2, ..., l_N)$ stands for N-pairs positions of elements. Then the set of all possible vector sums of pairs of elements is the binary dilation K by L, there one vector comes from K and the other one from L. As described, binary dilation works with binary images therefore the name. [52] [19]

Let $K \subset E^N$ and $L \in E^N$. The shift K by L, marked as $(K)_l$ is defined as: [52] [19]

$$(K)_l = \left\{ m \in E^N \mid m = k + l \quad \text{for some } k \in K \right\} \tag{2.5}$$

Let $K, L \subset E^N$. Binary dilation K o B, marked as $K \oplus_b L$, is defined as: [52] [19]

$$K \oplus_l L = \left\{ m \in E^N \mid m = k + l \quad \text{for any } k \in K \quad \text{k } l \in L \right\} \tag{2.6}$$

The set K and set L have the symmetrical role. For a better understanding of the dilation here is the definition of local dilation (one step): Dilation $K \oplus_l L$ is the place of all centers m, in the way, that shifts $(L)_m$ which is placed at the beginning L in m, hits the set K.[52] [19]

Properties of the dilation: [52]

$$\text{If K contains beginning:} 0 \in L, \text{then} K \oplus L \supseteq K \tag{2.7}$$
$$\text{K} \oplus_l L = L \oplus_l K \tag{2.8}$$
$$(K \oplus_l L) \oplus_l M = K \oplus_l (L \oplus_l M) \tag{2.9}$$
$$(K)_x \oplus_l L = (K \oplus_l L)_x \tag{2.10}$$
$$(K)_x \oplus_l (L)_{-x} = K \oplus_l L \tag{2.11}$$
$$\text{If } K \subseteq L, \text{ then } K \oplus_l M \subseteq L \oplus_l M \tag{2.12}$$
$$(K \cap L) \oplus_l M \subseteq (K \oplus_l M) \cap (L \oplus_l M) \tag{2.13}$$
$$(K \cup L) \oplus_l M = (K \oplus_l M) \cup (L \oplus_l M) \tag{2.14}$$

2.4.3 Binary Erosion

Erosion is based on the same principle one input binary image and one binary mask or core. Erosion is an opposite operation to dilation where it combines two sets using vector subtraction of set elements. In a simpler way, let K and L be sets in E^N with elements k and l. The erosion K by L is a set of all the elements x for which applies $x + k \in K$ and for all $l \in L$. [52] [14]

The definition of binary erosion K by L which is denoted as $A \ominus_b B$ is: [51]

$$K \ominus_l L = \{x \in E^N | x + l \in K \text{ for each } l \in L\} \tag{2.15}$$

Another interpretation of binary erosion $K \ominus_l L$ can be a place of all centers s so that the shift $(L)_m$ is completely included in the set K. The definition in this case is: [51] [14]

$$K \ominus_l L = \{m \in E^N | L_m \subseteq K\} \tag{2.16}$$

Properties of the binary erosion: [52]

$$\text{If L contains beginning: } 0 \in L, then K \ominus_l L \subseteq K \tag{2.17}$$
$$(K \ominus_l L) \ominus_l M = K \ominus_l (L \oplus_l M) \tag{2.18}$$
$$K \oplus_l (L \ominus_l M) \subseteq (K \oplus_l L) \ominus_l M \tag{2.19}$$
$$K_x \ominus_l L = (K \ominus_l L)_x \tag{2.20}$$
$$\text{If } K \subseteq L, \text{ then } K \ominus_l M \subseteq L \ominus_l M \tag{2.21}$$
$$(K \cap L) \ominus_l M = (K \ominus_l M) \cap (L \ominus_l M) \tag{2.22}$$
$$(K \cup L) \ominus_l M \supseteq (K \ominus_l M) \cup (L \ominus_l M) \tag{2.23}$$
$$K \ominus_l (L \cup M) = (K \ominus_l L) \cap (K \ominus_l M) \tag{2.24}$$
$$K \ominus_l (L \cap M) \supseteq (K \ominus_l L) \cup (K \ominus_l M) \tag{2.25}$$

The algebraic basis for mathematic morphology was introduced by Heijmans and Ronse [21]. As you could guess, dilation is not always reversible by erosion and the other way around. $M = K \ominus_l L$ can be as example, when if erosion is used on both sides by L has as result $M \ominus_l L = (K \oplus_l L) \ominus_l L \neq K$. So in the end the relation of closure $(K \oplus L) \ominus L \supseteq K$ is a valid relation in this case. One interesting thing is that when the dilation and erosion are used clearly in binary image K in a structured element L, the index l is omitted as can be seen in this relation. One example of binary erosion and dilation is shown in the Figure2.11. [14]

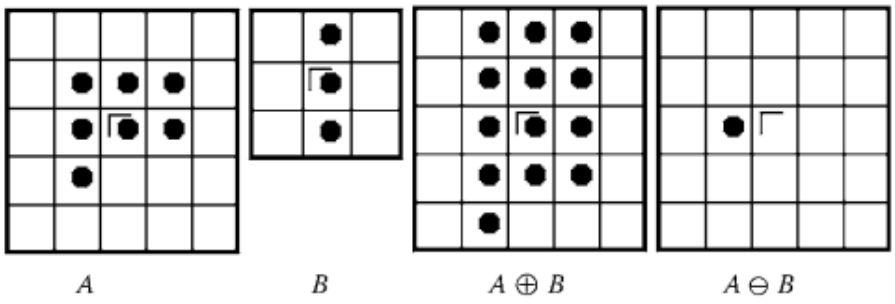

Figure 2.11: Binary dilation and erosion [52]

2.4.4 Thinning

Thinning is a special kind of erosion. The main difference is in the factors, which are: 1) the component in an image on the input can not disappear. 2) the output of thinning are 1 pixel wide objects, which means that the 1px wide line is in the same distance from all the original edges (in the center). The structural shape of the object represented as a graph is the output of thinning. That is why it is used as a basic step in image processing in many industries such as control of industrial parts or in our fingerprint detection. [52]

The main use of the result of thinning which is the skeleton of an object, is the representation and classification of objects in a binary image. The first form of thinning was the definition of digital skeletons and algorithms based on set transformation and it was introduced by Jang and Chin [32]. Another view on the thinning operation was introduced by Serra and Meyer and their approach was based on the operation „hit-or-miss“ and in

this approach the definition is: [50]

$$K \wedge L = K - (K \text{ s}^* L) = K \cap (K \text{ s}^* L)^m \tag{2.26}$$

The definition shows that thinning is basically nothing else than finding and deleting. All the presence of L in K are localized and subtracted by the operation Ks^*L. All the elements that were found in K are deleted. To complete the whole process of thinning, the sequence of structural elements is used: $\{K\} = \{L^1, L^2, \ldots, L^8\}$, where K^i is turned version of K^{i-1}. When used on digital images and this implementation of sequence of 8 elements is used: [50]

$$L^1 = \begin{bmatrix} 0 & 0 & 0 \\ \times & 1 & \times \\ 1 & 1 & 1 \end{bmatrix}, \quad L^2 = \begin{bmatrix} \times & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & \times \end{bmatrix}, \quad L^3 = \begin{bmatrix} 1 & \times & 0 \\ 1 & 1 & 0 \\ 1 & \times & 0 \end{bmatrix}, \quad L^4 = \begin{bmatrix} 1 & 1 & \times \\ 1 & 1 & 0 \\ \times & 0 & 0 \end{bmatrix},$$

$$L^5 = \begin{bmatrix} 1 & 1 & 1 \\ \times & 1 & \times \\ 0 & 0 & 0 \end{bmatrix}, \quad L^6 = \begin{bmatrix} \times & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & \times \end{bmatrix}, \quad L^7 = \begin{bmatrix} 0 & \times & 1 \\ 0 & 1 & 1 \\ 0 & \times & 1 \end{bmatrix}, \quad L^8 = \begin{bmatrix} 0 & 0 & \times \\ 0 & 1 & 1 \\ \times & 1 & 1 \end{bmatrix}$$

Then the thinning by eight elements is defined as: [52] [50]

$$K \wedge \{L\} = \left(\cdots \left(\left(K \wedge L^1\right) \wedge L^2\right) \cdots\right) \wedge L^8 \tag{2.27}$$

In the process of thinning, erase of the edges must be achieved under the condition that the connectivity can not be changed. All eight cores are designed the way that they are together to meet the conditions. The process of using all eight cores is repeated until there is no change from the last step. The result of thinning is shown in the Figure 2.12 [52]

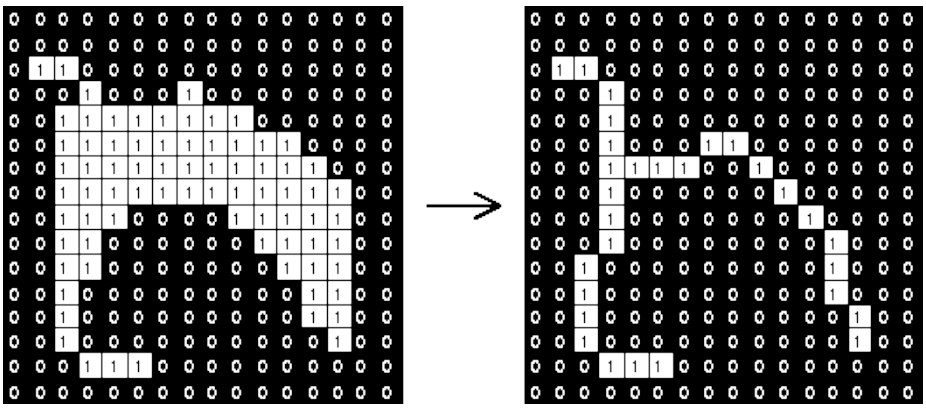

Figure 2.12: Thinning [5]

2.5 Digital Image Processing

Image processing is used to extract some useful information or to get an enhanced image from the image on the input. The use of image processing has been increasing exponentially in the last decades and it is being used from entertainment to medicine. [12]

Digital image processing is a huge field and it consists of encompassing digital signal processing techniques and specific techniques on image. Image can be described as function

f(x,y), where x and y are continuous variables. It has to be sampled and transformed into a matrix of numbers for digital processing. So the digital image processing stands for manipulation with these finite numbers. There are several types of image processing: [12]

- **image enhancement:** Image is manipulated mostly by heuristic techniques and it is easier to extract the information from the result,
- **image restoration:** Processes the damaged images to find statistical or mathematical descriptions of degradation in order to revert the degradation,
- **image analysis:** processing of an image for automatic extraction of information. (examples are image segmentation or edge extraction),
- **image compression:** compression is applied on digital images with a goal to reduce their cost for transmission and storage.

2.5.1 Thresholding

The most common and basic image processing technique to obtain binary or multilevel images from gray-scale images is thresholding. The basic principle of thresholding methods is to replace each pixel in an input image with a black pixel if the image intensity is less than the global threshold value or with a white pixel if the image intensity is higher than the threshold value. There is also the locally adaptive threshold. Both of these approaches come with disadvantages. The global thresholding does not work with the local information and local thresholding does not work with the global information. [46] Therefore the methods like adaptive thresholding exists. [48] [11] This thesis is works only with global thresholding.

2.5.2 Histogram Equalization

Histogram equalization is a technique for adjusting image intensities to enhance contrast. The histogram is a graphical representation of the intensity distribution of an image. Enhancing image contrast is accomplished by spreading out the most frequent intensity values. The global contrast is usually increased by this technique where the data on the input were represented with very close contrast. [55]

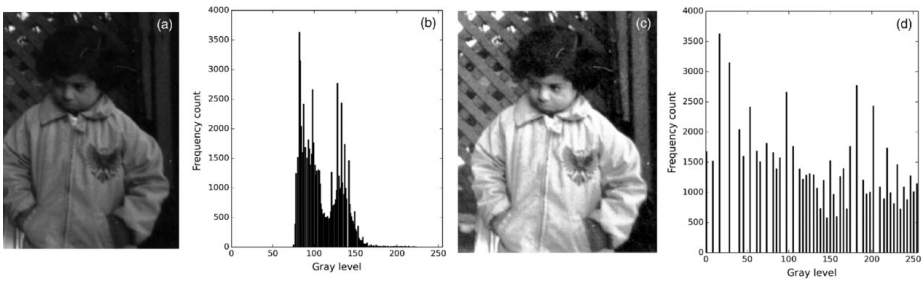

Figure 2.13: Histogram equalization: a) input image, b) input image histogram, c) input image after histogram equalization d) input image histogram after equalization [55]

Contrastive Limited Adaptive Equalization (CLAHE)

The technique of histogram equalization has problems with over-amplification of noise. Therefore the separate histograms are calculated in adaptive histogram equalization (AHE).

AHE can achieve higher contrast enhancement. But because some homogeneous regions may be oversaturated, it amplifies the noise. Therefore the amplification is limited by clipping the histogram at a predefined value before computing cumulative distribution function in the CLAHE algorithm. [55]

2.5.3 Morphing

Morphing itself is the process of transformation between states of appearance. Basic operations used in morphing are translations rotations but it gets to harder operations like shapeshifting. [17] Image morphing has been shown as an extremely powerful tool not only for entertainment purposes. Morphing is mostly known as a technique that uses coupling image warping with color interpolation. Image warping is used for geometric transformation in 2D for maintaining feature alignment between features like the position of the nose and eyes. And the color interpolation is used for blending colors from input images. [36] [58]

The first step of morphing is finding the location of features from the input images. Then the primitives like mesh nodes, curves, or points are made correspondence between them is established. The primitives are established like that each is specifying one image feature or landmark. The second step is to compute the mapping function defying the special relationships between all points from both images. Mapping functions are used for interpolating positions while morphing two images together. After interpolation of positions, the color interpolation is run as the last step of morphing. [58]

In the past few years, big data and deep learning have become a trend, which can be seen even in morphing. For example, the generative adversarial network works for morphing well. The network is trained to synthesize frames by a given data-set of samples along the whole process of transformation. [17]

Chapter 3

The Design of Solution

This Chapter describes the techniques needed to create morphed fingerprints. For certain parts, such as thinning, it is necessary to know the theory described in the Chapter 2.

3.1 Fingerprint Image Preprocessing

The first step is to convert the fingerprint image from input to a gray-scale image. The papillary lines are thus marked in a darker gray color in the Figure and the valley in white or in a lighter shades of gray. Before any calculations, the image is equalized by CLAHE. First, so that we do not work on unnecessarily large fingerprint images, the detection of fingerprint boundary points (leftmost point, lowest point, ..) and subsequent trimming of the excess background is performed. After trimming, the smaller fingerprint is resized to the size of the bigger fingerprint. For subsequent processing, a fingerprint mask is calculated in the same step. For the most realistic results of morphing, it is necessary to ensure the highest possible similarity between fingerprints before its implementation. Therefore the color normalization is performed using the minimax algorithm where there is a linear mapping to the interval between min and max.

3.2 Fingerprints Alignment

To find the best position to align two fingerprints on top of each other, a technique based on oriented fields is used. The use of kernel or delta alignment has also been considered but some fingerprints are very different and a similar implementation would then not work properly.

3.2.1 Algorithm for Fingerprint Orientation Field

Let $[x, y]$ be a pixel in the fingerprint image. Then the angle $\theta_{x,y}$ is a local orientation of all the ridges in pixel $[x, y]$ that are coming through the arbitrarily large surroundings centered in pixel $[x, y]$. $\theta_{x,y}$ is in the range $[0..\pi]$ just because the ridges have no direction. Then the fingerprint orientation field is matrix D, in which the elements store the local orientation of the ridges in the given surroundings. Each element $\theta_{x,y}$ of the matrix D indicates the average orientation of the lines around the pixel $[x, y]$, where the pixel $[x, y]$ represents the square grid located above this point. The result fingerprint orientation field can be seen in the image 3.1. Each element $\theta_{x,y}$ is often associated with a value of $r_{x,y}$, which indicates the degree of reliability of the orientation. [37]

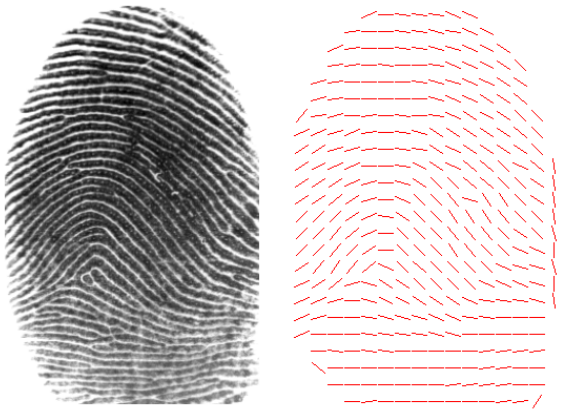

Figure 3.1: Input fingerprint image on the left side, fingerprint orientation field on the right side

Description of the algorithm:

1. In the beginning, the normalized image $\mathcal{G}$ needs to be divided into w*w blocks. Then there is no ridge in the block, value 0 is assigned and r is set to 0 as well otherwise the r is set to 1 and the calculation can begin. [37] [57]

2. For each pixel (x,y) the gradients $\partial_i(x,y)$ and $\partial_j(x,y)$ need to be computed by the Sobel operator.[37]

3. The local orientation is set for each block with center in pixel (x, y), by these equations: [37] [57]

$$\mathcal{V}_i(x,y) = \sum_{u=x-\frac{w}{2}}^{x+\frac{w}{2}} \sum_{v=y-\frac{w}{2}}^{y+\frac{w}{2}} 2\partial_i(u,v)\partial_j(u,v) \tag{3.1}$$

$$\mathcal{V}_j(x,y) = \sum_{u=x-\frac{w}{2}}^{x+\frac{w}{2}} \sum_{v=y-\frac{w}{2}}^{y+\frac{w}{2}} \left(\partial_i^2(u,v) - \partial_j^2(u,v)\right) \tag{3.2}$$

$$\theta(x,y) = \frac{1}{2}\tan^{-1}\left(\frac{\mathcal{V}_j(x,y)}{\mathcal{V}_j(x,y)}\right) \tag{3.3}$$

   Where $\theta(x,y)$ is the smallest square estimation of the local orientation of the ridges in the block centered at (x, y). [57]

4. To remove or at least to reduce errors that cause noise or other, the low pass filter can be used. It can be used because of the low variability of ridges in the near surroundings except around the singular points. Before filtration, every $\theta(x,y)$ needs to be converted into the vector field. Equation of conversion to vector field: [37] [57]

$$\Phi_i(x,y) = \cos(2\theta(x,y)) \tag{3.4}$$

$$\Phi_j(x,y) = \sin(2\theta(x,y)) \tag{3.5}$$

The filtration with low pass filter: [37]

$$\Phi'_i(x,y) = \sum_{u=-w_\Phi/2}^{w_\Phi/2} \sum_{v=-w_\Phi/2}^{w_\Phi/2} W(u,v)\Phi_i(x-uw, y-vw) \tag{3.6}$$

$$\Phi'_y(x,y) = \sum_{u=-w_\Phi/2}^{w_\Phi/2} \sum_{v=-w_\Phi/2}^{w_\Phi/2} W(u,v)\Phi_j(x-uw, y-vw) \tag{3.7}$$

Where the W is a low pass filter. In default, 5x5 is the size of the filter. [57]

5. The computation of the smoothed local orientation is then performed using the following equation: [37] [57]

$$\mathcal{O}(x,y) = \frac{1}{2}\tan\left(\frac{\Phi'_j(x,y)}{\Phi'_i(x,y)}\right) \tag{3.8}$$

3.2.2 Algorithm for Alignment Using Fingerprint Orientation Field

After computing the oriented field for both fingerprints, it is finally possible to start aligning them. The method tries to maximize the similarity of oriented fields at their intersections.

The oriented fields $\mathcal{O}_1$ and $\mathcal{O}_2$ are calculated in blocks $w \times w$ and at each position (x, y) it contains the value of the size of the angle θ and r = [0,1]. Then the similarity between oriented arrays is calculated: [16]

$$S\left(\mathcal{O}^1, \mathcal{O}^2\right) = \frac{\sum_{(x,y)\in(V_{\mathcal{O}1}\cap V_{\mathcal{O}2})}\left(r^y_{x,y} + r^2_{x,y}\right)\cdot\psi\left(\theta^1_{x,y}, \theta^2_{x,y}\right)}{\sum_{(x,y)\in\left(V_{\mathcal{O}^1}\cap V_{\mathcal{O}^2}\right)}\left(r^1_{x,y} + r^2_{x,y}\right)} \tag{3.9}$$

, where $\psi\left(\theta_1, \theta_2\right)$ is the similarity of orientation fields:

$$\psi\left(\theta_1, \theta_2\right) = 1 - \frac{2\cdot|\theta_1 - \theta_2|}{\pi} \tag{3.10}$$

and positions of the front are included in $V_{\mathcal{O}}$: [16]

$$V_{\mathcal{O}} = \{(x,y)|o_{x,y} \in \mathcal{O} \wedge r_{x,y} > 0\} \tag{3.11}$$

The alignment is computed for every position and angle. The positions are skipped for positions of oriented fields on which they do not overlap sufficiently (at least 25 % overlap). [16]

$$\frac{|V_{\mathcal{O}^1} \cap V_{\mathcal{O}}|}{\max\left(|V_{\mathcal{O}^1}|, |V_{\mathcal{O}^2}|\right)} \geq min_{VR} \tag{3.12}$$

In this implementation, the only considered rotation is in the range $< -\pi/2; \pi/2 >$ and a shift of a maximum of 50 % of the width when moving right or left and 50 % of the height when moving up or down. Larger changes either led to inaccuracies or did not affect the result. After aligning the fingerprints, parts that do not intersect are trimmed. An example of the result is the Figure 3.2.

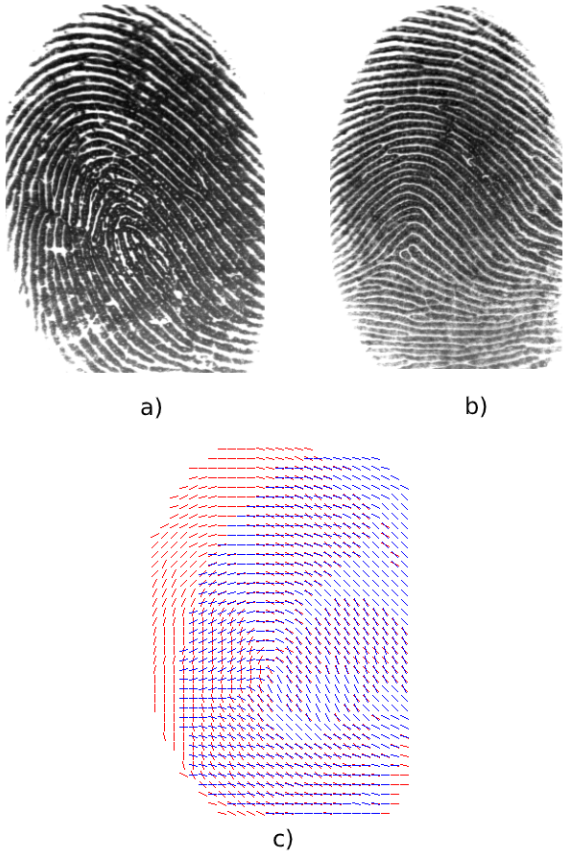

Figure 3.2: a) first fingerprint, b) second fingerprint c) aligned fingerprint orientation fields

3.3 Ridge Frequency Image

It is possible to model levels of gray as sine waves in the near neighborhood of the fingerprint. This can be applied only in cases where no singular point is present in the examined neighborhood. For these cases, the frequency from near neighborhoods will be used. The definition and equations: Let's have $\mathcal{G}$ denoting the normalized fingerprint image and $\mathcal{O}$ the oriented field calculated as described in the previous Section. Then: [47]

1. The fingerprint image ($\mathcal{G}$) is divided into blocks with size $w \times w$ usually 16×16. The block size has to be the same as block size for the fingerprint orientation field($\mathcal{O}$). [26]

2. Compute an *oriented window* (something else than 3.2.1) with size $l \times w$. Usual size is 32×16 for each block centered at (x, y). [26]

3. Compute the x-signature $X[0], X[1], ...X[l-1]$ of ridges inside oriented window for every block centered in (x, y), where:

$$X[k] = \frac{l}{w} \sum_{d=0}^{w-1} \mathcal{G}(u, v), \quad k = 0, 1, \ldots, l-1 \tag{3.13}$$

$$u = x + \left(d - \frac{w}{2}\right) \cos \mathcal{O}(x, y) + \left(k - \frac{l}{2}\right) \sin \mathcal{O}(x, y) \tag{3.14}$$

$$v = y + \left(d - \frac{w}{2}\right) \sin \mathcal{O}(x, y) + \left(\frac{l}{2} - k\right) \cos \mathcal{O}(x, y) \tag{3.15}$$

In the case where no singularity or minutiae is present in the oriented window, x-signature creates a sin curve and frequency can be counted from it. Let the average number of pixels between two ridges be $\mathcal{T}(x, y)$. Then the equation for frequency $\Omega(x, y)$ is: $\Omega(i, j) = 1/\mathcal{T}(i, j)$. Value -2 is assigned if no ridge is found in the oriented window, so it is marked as non-valid. [26]

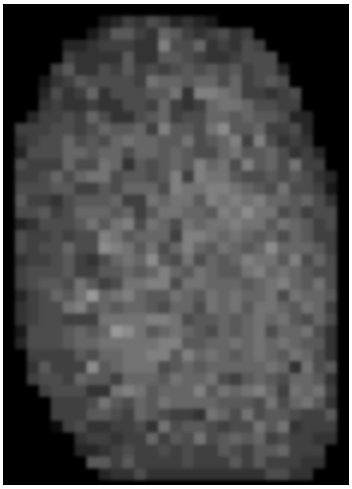

Figure 3.3: Ridge frequency image of fingerprint from image 2.7

3.4 Minutiae

The application uses an algorithm that works on the extraction of minutiae with the use of thinning. To extract the minutiae, it is necessary to first calculate the oriented field described above. Then to distinguish the background and the fingerprint. An important step is the extraction of ridges and that creates a binary image, where the ridges carry the value 1. The last step of line extraction is thinning, see 2.4.4.

Subsequent extraction of the minutiae is performed by using the crossing number method on the neighborhood of the point P at a distance of 1. Or on its 8 surrounding pixels: [34]

$$CN = 0.5 \sum_{i=1}^{8} |P_i - P_{i+1}|, \quad P_9 = P_1 \tag{3.16}$$

.

Subsequently, false markers will be removed in post-processing. The first part is ridge counting. Ridge counting determines how many lines pass through or touch the line that connects the core and delta. For this operation, it is necessary to determine the delta and the core of the fingerprint as accurately as possible. Then: [34]

- If ridge bifurcation lays on the ridge, then two ridges are counted
- If the ridge is touching the island or dot, then one ridge is counted.
- The valley must be interrupted by a line between the delta and the core, otherwise, one line is subtracted. (Applies to loop lines only.)

In the end, the minutiae validity needs to be checked. The check is made by distance measurement. Firstly the average count of pixels between 2 ridges is counted (ridge counting distance = RCG). Secondly, the distance between delta and core (DC) is counted. Then the distance of minutiae from the next minutiae has to be larger than RCG/DC. [34]

3.5 The Cutline Estimation

This is the line around which most minutiae occur and when generating a new fingerprint, it ensures that there will be a sufficient number of minutiae from both fingerprints in the newly generated fingerprint. The straight line is calculated after performing all previously described operations, so it is computed from aligned and truncated impressions from parts of the non-intersection and with their oriented fields. Minutiae extracted from these prints are denoted as T^1 for fingerprint 1 and T^2 for fingerprint 2. [16]

The $\rho = (\rho_x, \rho_y)$ is assigned as a barycenter. The article does not say what exactly it is, so the core was used or delta if core is not present. [16]

Then the cutline l has these parameters: [16]

$$\begin{aligned} &a_l \cdot x + b_l \cdot y + c_l = 0 \\ &a_l = \sin(\beta), \quad b_l = \cos(\beta) \\ &c_l = -\rho_x \cdot \sin(\beta) - \rho_y \cdot \cos(\beta) \end{aligned} \tag{3.17}$$

Now the line is rotated around the barycenter by a predetermined step in the range $(0°, 180°)$. A straight line score $S_(c)$ is specified for each β angle: [16]

$$S_c = \omega_o \cdot S_o + \omega_v \cdot S_v + \omega_m \cdot S_m \tag{3.18}$$

Where S_o is the similarity of the oriented fields, S_v is the similarity of the frequency characteristics, S_m is the score used to evaluate the generation of a fingerprint similar to both based on the found minutiae of fingerprint 1: T_1, and fingerprint 2: T_2. $\omega_o, \omega_v, \omega_m$ are then the weights of individual components in the range $< 0; 1 >$, $\omega_o + \omega_v + \omega_m = 1$. [16]

$$S_o = \frac{\sum_{(i,j)\in C} \left(r^1_{i,j} + r^2_{i,j} \right) \cdot \psi \left(\theta^1_{i,j}, \theta^2_{i,j} \right)}{\sum_{(i,j)\in C} \left(r^1_{i,j} + r^2_{i,j} \right)} \tag{3.19}$$

$$S_v = \frac{\sum_{(i,j)\in C} \left(1 - \frac{|v^1_{i,j} - v^2_{i,j}|}{(\max_F - \min_F)} \right)}{|C|} \tag{3.20}$$

Only the elements that are in the intersection of oriented fields and their distance is maximal d_{max} from actual cutline l is in the C. So: [16]

$$C = \{(i,j)|(i,j) \in \left(V_{\hat{O}^1} \cap V_{\hat{O}^2}\right) \wedge \\ \text{dist}_l(i,j) \leq d_{\max}\} \tag{3.21}$$

$$\text{dist}_l(x,y) = \frac{|a_l \cdot x + b_l \cdot y + c_l|}{\sqrt{a_l^2 + b_l^2}} \tag{3.22}$$

$$S_m = \max\left(\zeta_m\left(T^1, T^2\right), \zeta_m\left(T^2, T^1\right)\right) \tag{3.23}$$

$$\zeta_m(A,B) = \\ \frac{Z\left(|A|_l^P, \mu_m, \tau_m\right) + Z\left(|B|_l^N, \mu_m, \tau_m\right)}{2} \tag{3.24}$$

where the $|T|_l^P$ stands for minutiae which are over the cutline (from the first fingerprint) and $|T|_l^N$ stand for minutiae under the cutline. So:[16]

$$|T|_l^P = |\{m \in T|\phi_l\left(m_x, m_y\right) \geq 0\}| \\ |T|_l^N = |\{m \in T|\phi_l\left(m_x, m_y\right) < 0\}| \tag{3.25}$$

The result needs to be normalized into the range $< 0; 1 >$. Therefore the sigmoid function Z is and is controlled by the parameters μ a τ. [16]

$$Z(v, \mu, \tau) = \frac{1}{1 + e^{-\tau(v-\mu)}} \tag{3.26}$$

As the result cutline is assigned cutline with highest score S_c. [16]

3.6 Image-based Fingerprint Generation

The fingerprint is generated by merging fingerprint intersections. Let $\hat{F}^p$ be the part of the fingerprints above the cutline and $\hat{F}^n$ be the part of the fingerprints below the cutline, which we determine as follows: [16]

$$(p,n) = \begin{cases} (1,2) & \text{if } \zeta_m\left(T^1, T^2\right) \geq \zeta_m\left(T^2, T^1\right) \\ (2,1) & \text{otherwise} \end{cases} \tag{3.27}$$

Then the fingerprint generation looks like: [16]

$$D^I(x,y) = w_{x,y}^{l_{\max}} \cdot \hat{F}^p(x,y) + \left(1 - w_{x,y}^{l_{max}}\right) \cdot \hat{F}^n(x,y) \tag{3.28}$$

where $w_{x,y}^{l_{\max}}$ is weight to balance the mixing in near distance to the cutline. [16]

$$w_{x,y}^{l_{\max}} = \begin{cases} 1 - \max\left(0, \frac{d_{\max} - \text{dist}_{\max}(x,y)}{2 \cdot d_{\max}}\right) \\ \text{if } a_{\ln ax} \cdot x + b_{\ln ax} \cdot y + c_{\ln ax} \geq 0 \qquad s \\ \max\left(0, \frac{d_{\max} - \text{dist}_{\ln aax}(x,y)}{2 \cdot d_{\max}}\right) \text{ otherwise} \end{cases} \tag{3.29}$$

3.7 Improved Image Based Generation

After going through image-based generation from the previous Section 3.5 some ideas have been shown. Some improvements were made in steps after generation and mostly to make the generation more universal to different fingerprints. The first improvement was made on the computation of the mask for fingerprint. Since the given database contains fingerprints scanned with some noise, maximal movement in pixels had to be added to avoid containing it in the mask, where the optimal value was considered as 5px.

3.7.1 Mask Alignment

The first change in the algorithm was made in aligning the fingerprints. Since the alignment by orientation field sometimes cuts a lot from one of the fingerprints it was tried to minimalize this mistake by aligning the fingerprints with masks. This approach maximizes the space coverage of fingerprints.

Masks contain values 0 = background, 1 = foreground and they are the same size. Mask from 2nd fingerprint is being aligned on the mask from 1st fingerprint. The rotated masks in the range $< -\pi/2; \pi/2 >$ are computed and stored in an array. Each rotated mask is moved over the 1st fingerprint mask in the range $< -h\_w; h\_w >$ for width and $< -h\_h; h\_h >$ for height where h_w is half of the width and h_h is half of the height of the fingerprint mask.

Then the similarity value is computed as the sum of values of the intersection of the 1st fingerprint mask and the rotated mask. The rotated and moved mask position with the highest score is assigned as the optimal alignment.

3.7.2 Cutline

The main change that makes the best improvement is to rotate the cutline around the center of the first fingerprint. The option of rotating the cutline around the core or delta can lead to the maximalization of only one fingerprint on the whole image based on the position of the core or delta on the fingerprint. This can be seen in the example from picture 3.4.

Figure 3.4: Demonstration of fingerprint generation problem (in1, in2 = input fingerprints, out = morphed fingerprint). Only small part at the bottom is contained from fingerprint in1

The last change made was going over the result with the Gaussian filter with kernel 5x5 to blur the whole fingerprint a bit and make the transitions between fingerprints less visible. This option works properly only in some cases and is not included in the final testing.

3.8 Minutiae-based Fingerprint Generation

To complete all ideas about fingerprint image generation, the second fingerprint generation from paper [16] is mentioned as well. But because of different topics and the author's tests against VeriFinger which ended worse than image-based morphing, this solution was neither implemented nor tested.

A realistic fingerprint can be generated synthetically based on features with a combination of fingerprint orientation fields and frequencies. The first step is to obtain dual identity fingerprint information ($\tilde{O}$ orientation field, $\tilde{\Upsilon}$ frequency field and $\tilde{T}$ markers) by combining the first and the second fingerprints similar to the method 3.6. [16]

$$\tilde{O}(x,y) = w_{x,y}^{l_{\max}} \cdot \hat{O}^p(x,y) + \left(1 - w_{x,y}^{l_{\max}}\right) \cdot \hat{O}^n(x,y) \tag{3.30}$$

$$\tilde{\Upsilon}(x,y) = w_{x,y}^{l_{\max}} \cdot \Upsilon^p(x,y) + \left(1 - w_{x,y}^{l_{\max}}\right) \cdot \Upsilon^n(x,y) \tag{3.31}$$

$$\tilde{T} = \{m \in T^p, \phi_{l_{\max}}(m_x, m_y) \geq 0\} \cup \{m \in T^n, \phi_{l_{\max}}(m_x, m_y) < 0\} \tag{3.32}$$

The generation itself is based on two steps:

1. In the first step, we work only with minutiae and fingerprint orientation field from the input which carry information about the position x_i, y_i and the type of marker. We recognize only two types of minutiae and that is the bifurcation and the ridge ending. Then the minutiae are placed in position according to the position and turned by an angle according to the field of orientations. [10]

2. In the second step, the papillary lines are "grown" from the markers using a Gabor filter modified to work with the frequency field v and the fingerprint orientation field phi_{xy}: [10]

$$\text{gabor}(r, s : \phi_{xy}, \nu) = e^{-\frac{(r+s)^2}{2\sigma^2}} \cdot \cos\left[2\pi\nu\left(r\sin\phi_{xy} + s\cos\phi_{xy}\right)\right] \tag{3.33}$$

In the end, it is possible to crop the fingerprint to the shape of the fingerprint and add a little noise to make it "real" for the human eye as well. Trimming will be performed according to the mask calculated in the Section 3.1.

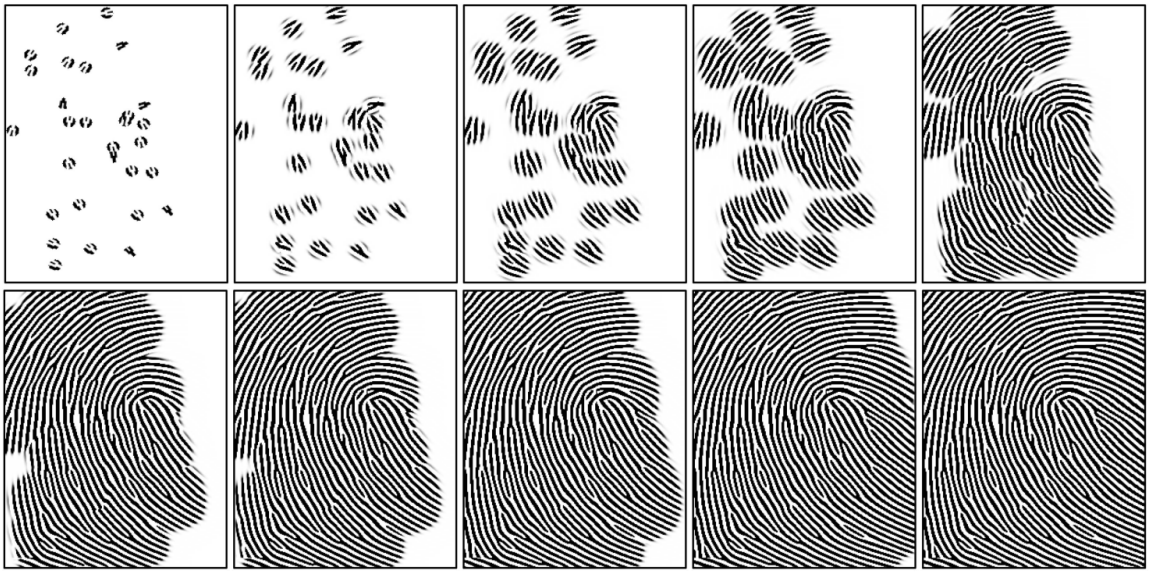

Figure 3.5: Demonstration of fingerprint generation [10]

Chapter 4

Implementation

This Chapter describes the implementation of both morphing applications and fingerprint matching applications with the use of tools from Innovatrics [4] and Neurotechnology [1]. A laptop with the following configuration was used for testing:

- CPU: Intel Core i5-6200U
- RAM: 8GB DDR4
- SSD: Samsung SSD 860 EVO, M.2 - 500GB
- OS: Windows 10

4.1 Morphing Application

The application for morphing was implemented in Python version 3.8.5. as a console application with the use of the following libraries: NumPy, SciPy, PIL, Matplotlib, OpenCV, and a GitHub repository [3], which was used for minutiae detection and thinning. The application generates a morphed fingerprint from two fingerprints on the input based on an algorithm described in Chapter 3.

The application runs in two modes. The first is used to generate one morphed result, the other is to generate all the results from folders on the input. Both need to set `--blocksize` or optionally `--center`, `--plot` or `--save`. The first mode then needs to set the input images (`--image_1` and `--image_2`). The other needs to set the input folders (`--filder1`, `--folder2` and `--folder3`) and set the suffix of input images (`--suf`).

Application Control

```
python morph.py [-h] [--image_1 /path/to/first/fingerprint/image/]
[--image_2 /path/to/second/fingerprint/image/] --blocksize int [--save
filename] [--folder1 folder] [--folder2 folder] [--folder3 folder] [--plot]
[--suf .bmp/ .tif /...]  [--center]
```

- `-h`: Show help
- `--image_1 /path/to/first/fingerprint/image/`: the first input image for morphing. Only for one generation.

- `--image_2 /path/to/second/fingerprint/image/`: the second input image for morphing. Only for one generation.
- `--blocksize int`: Integer value for blocksize
- `--save filename`: Save to a file "filename". If not set nothing is saved.
- `--folder1 folder`: Folder with the first input images. Argument used only for generating over the whole folder.
- `--folder2 folder`: Folder with the second input images. Argument used only for generating over the whole folder.
- `--folder3 folder`: Folder for output images. Argument used only for generating over the whole folder.
- `--plot`: Plots few steps with Matplotlib. Steps: Input images without background, orientation fields, alligned fingerprints, frequency image and minutiae, optimal cutline with barycenter and result. Demonstration of output can be seen in Figure 4.1
- `--suf .bmp/.tif /...`: Suffix of files in the folders that should be taken as an input.
- `--center`: Take center of fingerprints as a barycenter for cutline.
- `--mask`: Align fingerprints by most coverage of masks.
- `--eq`: Use CLAHE on result.
- `--gaus`: Use gaussian blur on result.

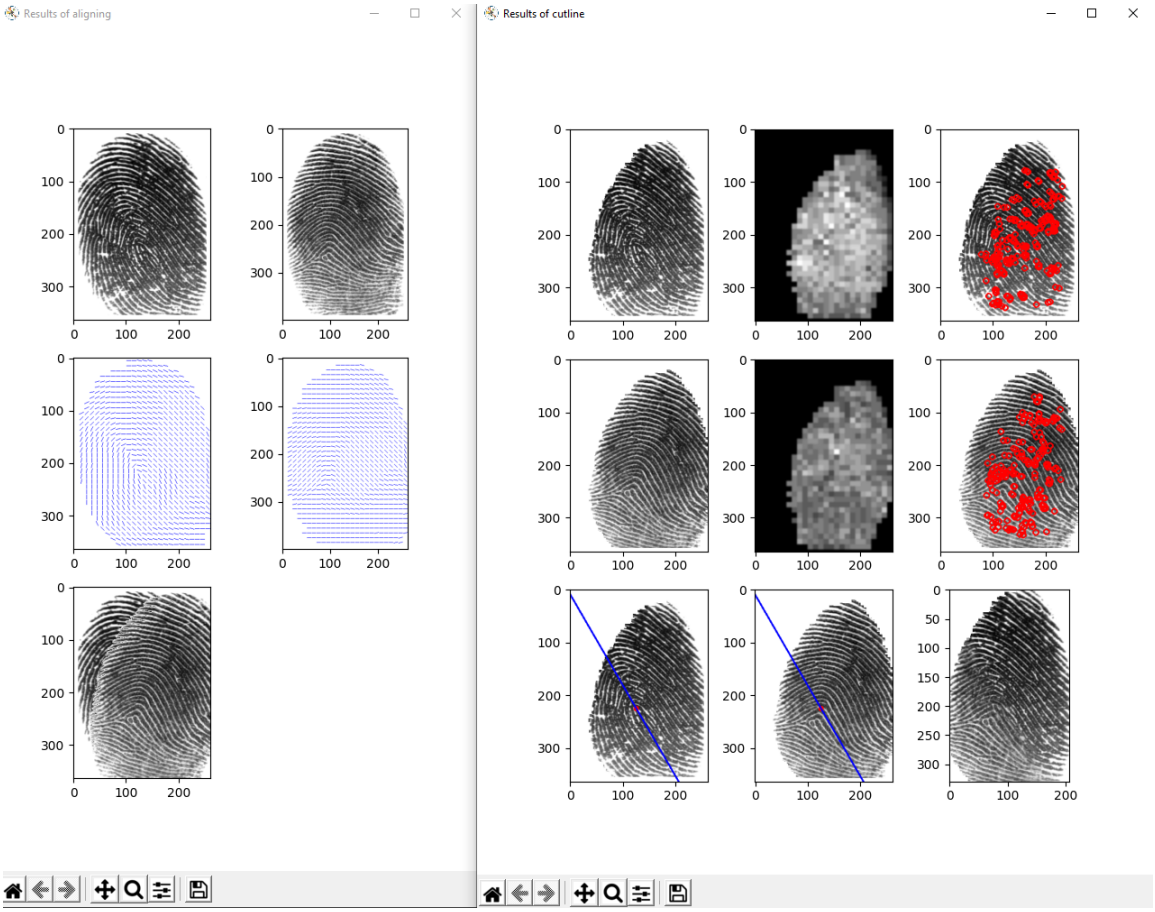

Figure 4.1: Demonstration of plotting the output with steps from morphing application (left window: 1st line = fingerprints with cropped background, 2nd line = orientation fields, 3rd line aligned fingerprints; right window: 1st and 2nd line = fingerprint with cropped overlaying part, frequency image, minutiae; 3rd line= cutline on 1st and 2nd fingerprint, last is result)

4.2 Innovatrics IDKit PC SDK v8.0.1.0

IDKit [4] is a tool for creating applications used for fingerprint identification. The IDKit PC SDK supports a number of image formats (RAW, PNG, BMP, JPEG, JPEG 2000, GIF, WSQ, TIF) and is compatible with a wide range of scanners. The tool has support for working with the database, which can be in SQLite / MySQL / MSSQL / Oracle formats. [27]

Innovatrics IDKit PC SDK is officially supported for Windows 32-bit and 64-bit for version 7 and higher, Linux 32-bit and 64-bit (Red Hat), Centos version 7 and higher. [27]. The application was implemented for Windows 10 and the source code is also submitted with the project, which can be run in Visual Studio 2019.

4.2.1 Application for Comparing Fingerprints

The application is implemented in C++ using the Innovatrics IDKit PC SDK. Visual Studio 2019 was used for development. It is possible to start evaluation over only three

inputs (fingerprint 1, fingerprint 2, morphed fingerprint from the two previous ones) when the morphed fingerprint is evaluated against two inputs. But also evaluation over the whole folder is possible. Then the output is a .csv file that contains the similarity score accumulator, from which then it is possible to create an FMR curve. The lowest size for image on input is 90px width and 90px height.

The outputs of the application are scores for comparison over just three fingerprints (just one run) for both fingerprints from input in comparison with the morphed one. For the whole folder, only the lower score is stored in the .csv file. To load the file into excel, just simply load data from .csv and set the delimiter as coma, if it is not set automatically.

To run the application, it is first necessary to generate files for the Visual studio using CMake in the build folder. Then, in the project, set the language to C++ 17 (project entry → Properties → C / C ++ → Language → C ++ Language Standard). Finally, set compare as the running project. The source files are located: `evaluation_modules/Compare_fingerprint/src`, the project for Visual studio here:`evaluation_modules/Compare_fingerprint/build/` compare and executable in the Debug pad.

Application Control

`./compare.exe -db _ -s -f1 _ -f2 _ -i1 _ -i2 _ -gen _ -geni _ -suf _`

- `-db "databaseString"`: The default value is iengine.db.
- `-s`: Save the result to the database.
- `-f1 "finger.bmp"`: First fingerprint image.
- `-i1 integer`: First fingerprint index in the database.
- `-f2 "finger2.bmp"`: Second fingerprint image.
- `-i2 integer`: Second fingerprint index in the database.
- `-gen "genFinger.bmp"`: Morphed fingerprint image.
- `-geni integer`: Index of morphed fingerprint image in the database.
- `-suf suffix`: Suffix of images in folder.

4.2.2 Console Application for Fingerprint Classification

The application is implemented in C++ with the use of the same tools as the previous application, described in Section 4.2.1 and with the use of functions described in Section 4.2.3. The application is meant to be run using python script located in `src/scripts/divideToClasses.py`. That is why the application is having the width and height of the image as arguments. The class is returned as a return code and also printed into the terminal. Return values are integer numbers: 0 is unknown, 1 is left loop, 2 is right loop, 3 is arch, 4 is whorl. The lowest size for image on input is 90 px in width and 90 px in height.

Application Control

`./class_detect.exe image_name width height`

- `imagename`: `path/to/image.bmp` of the fingerprint which is needed to get the type of class.
- `width`: Width of fingerprint image.
- `height`: Height of fingerprint image.

Python Script

Python script was created to run the application made in C++ over the whole folder. Script just recursively goes through a folder, finds all the images and puts each image on the input of the application. Based on the returned result, it copies result into suitable folder. The output folders are `arch`, `left_loop`, `right_loop`, `whorl` and `unknown` representing each class.

Script control

```
python divideToClasses.py input_folder input_image_suffix output_folder
path_to_class_detect.exe
```

- `input_folder`: Folder containing fingerprint images
- `input_image_suffix`: Suffix of fingerprint images (.bmp/.tiff/...)
- `output_folder`: Folder for outputting results
- `path_to_class_detect.exe`: Path to executable file of the application for classifying fingerprints

4.2.3 Used API Description

This Section describes the functions called from C++ API, that were needed for the implementation of the application for comparing fingerprints. The documentation that comes with IDKit only describes how to compare fingerprints with the database. Therefore the database functions are described as well. Most of the functions have return values (`IDKIT_API`), which can be found in the documentation ([27]).

Initialization and Connection to Database

Before using the whole API, first, the library has to be initialized, which comes also with a license check. This all is made with function `IDKIT_API int IEngine_InitModule()`. The function takes no parameters as same as the function that should be called on the of use of the API `IDKIT_API int IEngine_TerminateModule()`.

To connect the database that will contain biometric data `IDKIT_API int IEngine_Connect(const char * connectionString)` function is used. The function takes one parameter a connection string. It is allowed to connect to an SQLite database or memory database. If the string contains only the file name (for example `idkit.db`) the SQLite connection is selected automatically and if the file does not exist a new one is created. After finishing the job with the database, connections should be closed with the function `IDKIT_API int IEngine_CloseConnection(IENGINE_CONNECTION connection)`, which takes no parameters.

Register Fingerprint

The first step in registering a fingerprint is the initialization of the user, which is made by calling the API function `IDKIT_API IENGINE_USER IEngine_InitUser()`. This function returns an ID of the user or NULL in an unsuccessful case. The user is just created and it needs to be said that it is still not registered in the database. To free the user, function `IDKIT_API int IEngine_FreeUser(IENGINE_USER user)` is used.

Then we need to load the fingerprint picture and add it to the created user using the function
`IDKIT_API int IEngine_AddFingerprintFromFile(IENGINE_USER user,`
`IENGINE_FINGER_POSITION fingerPosition, const char * filename)`. The first parameter is the user we just created, the second is the finger to which the image of fingerprint belongs (for more see [27] page 106) and the last is the filename of the image we are trying to upload.

Now after assigning the fingerprint to the user we need to register the user to the database simply by calling `IDKIT_API int IEngine_RegisterUser(const IENGINE_USER user,`
`int * userID)`, where the first parameter is the user we created and the second is the return value for an ID of a just registered user. To remove the user from the database, the function `IDKIT_API int IEngine_Remove User(int userID)` is used.

To be able to create another user with its own data, the structure of the user needs to be cleared after storing it in the database. This is made by calling the function `IDKIT_API int IEngine_ClearUser(IENGINE_USER user)`.

Match Fingerprints

First, we need to prepare the user to be matched, as described in the Section before, or select user from the database with function `IDKIT_API int IEngine_GetUser(IENGINE_USER user, int userID)`, where the user is the output for founded user in the database by userID. The second threshold needs to be set as a deciding parameter with function `IDKIT_API int IEngine_SetParameter (IENGINE_CONFIG parameter, int value)`. The first parameter is the parameter we want to set (in our case `CFG_ SIMILARITY_THRESHOLD`), and we want to set the value as 0 in our case because we want all the results. Otherwise, all results below the threshold are set to 0. The default value for the threshold is 40.

Matching of two fingerprints is then done using a function `IDKIT_API int IEngine_MatchFingerprint(const IENGINE_USER user, int fingerprintIndex, int userID,`
`int * bestIndex, int * score)`. The first parameter is the input user with which will be made the comparison. Parameter fingerprintIndex is an index of the fingerprint that we are comparing. In my implementation, all fingerprints are always saved at index 0. Parameter userId is the id of the user in the database, against which we want to compare our user. Parameter bestIndex is output for best-matched fingerprint index from the user in DB. If this is not important and we do not need this output, the value can be set to NULL and this output is ignored. The last parameter score is the return value of the match score of two fingerprints. If the score falls below the threshold then the return value is 0.

Fingerprint Image from Database

To get fingerprint from db function `IDKIT_API int IEngine_GetFingerprintImage(const IENGINE_USER user, int fingerprintIndex, IENGINE_IMAGE_FORMAT format,`

`unsigned char * fingerprintImage, int * length)` is used. user is an input parameter for the user from which we want to get the fingerprint image, fingerprintIndex is an index of the fingerprint we want to get an image from. Formats are listed in [27] on page 106. fingerprintImage is the output image of our fingerprint and length is the size of allocated memory for the resulting image.

Detect Fingerprint Class

To detect the fingerprint class function `IDKIT_API int IEngine_GetFingerprintClass( const unsigned char* fingerprintImage, int length, int * fingerprintClass)` can be used. Parameters fingerprintImage and length are the same as described in Section 4.2.3. Parameter fingerprintClass is output for the class of fingerprint. API supports 4+1 kinds of classes and these are: left loop, right loop, arch, whorl + unknown.

4.3 Neurotec Biometric SDK

Neurotec Biometric SDK is a toolkit for all SDKs (MegaMatcher SDK, VeriFinger SDK, VeriLook SDK, VeriEye SDK, VeriSpeak SDK) from Neurotechnology. It contains all the necessary libraries, documentation and tutorial, and some sample programs. Where, of course, there is the possibility of connecting Different SDKs into one solution. [44]

VeriFinger is then only a fingerprint identification tool. VeriFinger allows both 1 to 1 and 1 to N fingerprint comparisons. The Neurotec fingerprint engine has been recognized by NIST in accordance with the MINEX tests. [44]

4.3.1 Fingerprint Match Console App VeriFinger v6 and v12.1

The applications were created according to the tutorial supplied with the Neurotec package in the C/C++ language. The application is used to compare fingerprints 1: 1 and returns the similarity score both on the return code of the application and on the standard output.

For evaluation using VeriFinger v6 or v12.1, it is necessary to have images with DPI 500, for which a python script `changeDPIFolder.py` was created in the scripts folder. The script is executed as: `python3 changeDPIFolder.py „input / folder" „DPI"`. The results of the converted files to the required DPI are generated in the `res` folder and the files have `.tif` extension.

Console Application VeriFinger v6 Controls

Upload the evaluation folder to start the application `evaluation_modules/Neurotec_Biometric_6_0_SDK_Trial/src` to folder `Neurotec_Biometric_6_0_SDK_Trial` and in the project, VerifyFinger changes the source file to main.c. Source files for included: `evaluation_modules/Neurotec_Biometric_6_0_SDK_Trial/src/Biometrics/C`, executable in: `/evaluation_modules/Neurotec_Biometric_6_0_SDK_Trial/Bin`.

`VerifyFinger.exe fingerprint1 fingerprint2`

- `fingerprint1` Path to the first fingerprint
- `fingerprint2` Path to the second fingerprint

A script (runtests.py) written in Python 3.7 was created to automatically evaluate the entire folder, which, as with the Innovatrics version, saves the output to a .csv file for

subsequent simple evaluation and creation of an FMR graph. The script is called: `python3 runtests.py „input /folder" „input.files.extension"`.

Console Application VeriFinger v12.1 Controls

The application was implemented using a tutorial, which can be downloaded with the VeriFinger v12.1. Therefore to translate the C++ application just copy the source file `VerifyFingerCPP.cpp` from `evaluation_modules/Neurotec_Biometric_12_1_SDK` to `/Neurotec_Biometric_12_1_SDK/Tutorials/Biometrics/CPP/VerifyFingerCPP`. Then compile the project with the language to C++ 14 (set the language as in 4.2.1) and use the generated ".exe" file.

`VerifyFinger.exe fingerprint1 fingerprint2`

- `fingerprint1` Path to the first fingerprint
- `fingerprint2` Path to the second fingerprint

The same script as in 4.3.1 is used for the evaluation of the whole folder.

4.3.2 Used API

This Section describes functions called from C++ API that was needed for the implementation of fingerprint the application for comparing fingerprints with the use of software from Neurotechnology.

Initialization

First, the `NBiometricClient` object has to be initialized. The initialization for the object is done without any parameters like `NBiometricClient biometricClient;`. The NBiometricClient is used for device integration such as fingerprint sensor and makes it easy to implement the typical workflow, such as enrollment of scanned images. In our case, we need to specify the threshold to 0 so nothing is cut to 0 for our FMR curve by `biometricClient.SetMatchingThreshold(0);` and set the level of matching speed to the lowest which means the precision to highest with `biometricClient.SetFingersMatchingSpeed(nmsLow);`.

Then the two subjects for verification need to be created by `NSubject subject;`. The initialization of the object again takes no parameters and everything is assigned later, because the subject insides depends on system used (can include faces, fingerprints, iris, ...). `NSubject` represents living creatures and stores their biometric data such as fingerprints or face. First the id of the subject is set by `subject.SetId(subjectId);` and then the finger object is created `NFinger finger;` with the loaded fingerprint image by `finger.SetFileName(fileName);`. In the end we add the fingerprint to our subject by `subject.GetFingers().Add(finger);`

Verification

After creating the both `NSubject`, lets say, `referenceSubject` and `candidateSubject` we can start the verification by `NBiometricClient` which was created before by `biometricClient.Verify(referenceSubject, candidateSubject);`. The verification returns `NBiometricStatus`. We look for return status is 1 (verification succeeded) or in case the threshold is set to any other value than 0. Then also return status is 610 which stands

for “Match not found“. All other status values are described in [45]. The matching result can be obtained from referenceSubject by
`referenceSubject.GetMatchingResults().Get(0).GetScore();`

Chapter 5

Evaluation of the Proposed Solution

To evaluate the application used for generating fingerprints by morphing a large database of fingerprints is needed. Because the database FVC2002 is used in the paper [16] which was a huge inspiration for this thesis the solution was tested over this database against VeriFinger v6. Than the database of fingerprints from the Brno University of Technology borrowed by Ing. Ondřej Kanich Ph.D. was used to generate new database of almost 10 000 morphed fingerprints to test the solution against software from VeriFinger v6 and Innovatrics.

5.1 Fingerprint Database EBD

The database was obtained by Security Technology Research and Development which is the Research group at the Faculty of Information Technology, Brno University of Technology. The database was scanned for academic purposes in several projects. Because of this purpose and legal terms in relating to privacy, none of the fingerprints are shown in this work. For visual demonstration of how the application works mostly public images were found by the Google search engine were used.

The database in the received version consists of 9450 fingerprints which makes up 945 different people, based on the fact that from every person fingerprints from all fingers on both hands were taken. The fingerprints were obtained by different sensors and this all gave more than enough variety for the application testing.

5.2 Fingerprint Database FVC2002

The database FVC2002 was used in the Second International Competition for Fingerprint Verification Algorithms. Only 31 competitors were included and the evaluation of competition was done in April 2002 where the results were presented at 16th International Conference on Pattern Recognition. [2]

The database consists of 4 sub-databases: [2]

- **DB1:** fingerprints were taken by the optical sensor „TouchView II“ by Identix,
- **DB2:** fingerprints were taken by the optical sensor „FX2000“ by Biometrika,
- **DB3:** fingerprints were taken by the capacitive sensor „100 SC“ by Precise Biometrics,

- **DB4:** fingerprints synthetically generated by SFinGe v2.51.

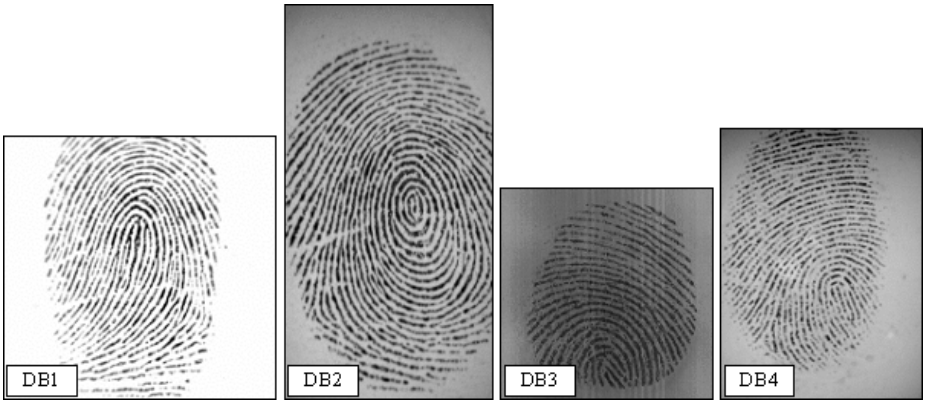

Figure 5.1: Sample image of each sub database from database FVC2002 [2]

For the use of this thesis, only the first database was used to compare the solution to the paper [16]. The free version of the database is downloadable on the webpage of the competition [2]. The free version consists of 80 fingerprints which are more than needed for the generation of fingerprints since the fingerprints are randomly selected.

5.3 Databases of Morphed Fingepritns

To evaluate the final applications, a reduction of the database mentioned in 5.1 was needed to decrease time complexity. On average, a fingerprint takes 37 seconds to generate on the working machine specified at the beginning of this Chapter. The time differs a lot based on the used block size. So the mentioned time was measured for 10-pixel blocksize, which was chosen based on the overall best results.

To try most of the variability and most possibilities, first, the fingerprints were sorted based on their classes with the use of the script from Section 4.2.2. Then 2 folders were created in each class folder with excluding the unknown class folder. Folder a) included 10 randomly chosen fingerprints and folder b) included 100 randomly chosen fingerprints. Then the morphing was run over these folders as each fingerprint from folder a) was combined with every fingerprint from folder b).

To generate interclass fingerprints, a combined folder was created. From the first two classes (left loop and right loop) 3 random fingerprints were selected. Then from the other two, only 2 fingerprints were selected. For folder b) 25 fingerprints from each class were selected and then the generation was run the same way. This generation was run twice, once for generation from the center and once from the core (described in Section 4.1). That means, that even with this reduction the time taken by generation was over 102 hours.

The finished result are 2 databases, each for one method described in Sections 3.5 and 3.6. Each database consists of 5 sub-databases representing one of the classes and one mixed. All together, there are 9 982 morphed fingerprints generated and prepared for the final evaluation of this thesis. An example of each class can be seen in Figure 5.2.

Figure 5.2: Sample fingerprint images of each sub-database from the database of morphed fingerprints, each letter represents one fingerprint generated out of each class: a) arch, b) left loop, c) right loop, d) whorl, e) combined.

The number of fingerprints is not exactly 10 000 because some generated fingerprints were incompatible with the applications for comparing fingerprints. Mostly because of the size, where the Innovatrics input has to be a fingerprint image at least 80 pixels wide and 80 pixels high.

Also, not all the generated fingerprints are perfect, some are very easily recognizable as fake by the human eye but the tested systems from Innovatrics or VeriFinger evaluated the comparison with a high score was found as the biggest hole for the Innovatrics or Neurotechnology fingerprint matching in comparison to morphed fingerprints. Where there can be two cores included in a fingerprint, but not as shown as in Figure 5.3. This fingerprint was generated by the method mentioned in Section 3.6 and it could achieve a similarity score of 122 compared to Innovatrics. These images are mostly included in the `res_center` folder for the whorl class of fingerprints.

Figure 5.3: Morphing result with two cores

The worst generated fingerprints are from the arch class. These fingerprints have their core in the lower part of the fingerprint which leads to bad results if we try to put the cutline in the core and divide the fingerprint into the lower and the higher part. Also because the methods for generating fingerprints do not include ridge line width adjustment, some fingerprints look similar to in Figure 5.4.

Figure 5.4: Bad case of morphed fingerprint

5.4 Results

As a first step, the first fingerprint generation (3.5) was evaluated against the paper [16] even though the paper says that the fingerprints were selected randomly. For evaluation, the 1 315 fingerprints were generated and tested against the VeriFinger v6 where this implementation was successful in 76.3 % of all comparisons against the default threshold. In the paper, they were successful in 81.1 % of all cases. The difference could be caused by the differently picked random images or some slight differences in implementation. The FMR curve of the result can be seen in Figure 5.5.

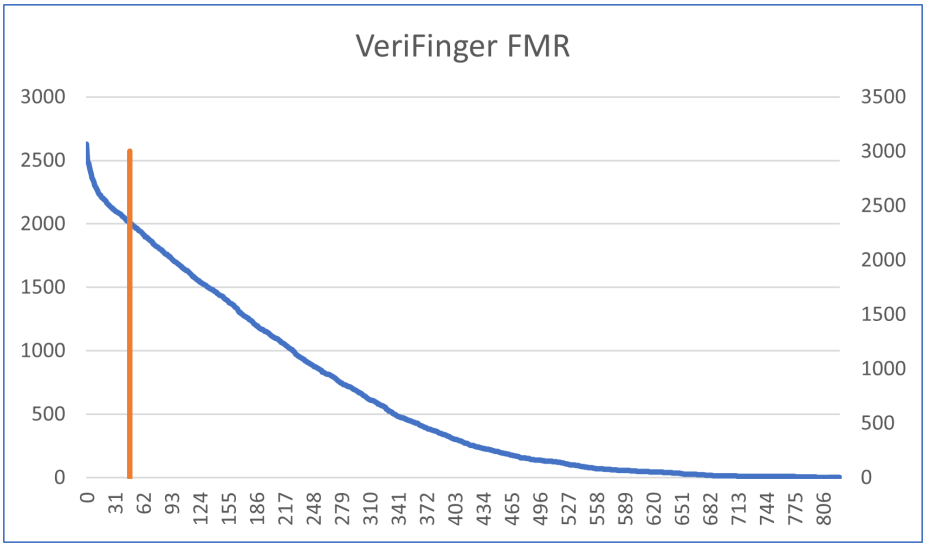

Figure 5.5: FMR for VeriFinger v6, the orange line shows the default threshold of 40.

After the generation of the morphed database and the first testing, the testing against the new versions of the software was run using the application from Section 4.2.2 over each folder containing the results from each class. As it can be seen in Figures 5.6 and 5.7. The method with morphing fingerprints, based on method 3.5, which uses the core of the fingerprint as barycenter is not as effective. With the comparison using the default threshold, only 24.68 % of the morphed fingerprints were effective for Innovatrics and 27.9 % for Neurotechnology VeriFinger.

We can also see that the least successful class of fingerprints was the arch which is because the core of this class is positioned far from the center of the fingerprint close to the bottom as shown in Figure 3.4. Where whorl, on the other hand, is placed in the center of fingerprint in most cases so the same part of both fingerprints is included in the result morphed image.

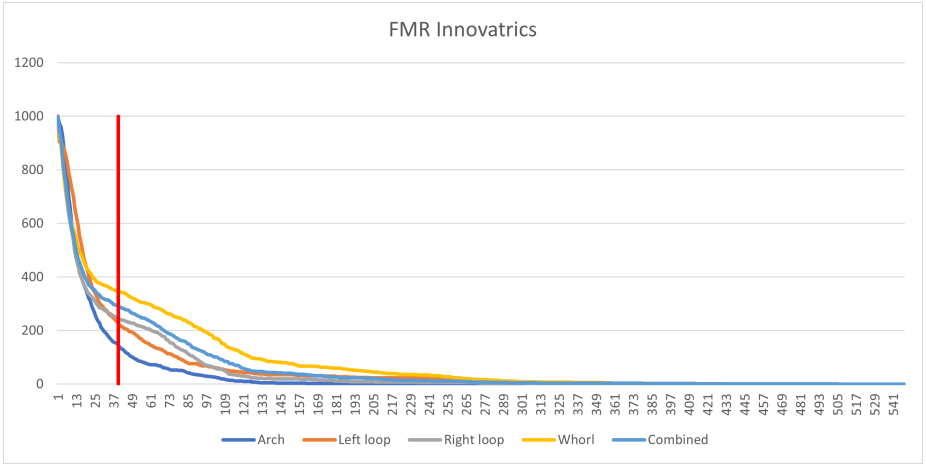

Figure 5.6: The FMR curve of results for each class for morphing from Section 3.5 for Innovatrics. The Red line shows the default threshold of 40.

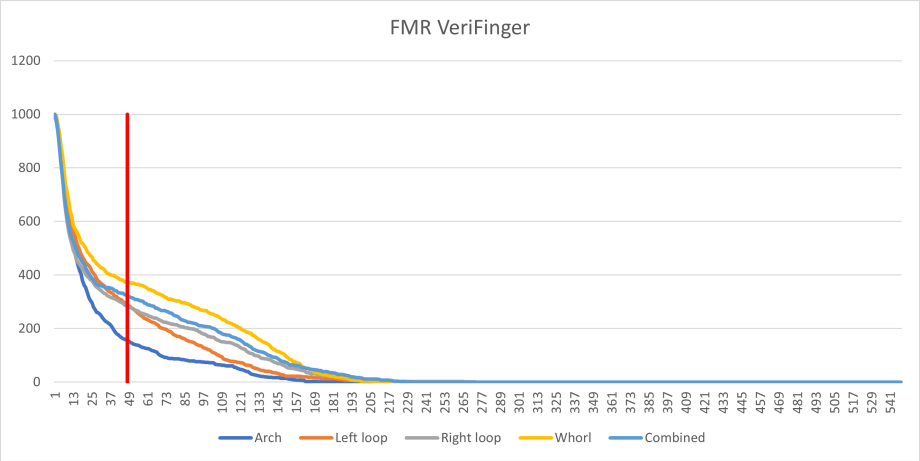

Figure 5.7: The FMR curve of results for each class for morphing from Section 3.5 for Neurotechnology VeriFinger. The Red line shows the default threshold of 48.

On the other side, the second method (described in 3.6), which uses the center of fingerprint as barycenter for the cutline and alignment based on masks, ended with 41.72 % for Innovatrics and 41.7 % for Neurotechnology VeriFinger with the comparison using the default threshold. The results are displayed in Figures 5.8 and 5.9.

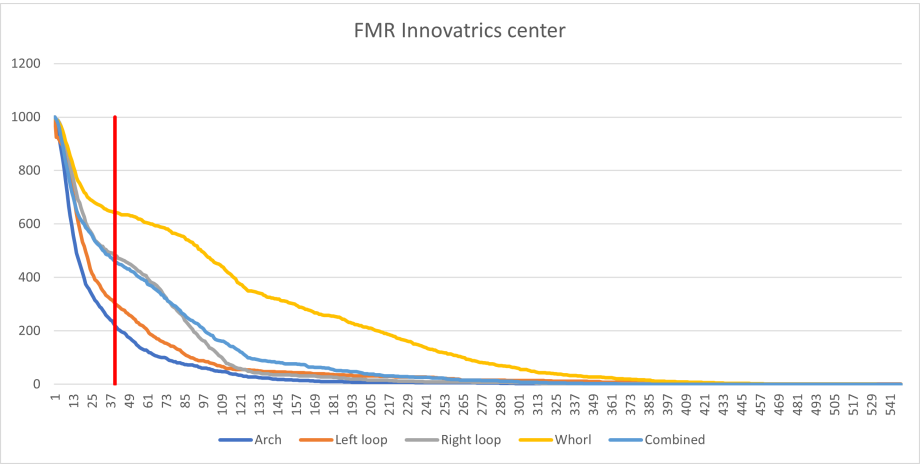

Figure 5.8: The FMR curve of results for each class for morphing from section 3.6 for Innovatrics. The Red line shows the default threshold of 40.

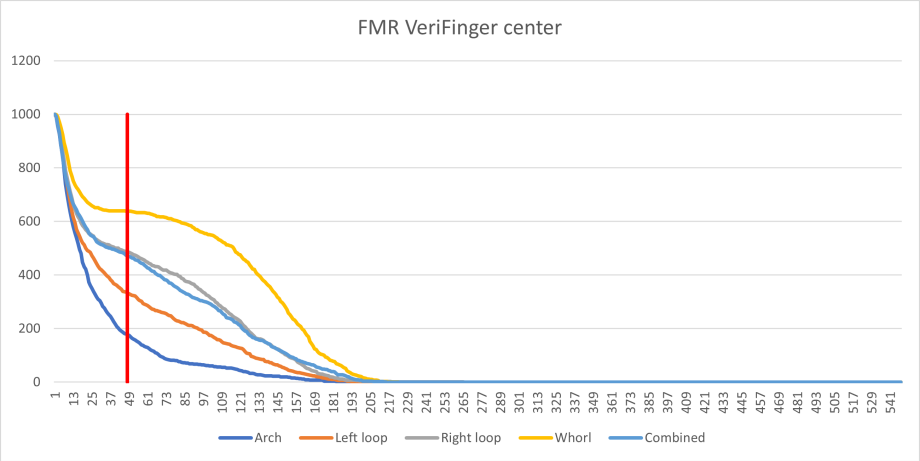

Figure 5.9: The FMR curve of results for each class for morphing from section 3.6 for Neurotechnology VeriFinger. The Red line shows the default threshold of 48.

Although the second image-based method tries to improve the method of morphing by moving the barycenter for cutline into the center of fingerprint, the arch class is still performing the worst, at around 20 %. This is caused because the arch class in most cases still has the most important part of the fingerprint at bottom of the fingerprint. So although the cutline is maximalizing the count of minutiae, the frequency and local orientation fields, the final result includes less important information from the second fingerprint.

On the other side stands whorl class, which is successful in more than 60 % of all cases. The core is situated in the center of the fingerprint and the position slightly differs which leads to fingerprints that contain both cores, as described and showed in Section 5.3. The tested software from Innovatrics and Neurotechnology VeriFinger is not capable of recognizing these fingerprints as fake and in contrast, evaluates them with high scores.

Chapter 6

Conclusion

This thesis contains the theory needed for the implementation of the applications for the fingerprint morphing and matching fingerprints and the theory behind capturing of fingerprints which was needed for the testing fingerprint database maintained by the faculty. This thesis also covers the design and implementation of an application used for the generation of fingerprints using the morphing methods. The aim of this work was to test fingerprint morphing methods against the real biometric systems from companies Innovatrics and Neurotechnology and to show the threat which this method represents. The first step was researched about existing solutions and only two were found in the paper On the Feasibility of Creating Double-Identity Fingerprints.

The paper contains a description of the two algorithms based on the same technique. Only the image-based morphing was implemented because the second technique was less effective in the author's results. It was decided to put more time into improving the image-based morphing and some steps were adjusted to make the method work better on fingerprints captured in different conditions. As the result, the improved method was more than 16 % successful than the first one compared to Innovatrics biometric system. The results were generated from a database provided by the Faculty of Information Technology at the Brno University of Technology.

The finding that the technique for the generation of fingerprints by morphing that was designed and implemented as the part of this thesis can have a success rate over 41 % against the real system is in my opinion the main benefit of this work. In the future, there is a lot of space for improvements. New techniques that are more precise can be found, for example, the cutline can be moved more to the center of minutiae of both fingerprints after the alignment, or some minimal amount of minutiae from each fingerprint that should be included from each fingerprint can maybe help to improve the result.

To summarize the whole thesis, Chapter 2 covers basic concerns (that need to ne understood): the basics about fingerprints and their recognition, capturing the fingerprints and a little bit from mathematical morphology and digital image processing. In Chapter 3 the suggested solution for morphing application, starting with pre-processing the input, aligning the fingerprints, computing the cutline, and generating the final result. Chapter 4 then describes the implementation of morphing applications and both applications for the detection and the description of used Innovatrics API. Chapter 5 then includes measured results and proves that the morphed images can be considered a threat to actual biometric systems.

Bibliography

[1] *Fingerprint, face, eye iris, voice and palm print identification, speaker and object recognition software* [https://www.neurotechnology.com/]. (Accessed on 07/25/2021).

[2] *FVC2002 - Second International Fingerprint Verification Competition* [http://bias.csr.unibo.it/fvc2002/]. (Accessed on 07/17/2021).

[3] *GitHub - mehmetaydar/fingerprint-alignment: Fingerprint enhancement, minutiaes extraction, core point detection, alignment of minutiaes based on core point.* [https://github.com/mehmetaydar/fingerprint-alignment]. (Accessed on 07/25/2021).

[4] *Innovatrics | Biometric Solutions | Trusted by Governments and Enterprise* [https://www.innovatrics.com/]. (Accessed on 07/25/2021).

[5] *Morphology - Thinning* [https://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm]. (Accessed on 12/23/2020).

[6] *Skin anatomy* [http://www.unm.edu/~lkravitz/Kinesiology/skinanatomy.html]. (Accessed on 12/24/2020).

[7] Akhtar, Z. Security of multimodal biometric systems against spoof attacks. march 2012.

[8] Babler, W. Embryologic development of epidermal ridges and their configurations. *Birth defects original article series*. 1991, vol. 27, no. 2, p. 95–112.

[9] Bolle, R., Pankanti, S. and Ratha, N. Evaluation techniques for biometrics-based authentication systems (FRR). In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. 2000, vol. 2, p. 831–837 vol.2. DOI: 10.1109/ICPR.2000.906204.

[10] Cappelli, R., Maio, D., Lumini, A. and Maltoni, D. Fingerprint Image Reconstruction from Standard Templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2007, vol. 29, no. 9, p. 1489–1503. DOI: 10.1109/TPAMI.2007.1087.

[11] Carneiro, R. F. L., Bessa, J. A., De Moraes, J. L., Neto, E. C. and De Alexandria, A. R. Techniques of binarization, thinning and feature extraction applied to a fingerprint system. *International Journal of Computer Applications*. Foundation of Computer Science. 2014, vol. 103, no. 10, p. 4.

[12] da Silva, E. A. and Mendonça, G. V. 4 - Digital Image Processing. In: CHEN, W.-K., ed. *The Electrical Engineering Handbook*. Burlington: Academic Press, 2005, p. 891–910. ISBN 978-0-12-170960-0.

[13] Daluz, H. M. *Fundamentals of fingerprint analysis*. CRC Press, 2018.

[14] Dougherty, E. *Mathematical morphology in image processing*. CRC press, 2018.

[15] Drahanský, M. and Kanich, O. *01. Úvod do biometrických systémů* [https://www.fit.vutbr.cz/study/courses/BIO/private/01_BIO_Prednaska_OK_2020.pdf]. (Accessed on 12/27/2020).

[16] Ferrara, M., Cappelli, R. and Maltoni, D. On the Feasibility of Creating Double-Identity Fingerprints. *IEEE Transactions on Information Forensics and Security*. december 2016, PP, p. 1–1. DOI: 10.1109/TIFS.2016.2639345.

[17] Fish, N., Zhang, R., Perry, L., Cohen Or, D., Shechtman, E. et al. Image morphing with perceptual constraints and stn alignment. In: Wiley Online Library. *Computer Graphics Forum*. 2020, vol. 39, no. 6, p. 303–313.

[18] Ghouzali, S., Lafkih, M., Abdul, W., Mikram, M., Haziti, M. et al. Trace Attack against Biometric Mobile Applications. *Mobile Information Systems*. january 2016, vol. 2016, p. 1–15. DOI: 10.1155/2016/2065948.

[19] Goutsias, J. and Heijmans, H. J. *Mathematical morphology*. IOS press, 2000.

[20] Haralick, R. M., Sternberg, S. R. and Zhuang, X. Image Analysis Using Mathematical Morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1987, PAMI-9, no. 4, p. 532–550. DOI: 10.1109/TPAMI.1987.4767941.

[21] Heijmans, H. J. and Ronse, C. The algebraic basis of mathematical morphology I. Dilations and erosions. *Computer Vision, Graphics, and Image Processing*. Elsevier. 1990, vol. 50, no. 3, p. 245–295.

[22] Henry, E. Classification and Uses of Finger Prints, London: Routledge. 1900.

[23] Holder, E. H., Robinson, L. O. and Laub, J. H. *The fingerprint sourcebook*. US Department. of Justice, Office of Justice Programs, National Institute of . . . , 2011.

[24] Hong, L. and Jain, A. Classification of fingerprint images. 1999, vol. 2, p. 665–672.

[25] Hong, L. and Jain, A. Fingerprint enhancement. Springer. 2004, p. 127–143.

[26] Hong, L., Wan, Y. and Jain, A. Fingerprint image enhancement: algorithm and performance evaluation. *IEEE transactions on pattern analysis and machine intelligence*. IEEE. 1998, vol. 20, no. 8, p. 777–789.

[27] Innovatrics. *IDKit SDK*. 2019.

[28] Jain, A., Ross, A. and Prabhakar, S. An introduction to biometric recognition. *IEEE Transactions on Circuits and Systems for Video Technology*. 2004, vol. 14, no. 1, p. 4–20. DOI: 10.1109/TCSVT.2003.818349.

[29] Jain, A. K., Flynn, P. and Ross, A. A. *Handbook of biometrics*. Springer Science & Business Media, 2007.

[30] Jain, A. K., Ross, A. A. and Nandakumar, K. *Introduction to biometrics*. Springer Science & Business Media, 2011.

[31] Jain, R. and Kant, C. Attacks on biometric systems: an overview. *International Journal of Advances in Scientific Research*. 2015, vol. 1, no. 07, p. 283–288.

[32] Jang, B.-K. and Chin, R. T. Analysis of thinning algorithms using mathematical morphology. *IEEE Transactions on pattern analysis and machine intelligence*. IEEE. 1990, vol. 12, no. 6, p. 541–551.

[33] Jiang, X. and Ser, W. Online fingerprint template improvement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2002, vol. 24, no. 8, p. 1121–1126. DOI: 10.1109/TPAMI.2002.1023807.

[34] Kasaei, S., Deriche, M. and Boashash, B. Fingerprint feature extraction using block-direction on reconstructed images. In:. January 1998, p. 303 – 306 vol.1. DOI: 10.1109/TENCON.1997.647317. ISBN 0-7803-4365-4.

[35] Khandelwal, C., Maheshwari, R. and Shinde, U. Review Paper on Applications of Principal Component Analysis in Multimodal Biometrics System. *Procedia Computer Science*. december 2016, vol. 92, p. 481–486. DOI: 10.1016/j.procs.2016.07.371.

[36] Liao, J., Lima, R. S., Nehab, D., Hoppe, H., Sander, P. V. et al. Automating Image Morphing Using Structural Similarity on a Halfway Domain. *ACM Trans. Graph.* New York, NY, USA: Association for Computing Machinery. september 2014, vol. 33, no. 5. DOI: 10.1145/2629494. ISSN 0730-0301.

[37] Lin Hong, Yifei Wan and Jain, A. Fingerprint image enhancement: algorithm and performance evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1998, vol. 20, no. 8, p. 777–789.

[38] Lu, Y. *Piezoelectric Micromachined Ultrasonic Transducers for Fingerprint Sensing*. Dissertation.

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

[40] Memon, S., Sepasian, M. and Balachandran, W. Review of finger print sensing technologies. In:. January 2009, p. 226 – 231. DOI: 10.1109/INMIC.2008.4777740.

[41] Msiza, I. S., Mistry, J., Leke Betechuoh, B., Nelwamondo, F. V. and Marwala, T. On the introduction of secondary fingerprint classification. *State of the art in Biometrics*. BoD–Books on Demand. 2011, p. 105.

[42] Mura, V., Orrù, G., Casula, R., Sibiriu, A., Loi, G. et al. LivDet 2017 Fingerprint Liveness Detection Competition 2017. 2018, p. 297–302. DOI: 10.1109/ICB2018.2018.00052.

[43] Mwema, J., Kimwele, M. and Kimani, S. A simple review of biometric template protection schemes used in preventing adversary attacks on biometric fingerprint templates. *International Journal of Computer Trends and Technology*. 2015, vol. 20, no. 1, p. 12–18.

[44] Neurotechnology. *MegaMatcher 6.0, VeriFinger 8.0, VeriLook 5.7, VeriEye 2.10 and VeriSpeak 3.0 SDK*. 3/18/2016.

[45] Neurotechnology. *MegaMatcher 12.1, VeriFinger 12.1, VeriLook 12.1, VeriEye 12.1 and VeriSpeak 12.1 SDK*. 4/15/2021.

[46] O'Gorman, L. Binarization and Multithresholding of Document Images Using Connectivity. *CVGIP: Graphical Models and Image Processing*. 1994, vol. 56, no. 6, p. 494–506. ISSN 1049-9652.

[47] Patriciu, V.-V. and Spinu, S. Fingerprint Ridge Frequency Estimation in the Fourier Domain. *Advances in Electrical and Computer Engineering*. Stefan cel Mare University of Suceava. 2014, vol. 14, no. 4, p. 95–98.

[48] Roy, P., Dutta, S., Dey, N., Dey, G., Chakraborty, S. et al. Adaptive thresholding: A comparative study. 2014, p. 1182–1186. DOI: 10.1109/ICCICCT.2014.6993140.

[49] Schuckers, M. E. *Computational methods in biometric authentication: statistical methods for performance evaluation*. Springer Science & Business Media, 2010.

[50] Serra, J. Mathematical morphology for complete lattices. *Image analysis and mathematical morphology*. Academic Press New York. 1988, vol. 2, p. 13–35.

[51] Serra, J. *Image Analysis and Mathematical Morphology*. USA: Academic Press, Inc., 1983. ISBN 0126372403.

[52] Shih, F. Y. *Image processing and mathematical morphology: fundamentals and applications*. CRC press, 2009.

[53] Sivaram, M., Ahamed A, M. U., Yuvaraj, D., Megala, G., Porkodi, V. et al. Biometric Security and Performance Metrics: FAR, FER, CER, FRR. In: *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*. 2019, p. 770–772. DOI: 10.1109/ICCIKE47802.2019.9004275.

[54] Song, K.-H., Choi, J. and Chun, J.-H. A Method for Enhancing the Sensing Distance of a Fingerprint Sensor. *Sensors*. october 2017, vol. 17, p. 2280. DOI: 10.3390/s17102280.

[55] Toet, A. and Wu, T. Efficient contrast enhancement through log-power histogram modification. *Journal of Electronic Imaging*. december 2014, vol. 23, p. 063017. DOI: 10.1117/1.JEI.23.6.063017.

[56] Wayman, J. L., Jain, A. K., Maltoni, D. and Maio, D. *Biometric systems: Technology, design and performance evaluation*. Springer Science & Business Media, 2005.

[57] Wieclaw, L. A minutiae-based matching algorithms in fingerprint recognition systems. july 2021.

[58] Wolberg, G. Image morphing: a survey. *The visual computer*. Springer-Verlag. 1998, vol. 14, no. 8, p. 360–372.

[59] Zaeri, N. Minutiae-based Fingerprint Extraction and Recognition. In:. June 2011. DOI: 10.5772/17527. ISBN 978-953-307-618-8.

Appendix A

Contents of the Attached storage media

- `db` - Databases of Morphed Fingepritns
- `evaluation_modules` - Source codes for fingerprint matching and classifying terminal applications
- `morphing` - Source codes for terminal fingerprint morphing application
- `thesis` - Digital version of this thesis
- `pip-install-req.sh` - Bash script for installing needed Python 3 packages
- `README.md` - Contexts of storage media