August 25, 2014

BACHELOR ASSIGNMENT

# FINGERPRINT ACQUISITION WITH A SMARTPHONE CAMERA

Wout Oude Elferink
s1225588

**Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)**
**Services, Cybersecurity and Safety**

Exam committee:
Prof.Dr.Ir. R.N.J. Veldhuis
Dr.Ir. L.J. Spreeuwers
Dr. M.M.J. Dhallé

UNIVERSITY OF TWENTE.

# Abstract

Fingerprint identification systems have been used for over a century to identify individual persons. In all those years the methods for identification gradually improved. Nowadays several different systems for acquiring digital fingerprint images have been developed. Alongside the development for sensor systems, better algorithms for enhancing and comparing fingerprints have been made.

Recently, the rapid developments in mobile phone technology made it possible for a big public to own a smartphone with a digital camera. Research on the use of this camera as fingerprint image capture device showed the possibilities of using it as a biometric identification system. The research done however focused on making novel algorithms starting from scratch.

In this research the possibilities of using software, which is developed for dedicated fingerprint capture devices, on the images captured with a smartphone camera are explored. By using this software instead of developing an all new algorithm, the knowledge gathered over more than a decade in enhancing dedicated sensor fingerprint images can be put to use directly.

In this report a new algorithm to improve smartphone captured fingerprint images is suggested. The algorithm focusses on enhancing the images in such a way that they show the same characteristics as the images from dedicated sensors. To do this, the software first segments the finger from the background, converts the image to gray scale by averaging the green and blue channel, normalizes the image according to its mean in a block size of 30x30 pixels and enhances the image according to its ridge orientation.

It is shown that the enhancement algorithm improves the quality of the images drastically. A comparison test between a dedicated sensor and the smartphone camera is carried out which shows that the smartphone has a comparable performance to the dedicated sensor. Furthermore a cross operability test is done between the images from a dedicated sensor and the smartphone. While the performance does go down, it is shown that there are certainly possibilities for cross matching these fingerprints.

# Acknowledgements

This report is written for the third year Bachelor assignment from Advanced Technology. The research is carried out at the Services, Cybersecurity and Safety (SCS) research group at the University of Twente in the last quartile of the 2013-2014 study year.

I would like to thank my supervisors Raymond Veldhuis and Luuk Spreeuwers for their feedback and help during the project. Furthermore I would like to thank Geert Jan Laanstra for helping with the practical part of the assignment.

I would also like to express my gratitude to the people who have willingly let me use their fingerprints for setting up the database. And finally I would like to thank my room mates and friends for their support and shown interest in the subject.

# Contents

# List of Abbreviations

DSCF = dedicated sensor captured fingerprint

EER = equal error rate

FAR = false acceptance rate

FMR = false match rate, is the same as FAR

FNMR = false non-match rate, is the same as FRR

FRR = false rejection rate

PPI = pixels per inch

ROI = region of interest

SCF = smartphone captured fingerprint

std = standard deviation

STFT = short time fourier transform

# 1 Introduction

## 1.1 Motivation

The usage of fingerprints to identify people has already been around for some while. It is used by forensic analysts to identify suspects in crime scenes and by the government in biometric passports. For these applications extensive research has been done in enhancing the fingerprint quality and extracting the minutiae points (the features of a fingerprint which are used for recognition). Recently, smartphone manufacturers also adopted fingerprint identification systems to unlock the phone or make payments [4]. To be able to get these fingerprints, a dedicated sensor is used. These sensors however take extra space in the smartphone and are to costly for low- to mid-end smartphones. The touch based sensors used nowadays also show problems with pressing a curved finger on a flat surface and leaving a print and dirt on the sensor. Furthermore, older smartphones don't possess such a dedicated sensor. These smartphones do however all already possess a camera. This camera can be exploited to produce high ppi fingerprint images. While these images can easily be captured, there are still many problems to produce reliable identification systems with these images. Problems mentioned in literature are a low ridge/valley contrast [8,9,11,14], low depth of field of the camera resulting in partly focused fingers [8,9,14], motion blurriness [8,9,11,16], very erratic backgrounds depending on the on time and place of capturing [11,14], perspective problems [6,11], finger rotation and distance differences [11,22] and shading of the finger due to its convex form [11]. Some of these problems are inherent to the capturing process, but others are also common within the world of dedicated sensors. While these problems are attempted to be solved by making better image enhancement algorithms [11], it is also mentioned that processing algorithms used nowadays are designed for images captured with capacitive or optical sensors [22]. The problem with these algorithms designed for dedicated sensors is not that they do not work, in fact they show impressive results on images captured with dedicated sensors, the problem is that the images from a smartphone camera are not properly recognized and enhanced by these algorithms. In this article a new approach is taken to enhance these images in such a way that they show similarities with fingerprint images from dedicated sensors. In this way the knowledge and powerful algorithms developed for the dedicated sensors can be used without the need of redesigning this software for smartphone camera captured fingerprint images.

## 1.2 Objectives

The quality of smartphone captured fingerprints (SCF) will be assessed by calculating the comparison score with the used enhancement and minutiae extractor software by VeriFinger and comparing this to the score of dedicated sensors captured fingerprint (DSCF) images. Furthermore a cross-match between SCF and DSCF images will be made to see if the images from these different sensors can be used alongside each other.

Fingerprint images captured using a smartphone camera will be enhanced in such a way that the VeriFinger software is able to identify and match these fingerprints against templates extracted from images of a smartphone camera and a dedicated sensor. To test the enhancement a comparison test of smartphone captured fingerprints (SCF) against SCF, SCF against dedicated sensor captured fingerprints (DSCF), and DSCF against DSCF is needed. In the ideal case these fingerprints matches should show similar quality

scores. To get these scores as high as possible, the SCF should be enhanced to have the most similarities with DSCF.

To enhance these images, it should first of all be clear what images from dedicated sensors look like. Example images of both optical and capacitive sensors can be found in figure 1.1 as well as an example image taken with a smartphone camera. The images from the dedicated sensors show a clear ridge valley pattern, but it should be noticed that there are many gaps inside the ridge structure. There are also visible scar lines on the images of both the dedicated sensor and camera. The image of the SCF also shows a clear ridge valley pattern, but the ridge valley pattern is much smoother causing trouble finding the exact ending of a ridge. There are also problems with shadows and light and dark spots on the finger. And finally the ridge pattern gets unclear near the edges due to the convex shape of the finger. To reduce the differences between SCF and DSCF and to be able to process the fingerprint by the VeriFinger software, the image should first be enhanced. To make sure that we do not start all over again developing ideas, an overview of related work and progress in this field is given.

Figure 1.1: Example images from dedicated sensors a) image from the Cross Match Verifier 300 dedicated sensor, image taken from the verifinger sample database [5] , b,c) image from the TouchView II optical sensor, image taken from the FVC2002 database 1 [1] d) image from the FX2000 optical sensor, image taken from the FVC2002 database 2 [1] e) image from the 100 SC capacitive sensor, image taken from the FVC2002 database 3 [1] f) cropped SCF image, image taken with the camera of the Sony Xperia P

## 1.3 Approach

To get to the goal of making an algorithm that enhances SCF images in such a way that it can be used in the VeriFinger software to reliably match fingerprints of SCF against SCF or DSCF, an approach is taken that consist of the following steps:

- A literature study on related work.
- Establishing a small database of SCF test images and a larger database of SCF and DSCF images to extract reliable results.
- Development of an enhancement algorithm which should consist of segmenting the finger from the background, converting the image to grayscale and enhancing the ridge pattern.
- Performing performance evaluation experiments to show the added value of the enhancement algorithm.

## 1.4 Intended Audience

This report is meant to be read by students and professors in the area of computer science or electrical engineering. It is assumed that the reader has no prior knowledge of fingerprint enhancement algorithms. However, having a basic understanding of algorithms and mathematics is required.

# 2 Related Work

An overview of related work is given below so we are able to see what has already been done in the area of enhancing SCF images. An attempt is made to make the text as comprehensible as possible. It is however not possible to describe the algorithms used in other work in detail. For the algorithms that are described in the related work and are also used in this research a more comprehensive description is given later on.

## 2.1 An Overview

There has already been some research in the field of gathering fingerprint images with smartphone cameras or webcams and processing them. Stein et al. [21] acquire the image with a smarthpone camera as a video stream, for this the camera on the smartphone must be able to focus on close objects and have an LED. The usage of the LED makes the finger appear brighter than the background which simplifies detection and segmentation of the finger, it also reduces camera noise and risk of blurring, and finally it stabilizes lighting conditions and provides a more homogeneous illumination. Capturing with a video stream can improve recognition by having more perspectives on the finger, and makes it possible to have advanced anti-spoofing techniques. The images are preprocessed by first segmenting the foreground and background by a threshold on the red channel of the image. Next the image is transformed to gray scale. Then a median filter with kernel size of 3 is used to reduce camera noise. These images are then binarized by applying an adaptive threshold, for this a block size of 19x19 is used for images with a resolution of 1280x720. Then the width of the image is scaled to a fixed size and the height is adjusted accordingly. They then use a minutiae extractor and template comparator, the MorphoLite SDK, to identify the images. Only the images which pass the quality check are used for comparison. Using this method the results as can be seen in 2.1 were produced.

Table 2.1: Results produced by the method of Stein et al. [21]

| Capture Method and Device | EER (equal error rate) | FRR (false rejection rate) (@FAR = 0.1%) |
|---|---|---|
| Photo, Nexus S | 1.20% | 2.70% |
| Photo, Galaxy Nexus | 3.10% | 6.70% |
| Video, Galaxy Nexus | 3.00% | 12.10% |

Mueller et al. [17] describe how images taken from a webcam are processed and used in an USB smart card system. The webcams used are the Microsoft Lifecam VX3000 and the Philips SPC630NC. It is found that the image techniques histogram stretching and normalization are not sufficient due to significant shadows and regional overexposure of the image. The processing steps which Mueller et al. used are a Gamma correction and enhancing the gray scale image with directional fourier filters. For testing, a total of 400 fingerprint images of 36 fingers from 6 persons were generated. These are compared to FVC2002 databases db1a through db4a. The image quality is tested with the NFIQ algorithm from NIST. From the quality comparison it becomes clear that the fingerprints from the webcam score very bad with almost all images. About 80% of the webcam images have the lowest score. While the database images have a much higher score. They also show some data for the false acceptance rate (FAR) and false rejection rate

(FRR) as can be seen in table 2.2.

Table 2.2: Results produced by the method of Mueller et al.

| Database | FAR (false acceptance rate) | FRR (false rejection rate) |
|---|---|---|
| Webcam | 0.18% | 10.29% |
| FVC2002 | 0.02% | 6.54% |

Hiew et al. [7,8] describe how pre and post processing can be done on images acquired with a digital camera as seen in figure 2.1. The processing consists of a conversion to gray scale. Then local normalization is used to reduce the problem of non-uniform lighting, the local normalization is computed with:

$$S(x_0, y_0) = \frac{o(x_0, y_0) - m_0(x_0, y_0)}{\sigma_0(x_0, y_0)} \tag{2.1}$$

where $o(x_0, y_0)$ is the original image, $m_0(x_0, y_0)$ denotes an estimation of a local mean of the original image and $\sigma_0(x_0, y_0)$ signifies an estimation of the local standard deviation. Meanwhile the fingerprint region is segmented from the raw image by applying skin color detection - on the chrominance component in YCbCr color space by comparing to 1056 human skin color patches - , adaptive tresholding and morphological processing. Now the image is cropped and enhanced by using the information of the ridge frequency using a Short Time Fourier Transform (STFT) and information about the ridge orientation. Finally, the core point is detected from the enhanced image. They conclude that it becomes clear that cropping and normalizing are an essential step in image enhancement. And they determine that it also shows that the proposed algorithm performs better than the enhancement by Hong et al. [10] and also performs better than root filtering, a standard image processing algorithm to enhance high spatial frequency information.

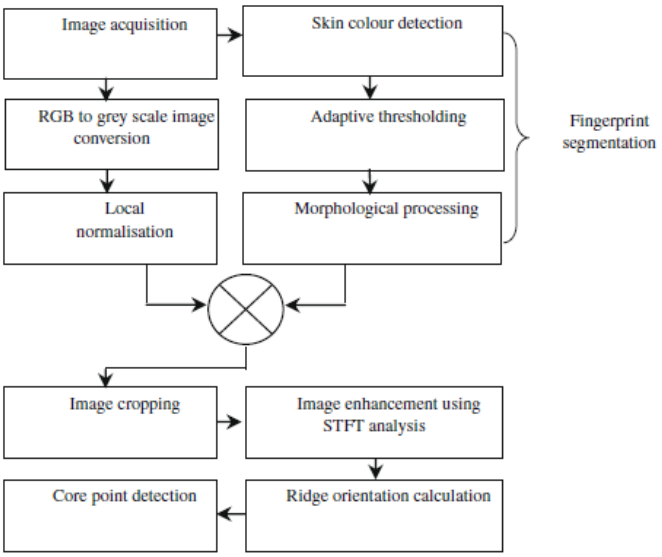



Figure 2.1: Image processing steps of Hiew et al. [8]

Piuri et al. [19] describe the enhancement process for fingerprint images acquired with a webcam. As preprocessing a Lucy-Richardson filter is used, with the point spread function estimated using a rotationally symmetric Gaussian lowpass filter with a squared matrix of 10 pixels with a spread of 3 pixels for a 1 mega pixel image. This algorithm can enhance a blurred image by calculating the most likely pixel values. The image segmentation is done by applying an upper and lower bound for each channel in YUV color

space, after which the holes are filled and an erosion with a circle of 1/50 image width is performed. For the last processing step a frequency band-pass filter is applied to the output, with the two cutoff spatial frequencies centered to the mean ridge frequency estimated with the Gabor filter approach. The images are then handed over to NIST comparison software and compared to the results from a dedicated sensor. Their results can be found in figure 2.2.

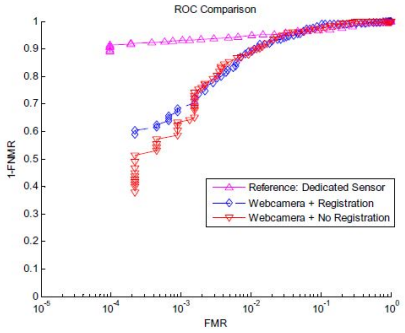



Figure 2.2: Comparison of the ROC curves between the presented techniques and the reference method: the complement of the False Non-Match Rate (FNMR) is plotted vs. the False Match Rate (FMR). From [19]

Lee et al. [14] show a method for enhancing fingerprint images captured with a mobile camera. First the fingerprint is segmented from the background. This is done with the information about color - the color of human fingerprints is similar which can be used for segmentation - , Texture - the fingerprint region shows a clear pattern but the background is blurred due to a small depth of field - and size - the fingerprint is the largest object in the image -. The results of color and texture information are combined and the largest region in the combined images is extracted. For the color segmentation a set of training images is created and the fingerprints are manually extracted. From these images the colors are transformed into the normalized color space, of these normalized images the mean and covariance of the color distribution are determined. With these test images a 2D Gaussian fingerprint model is made. With this model, the Mahalanobis distance - a measure of the distance between a point and a distribution - between an image and the model is determined and a threshold is defined for segmenting between skin color and background. The images are also segmented based on texture information. This is done with the use of a Discrete Wavelet Transform using Daubechies' 9/7 filter, after which a threshold is applied. The color and texture segmented images are combined with an AND operation and morphological processing is applied to gain the final image segmentation mask. Next the image gradients are determined, because of noise a simple least square methods cannot be used and an iterative least square method is proposed which removes the outliers for a more reliable orientation estimation. After this a fingerprint quality estimation is done. It is stated that the performance of the system is very sensitive to the quality of the captured fingerprint images. There are a couple of factors which lead to poor quality: Weak contrast caused by an incorrect field of depth, fingerprint scars and fingerprint abrasion. The experiments were conducted with a finger placed about 5 cm in front of the sensor. The images were 24-bit color images with a resolution of 640x512. The segmentation result is determined from the manually defined (true) and the computed (result) blocks on an 8x8 block basis and has the results as seen in table 2.3.

Table 2.3: Add caption

| True \ Result | Fingerprint Region | Background Region |
|---|---|---|
| Fingerprint Region | 98.06% | 1.94% |
| Background Region | 0.04% | 99.96% |

According to their images, the proposed orientation algorithm performs better than the least squares method. The quality estimation is compared with other quality estimation schemes and using manual false minutiae detection it is concluded that the proposed method works best as can be seen in figure 2.3.

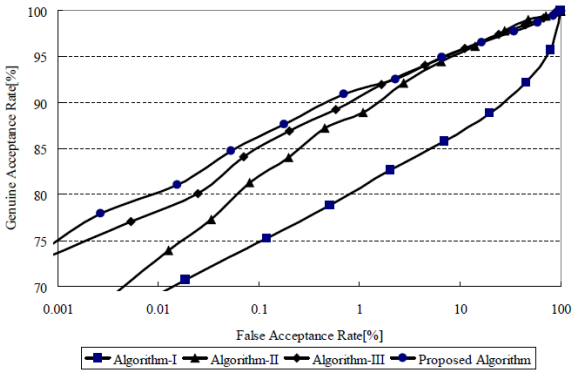



Fig. 16. Receiver Operating Curves(ROC); the ROC shows the improvement in verification performance using the preprocess algorithm

TABLE III
THE EXISTING ALGORITHM

| Algorithm | Segmentation | Orientation | Quality |
|---|---|---|---|
| I | Proposed method | Gradient Based | WSQ based |
| II | | | Gabor filter based |
| III | | | Coherence Based |

Figure 2.3: Comparison of the ROC curves. From [14]

Han et al. [6] describe the compatibility of photographed images with touch-based fingerprint verification software. For this the VeriFinger SDK is used. They first determine that the touch-based fingerprint has some flaws: different pressure on the sensor, improper finger placement, skin deformation, sensor noise and latent fingerprints. But the touch-less sensor also has some flaws: low ridge-valley contrast due to motion blur, noise and small depth of field and usually a strictly controlled environment has to be used. The fingerprint images were taken with two mobile phones with 5 mega pixel cameras (the Nokia X6 and the HTC HD2). From the experiments it is concluded that the best images are taken under these conditions:

- Direct sunshine without flash either inside or outside.
- Distance between finger and camera around 10cm, and the finger occupies the major area of the image.
- Finger parallel to the camera. Portrait orientation is preferred.
- No zoom, although macro is possible if the whole last joint of the finger can be included.
- Best images can be taken on a dark color background, such as black and blue background.

Only the size of the taken images are modified before they are processed by the VeriFinger software, no preprocessing is done. The results show that some minutiae points are not detected in the sample images as compared to the template, and the minutiae that are matched can have variable distance in between them. But a variation of 9% is still tolerable by the VeriFinger software. It is shown that the VeriFinger SDK can be used to compare and verify fingerprint images. They conclude that a 3D model of a fingerprint should be acquired. The 2D photographed images should then be mapped to the 3D model to retrieve the real futures on the 2D fingerprint image.

Li et al [16] conclude that images taken with a smartphone camera do only produce satisfying results in well controlled environments, showing EER rates of almost 50 percent in the worst environments. They however use the less good NIST extraction software and do not elaborate on their preprocessing.

Stein et al. [22] develop a recognition system for Android which includes capturing and preprocessing the images. For the capturing of the photo, the application sets the focus to macro, the LED is switched on, which makes segmentation easier, reduces the camera noise and blurring due to motion, it also stabilizes lightning conditions and creates more homogenous illumination. A photo is automatically taken when all criteria for the fingerphoto recognition are fulfilled. A quality assurance algorithm is used to detect if the sharpness is high enough for finding the minutiae points. Proposed is an edge-based approach, which is based on the fact that high frequency transitions (strong edges) can only appear in sharp images. For this a Sobel filter - a filter to find edges in an image - is used, this is combined with the own developed metric "edge density" to measure the sharpness of an image. After the fingerphoto is acquired, the finger is detected by cropping the image from the border until the red channel is over a defined threshold. Also only the upper segment of the finger is used and the rest is discarded. To compare photos, it is necessary that the finger is not rotated, so the fingers rotation is calculated to compensate for this. The rotation is calculated by setting two points in the image and moving these points to the edge until the border is reached (red value is exceeded). The correction angle can then be calculated by:

$$\alpha = atan(\frac{Py}{P2x - P1x}) \tag{2.2}$$

where Py is the distance between the points in vertical direction and $P2x - P1x$ is the distance between the points in x direction after they are moved to the edge.

After the correction, the foreground area has some possibility checks. The width and height as well as their ratio is checked. If these parameters do not correspond (for example image wider than the height), the image is discarded.

Next the width of the finger is scaled to a fixed width (and the height is scaled accordingly). This compensates the effect of different distances from the camera. Next a median filter is applied to reduce camera noise. And finally a local binarization filter is used to binarize the image (only using the red channel).

Now the minutiae are extracted from the preprocessed finger by the open source minutia extractor FingerJetFX from DigitalPersona. The generated templates are stored according to the standardized format ISO/IEC 19794-2.

The accuracy of the detected minutiae is not very high. Many false minutiae are detected and a high error in the calculated orientation angle could often be observed. The deficiency of the extractor is likely to be caused by the fact that the algorithm was designed to operate on images from optical or capacitive sensors. All extracted minutiae show a high quality score, making this scoring useless. The strength of the algorithm is however that its computational effort is low.

The algorithms are implemented in an android application and evaluated in a user test with 41 subjects using the Samsung Nexus S and Samsung Galaxy Nexus smartphones. The photos are taken with the highest supported resolution of the camera (5 megapixels), but were reduced to half of that because of memory issues on the smartphone. The data capture was on two different days. Three templates were created from the left and right index finger of each subject. Two photos for each template, with the one with the highest edge density approved and the other discarded.

In total, 1494 fingerphotos passed the algorithm for quality assurance and were used for testing. Manual

inspection showed that 51 finger foreground areas were not properly detected (3%), these errors were either falsely detected borders or a falsely calculated orientation angle. This finally resulted in figure 2.4

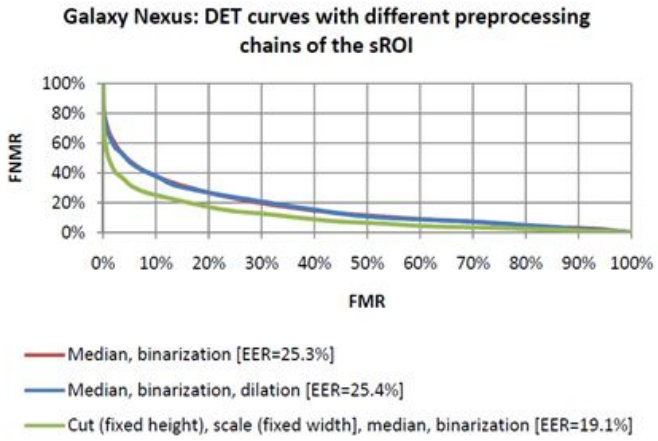



Figure 2.4: DET curves. From [22]

Labati et al. [13] describe a method for compensating rotation, pitching and other perspective problems. They first enhance the image, the region of interest (ROI) is determined using Otsu's method and refined by a morphological filling operator. Then the background is removed and the ridge pattern is enhanced by taking the log of the image. Next there is noise reduction by applying an 8-order Butterworth low pass filter with empirically determined size and cut off frequency. The enhancement and binarization is done by using the NIST mindtct algorithm. By knowing the distance from the camera, the image is normalized to a PPI of 500. Finally the fingers rotation is determined in several ways. First it is determined by checking the horizontal symmetry (especially in the top of the finger, the symmetry gives clues about the rotation). Second the rolling is determined by using symmetric Gabor filters. The final rotation is determined by a neural network based on these 2 parameters. It is shown that the rotational compensation lowers the EER from 3.04% to 2.20%.

## 2.2 Conclusion

From these articles it can be concluded which steps are taken in the enhancement of fingerprint images. The fingerprint should first be segmented from the background, either by a color threshold [21], skin color detection and an adaptive threshold [7, 8], via color, texture and size information [14], variance threshold [11], mean and covariance in YCbCr color space [11] or by Otsu's method [13].
It is also needed to convert the image to a gray scale image. Furthermore it is described how noise is reduced via median filters in [7,21,22] or an 8-order Butterworth low pass filter [13]. The image enhancement step is done in very different ways. Using gamma correction and directional fourier filters [17], enhancement in the fourier domain [7, 8], Lucy-Richardson filter, band-pass filter and gabor filter [19] and gabor filters [11]. Normalization is also a possible step as proposed in [7,8]. Quality estimation is done to remove unreliable areas using a recursive estimation of the coherence of the texture [14], using VMLOG [15] or by an edge based approach using a sobel filter [22]. There are also articles in which the image is binarized via an adaptive threshold [7, 21, 22]. Furthermore most images are scaled to a fixed width and an accordingly scaled height [8, 21, 22]. Rotation compensation is also mentioned in [22].

Which algorithms show the best performance is hard to tell. The used capturing devices differ between the articles, as well as the capture method and the used minutiae extraction software. This makes it impossible to determine the best approach. From the work by Stein et al. it can however be concluded

that the processing algorithm should not be too computationally expensive if it has to be deployed on a smartphone. In table 2.4 an overview of the performance of the different approaches can be found.

Table 2.4: Overview of performance

| **Author** | **Capture Method** | **Device** | **Enhancement Method** | **Minutiae extractor** | **EER** |
|---|---|---|---|---|---|
| Stein et al. [21] | LED on, non-erratic static background, macro mode | Nexus S | Median filter, binarization with adaptive treshold, scaling to a fixed size, quality check | MorphoLite SDK | 1.20% |
| | | Galaxy Nexus | | | 3.10% |
| Mueller et al. [17] | webcam | Microsoft Lifecam VX3000 and Philips SPC630NC | Gamma correction, directional fourier filters | NIST | FAR is 0.18% @ FRR is 10.29% |
| Piuri et al. [19] | webcam, daylight | Microsoft Lifecam VX-1000 | Lucy-Richardson filter, bandpass filter, scaling with respect to finger width, rotation compensation | NIST | 4.20% |
| | | Crossmatch Verifier 300 optical sensor (reference) | | NIST | 4% |
| Stein et al. [22] | Led on, macro mode | Galaxy Nexus | Quality check, rotation compensation, scaling to a fixed width, adaptive binarization, processing done on mobile device | FingerJetFX | 19.10% |
| Labati et al. [13] | Blue light, fixed distance, finger resting on a surface | Sony XCD-SX90CR CCD camera | Log of image, low pass filter, ppi normalization, rotation compensation using a neural network | NIST | 2.20% |

# 3 Fingerprint Photo Acquisition

To be able to make the photos consistently the same and make the process repeatable, an image capturing protocol is made. In the literature different ways of capturing the images are described. It is found that the best results where achieved with a smooth single coloured background, the LED of the phone turned on for a constant and uniform lighting and with the focus as close set to macro [22]. Therefore in this research the image capturing conditions are kept constant with these parameters:

- The background is a white sheet of A4 paper.
- The finger is resting on the paper.
- The distance between the finger and camera is 6 cm, which is the closest focus distance of the camera.
- The light conditions are a well-lit office environment and the flash LED of the smartphone is on.
- The focus is set to autofocus (which includes macro).
- The image resolution is set to the maximum of 3264x2248 pixels.
- The smartphone is held steady in a holder.
- The used smartphone is a Sony Xperia P.
- The used dedicated sensor is the U are U 4500 fingerprint scanner form digitalPersona.

For the data set, photos are acquired from 11 different persons.[1] From these persons 8 fingers are captured - every finger except the thumbs - twice. First four fingers on the right hand are recorded with the smartphone and then four from the left hand. This is repeated once, capturing a total of 16 photos with the smartphone per person. For the dedicated sensor the fingers are collected in the same order. This results in a dataset of 176 smartphone photos and 176 images from the dedicated sensor. If the captured finger is out of focus, the photo is taken again and the old one is discarded. This protocol results in photos as shown in figure 3.1

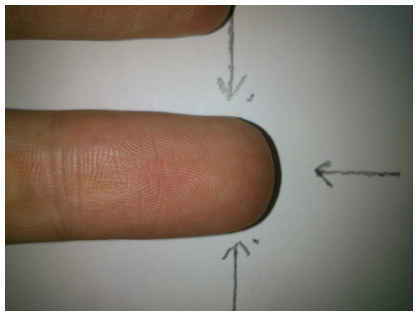


Figure 3.1: Fingerprint photo acquired using the protocol

[1] Due to a lack of time - among others due to late delivery of the fingerprint sensor - the first goal of 20 different persons was not met.

# 4 Enhancement Algorithm

From the related work as mentioned in section 2.2 it can be concluded that the enhancement algorithm needs at least these steps:

- Segmentation, separating the background from the fingerprint.
- Gray scale conversion, converting the color image to a gray scale image. This is needed because the VeriFinger software only accepts gray scale images and the enhancement algorithm works better on a gray scale image as a true color image, but with less computational power.
- Normalization, this is done to remove shadows and light and dark spots on the finger, providing a better quality image for the enhancement algorithm and making the image more consistent. Furthermore DSCF images do not show shadows either.
- Ridge orientation estimation, estimating the local orientation of the ridge valley pattern. This is needed for the enhancement algorithm to enhance only the patterns in the right direction.
- Enhancement, enhancing the image to make it better processable by the VeriFinger software. In this report, the enhancement algorithm aims on improving the image in such a way that the characteristics are the same as images from DSCF.

These steps will be described separately below. For every step the considerations on the approach will be described and the algorithm and test results will be displayed.

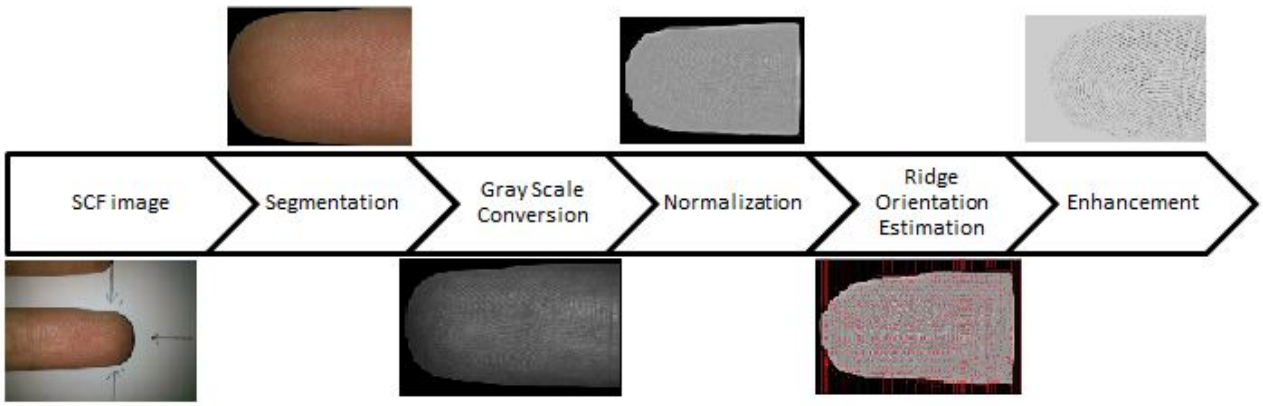
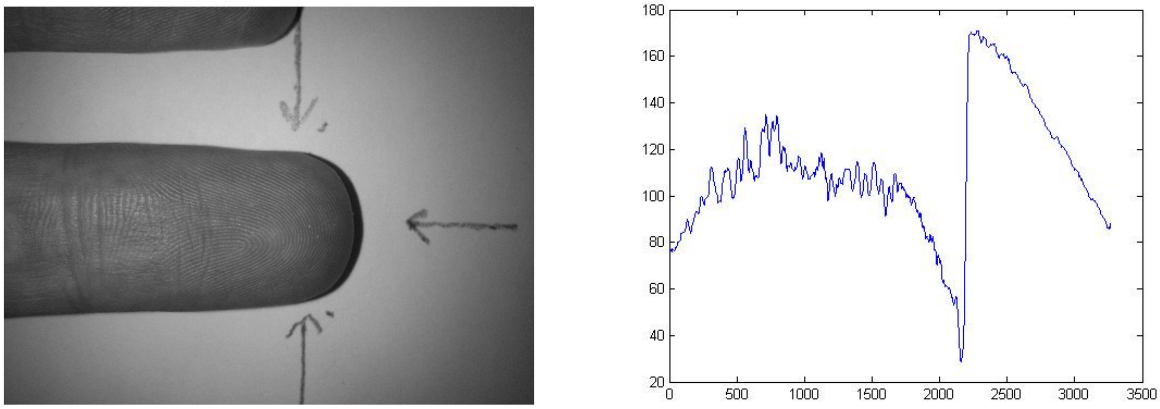



Figure 4.1: Overview of the total enhancement algorithm

## 4.1 Segmentation

In all related work, segmentation is the first step. Segmentation is not only crucial for removing the background - which could possibly result in false minutiae detections - it also reduces the size of the image resulting in lower computational complexity. The segmentation of the finger has been done before in various different ways, see section 2.2. Some of these mentioned algorithms can even handle complex backgrounds, but because the background is set fixed in the acquiring of the photos, this will not be needed. Instead it is chosen to separate the finger by detecting the finger start and end points on lines in the green channel. This approach has some similarities with the work by Stein et al. [22], but is different in the way that the segmentation is more precise. It separates the background completely from the fingerprint instead of just a rectangular box around the fingerprint. It also uses the green channel instead of the red channel, since the green channel shows sharper edges.

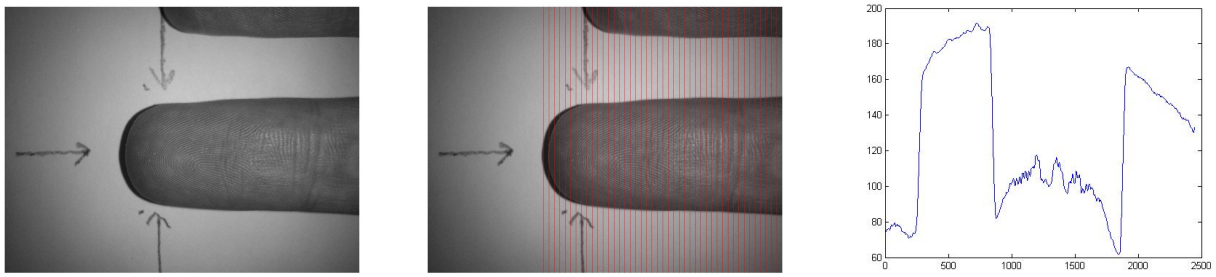
The photo is first cropped till only the fingertip is left in the image. This is done using the values of the green channel of the image. First a horizontal line along the center of the image is extracted and smoothed by averaging 30 pixels (figure 4.2b). This 30 pixel averaging is determined by trial and error. After the averaging, the minimum of this line is calculated, the index of this minimum is the place where the finger ends. If this index is over half the photo, then it is a left hand, otherwise it is a right hand. To equal the fingers, left hands are flipped over the vertical axis so all fingers are positioned in the same way.




(a) (b)

Figure 4.2: a) The green channel of the input image, b) the smoothed horizontal center line

Now the start of the finger in horizontal direction is determined, the fingerprint itself should be extracted. This is done by calculating the minima along vertical lines 50 pixels apart which are smoothed over 40 pixels, see figure 4.3b. This 40 pixel smoothing is also determined by trial and error.




(a) (b) (c)

Figure 4.3: a) The green channel of the flipped input image, b) the vertical lines along which measurements are taken, c) the 22th, as seen from the left, smoothed vertical line

The lowest values of the lines are calculated starting from the center of the line - this is pixel value 1224, since the pictures are 2448 pixels wide - and moving outwards. At first these minimum values are searched in an area of 524 pixels around the center value such that a possible second minimum from another finger is disregarded, see for example figure 4.4c. For these minima values, it is calculated when the first time occurs that the values on the line are 30 percent higher than the minimum to compensate for effects seen in figure 4.4b and figure 4.4d. This point is used as the edge of the finger. If no point is found, the search area for a minimum will be made bigger in steps of an extra 200 pixel from the center, in this way the first minimum from the center line is used to make sure the right finger is segmented and a possible second finger is removed. If two fingers are close together, a spike can be seen in the line, examples can be seen in figure 4.4e and figure 4.4f. These spikes are detected with the derivative, if both a high (>1.5) and low derivative (<-1.5) is found in the region 200 pixels around the detected point then the minimum closest to the center of the image is seen as the correct point.

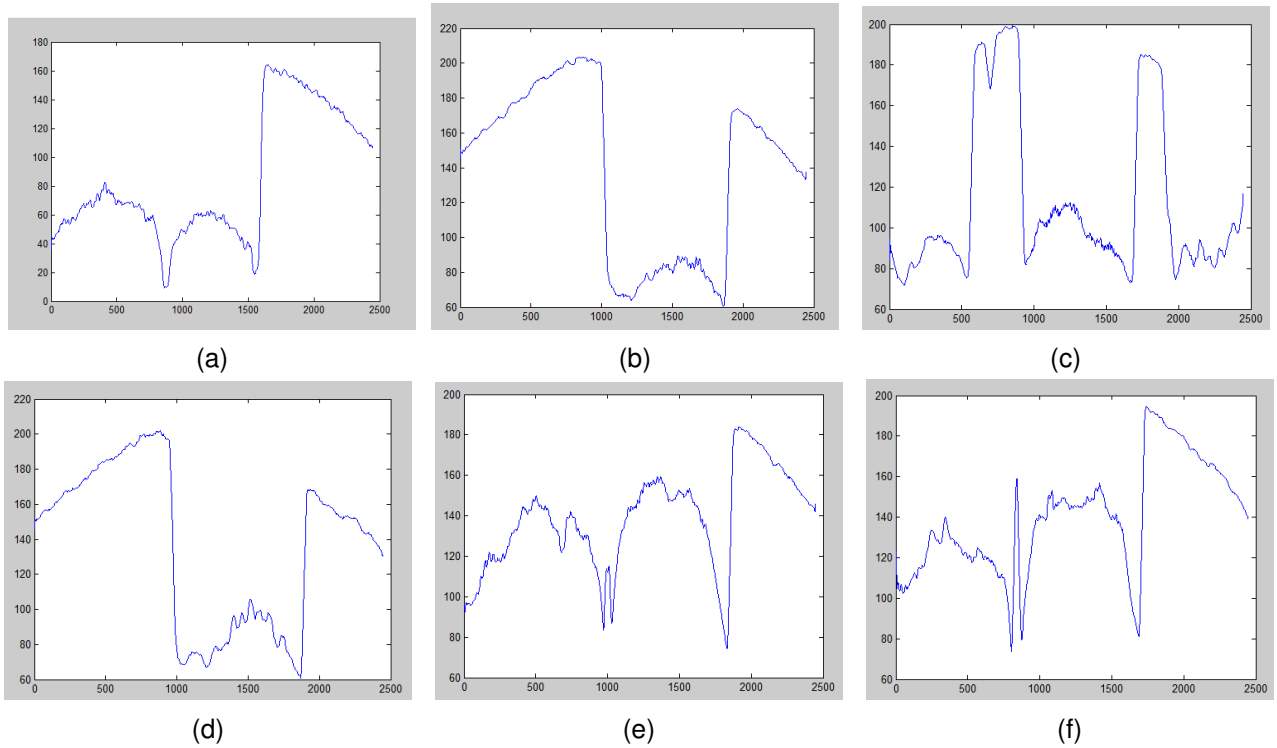



Figure 4.4: Possibilities for the smoothed lines to look like a) two fingers next to each other, b) only one finger in the image but with a flat area at the border, c) three fingers in the image with the white background in between fingers, d) one finger in the image but with a minimum which is not at the edge of the finger, e) two fingers next to each other with a very small strip of background in between, f) same as e but with a larger spike.

To compensate for incorrectly detected points, smoothing is used along the detected points of the upper part and lower part of the finger using a Robust Lowess linear fit method over 5 points [2]. By using this method the calculated points are smoothed and outliers are removed.

Now the finger is fit, the finger width is determined by getting the maximum distance of the finger width on the calculated lines. The region of interest (ROI) is then determined by the formula finger height $=$ finger width $*$ 1.6. This is an empirically determined value to remove the lower parts of the image which does not hold fingerprint information. The mentioned points and lines can be seen in figure 4.5.

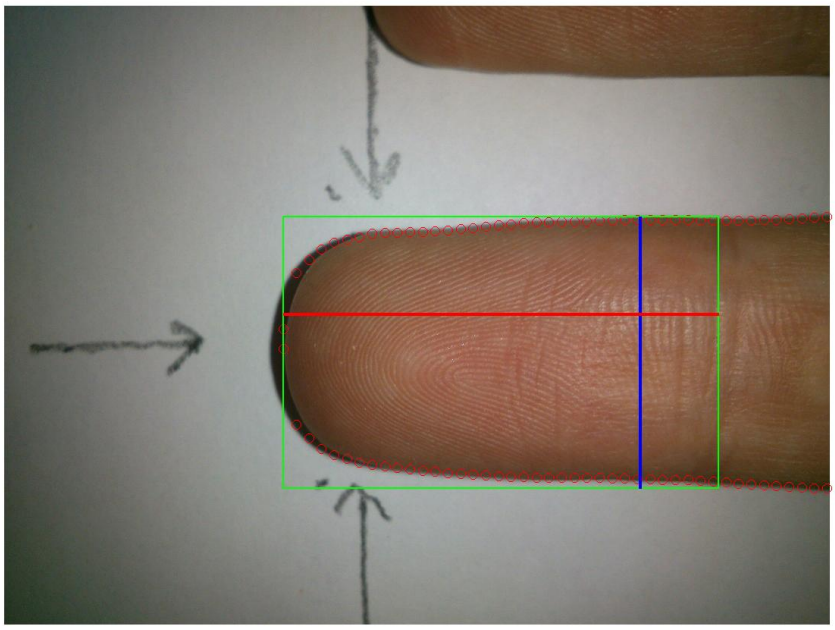
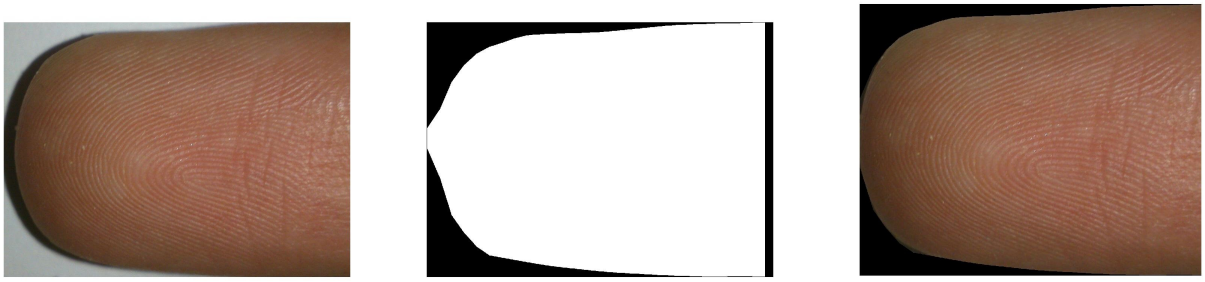


Figure 4.5: Red circles are the smoothed detected points on the edge of the finger. The green lines show the bounding box of the ROI to which the image is cropped. The blue line is the calculated finger width and the red one is the calculated height.

Since all finger dimensions are now known, the image can be cropped on its width and height, this can be seen in figure 4.6a. By interpolating the detected points of the fingers edge a background mask is made, see figure 4.6b. Applying this mask results in figure 4.6c.



(a) (b) (c)

Figure 4.6: a) The cropped input image, b) the segmentation mask, c) cropped image with segmentation mask applied

The algorithm described was able to segment all 40 images of the test set in a correct way.

In the segmentation process, the lines of the top edge and bottom edge of the finger are calculated. With this information, the finger can be easily adjusted for a rotation of the finger in the x,y-plane. This is done by calculating the angle of the top edge line of the finger and the angle of the bottom edge line of the finger as seen in equation 4.1. If both have an angle in the same direction, the smallest angle of the two is assumed to be the rotation. The finger rotation is then compensated using this rotation. An example of the lines that are calculated for the rotation can be seen in figure 4.7. In the example image, the angle of the top edge line is in another direction then the angle of the bottom edge line. This image will thus not be

compensated for rotation.

$$r = \arctan(\frac{mean(f')}{S}) \tag{4.1}$$

where:
r = rotation angle in degrees
f' = derivative of top or bottom edge line along 15 points
S = step size between the calculated points on the edge line, 50 in this case.

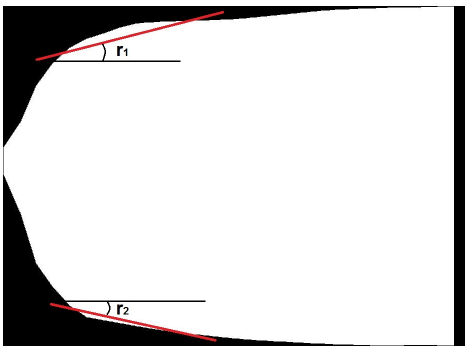


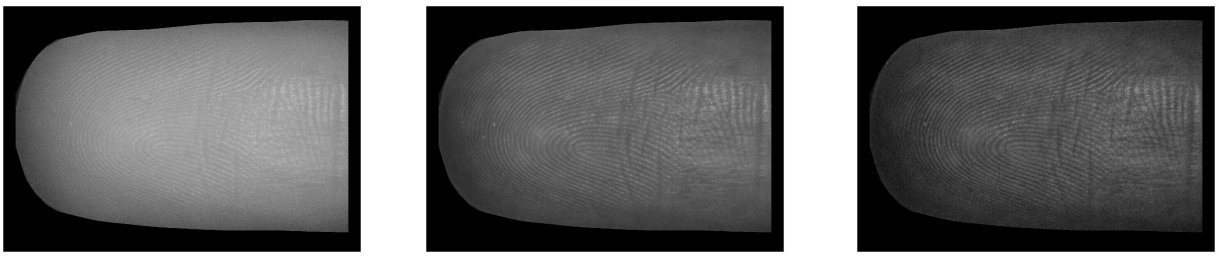

Figure 4.7: Lines for calculating the rotation of the finger

## 4.2 Gray Scale Conversion

Gray scale conversion is done in all related literature to convert the 3-channel true color image to a 1-channel gray scale image. This reduces the size of the image while keeping the important information. A gray scale image is also needed in the next processing steps and by the VeriFinger software. According to the literature a blue light source gives higher ridge valley contrast to an image [12, 20, 23]. Since a blue light source is not available when using a smartphone, the LED is turned on when capturing the photo. Because the photo is captured in RGB format, the red, blue and green channels can be extracted. Looking at these channels in figure 4.8 it can be seen that the blue channel indeed gives better ridge valley contrast than the red channel, it is however also visible that the green channel performs at least as good as the blue one. This difference can also clearly be seen in figure 4.19h, where the green and blue lines have a much more clear pattern. Since this is the case, it is decided to discard the red channel and create the gray scale image by averaging the blue and green channels according to equation 4.2. It is decided to do this averaging instead of just using the blue channel because by averaging the noise will be reduced. This results in the image seen in figure 4.9.

$$g(x,y) = \frac{f_g(x,y) + f_b(x,y)}{2} \tag{4.2}$$

where
$f_g(x,y)$ : the green channel of the original image
$f_b(x,y)$ : the blue channel of the original image



(a) (b) (c)

Figure 4.8: a) The red channel of the segmented image, b) the green channel of the segmented image c) the blue channel of the segmented image

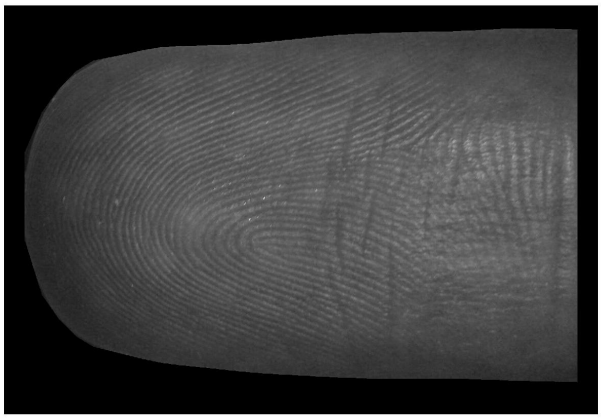


Figure 4.9: Segmented image converted to gray scale

## 4.3 Normalization

After segmentation and the gray scale conversion, the image is normalized. This is needed because the convex shape of the finger causes a shadow on the finger, see figure 4.11a. Furthermore, normalizing the image by the mean makes the further processing steps much easier. Hiew et al. [7] describe a normalization in which the standard deviation (std) is also used in the calculation. Their normalization algorithm can be found in equation 2.1. While a normalization using the std will make the image more consistent, it also throws away crucial information. A low std can mean bad quality or possibly a minutiae point. Certainly if the std is calculated using blocks instead of on lines in the direction of the ridge pattern. It is therefore chosen to only use the mean in the normalization calculation. The mean of the image is calculated at the middle of 30x30 sized blocks and is linearly interpolated between these blocks. This is done because a large block size is needed to make sure that local deviations from the mean, like ridge valley patterns, still stay intact. It is chosen to calculate the mean only at certain points, because calculating the mean on every point with a 30x30 block size is very computationally expensive, while the method with interpolation is not.

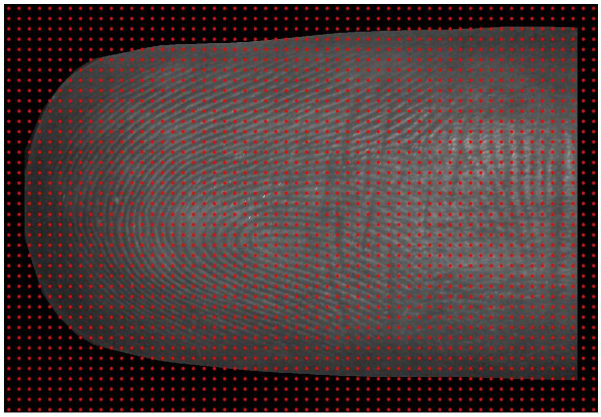


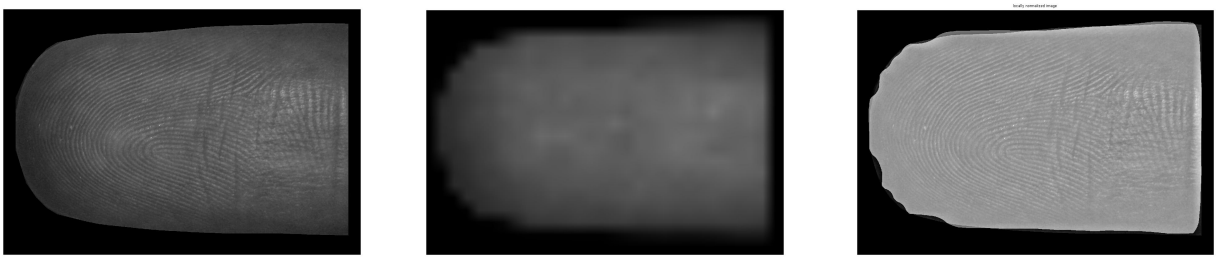
Figure 4.10: Points where the mean is calculated

After the mean mask is made. The image can be normalized, this is done by adding 150 to the gray image and subtracting the mean mask, see equation 4.3.

$$g(x,y) = f(x,y) + 150 - m_f(x,y) \tag{4.3}$$

where
$f(x,y)$ : the original image
$m_f(x,y)$ : the local mean of the image calculated with a 30x30 block size

Running this algorithm results in figure 4.11c.

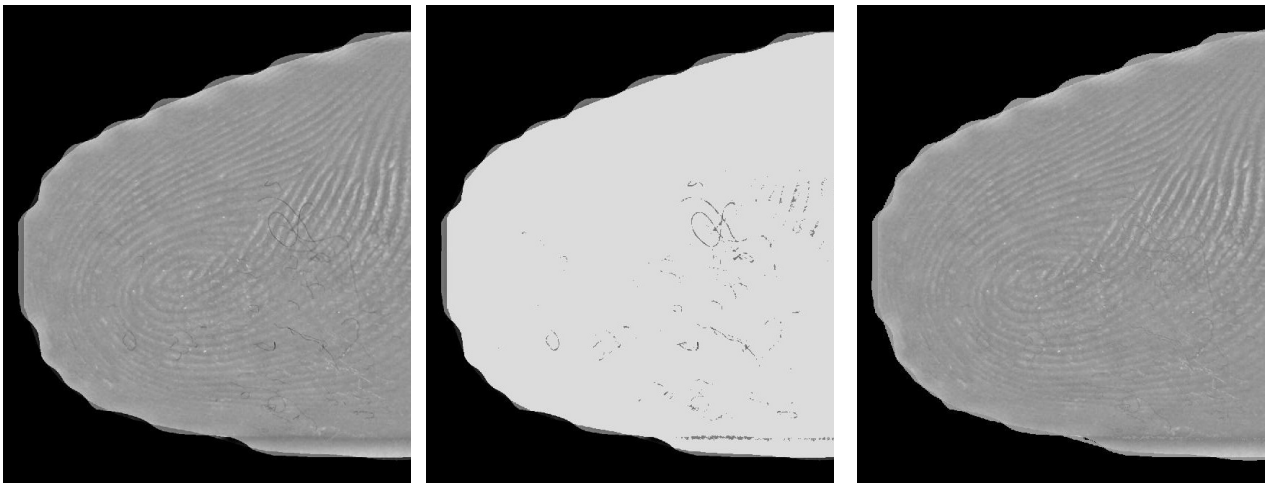


(a) (b) (c)

Figure 4.11: a) The segmented image, b) the calculated mean c) image after applying the normalization mask

As can be seen, the image is much more consistent and does not show any shadows any more.

Another advantage of the normalization is that dirty spots on the image like hairs or mud become more clearly visible and can easily be removed. The dirty spots are visible as black lines in the image as can be seen in figure 4.12a. By applying a threshold on the image, which is empirically determined to be all pixels with values below 130, the dirty spots can be removed. This removal is done by setting all these "dirty" pixels to the mean value of 150.



(a) (b) (c)

Figure 4.12: a) The dirty image, b) the detected dirty spots c) image after removing the dirty spots

## 4.4 Ridge Orientation Estimation

All enhancement algorithms need some sort of ridge orientation estimation. This is needed to be able to enhance the ridge valley pattern instead of the noise. Lee et al. [14] already mentioned that a simple least square methods cannot be used due to noise. They propose an iterative least square method which removes the outliers for a more reliable orientation estimation. Other work suggests an orientation estimation in the frequency domain [7]. While both methods show promising results, a more simple approach is suggested based on the std of the image. But first it is shown that just calculating the gradient of the image after averaging with a block size of 7x7 does not produce satisfactory results, see figure 4.13. This is because there is to much noise in the image, differences on ridges are about the same size as between ridges.

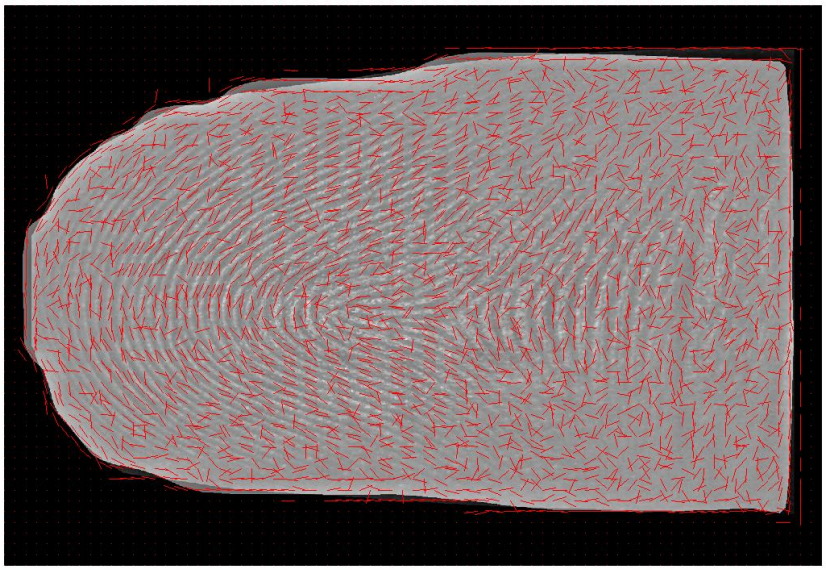


Figure 4.13: Orientation estimation using the gradient

To solve this problem a different approach is taken. This approach is based on the fact that on lines along the ridges the pixel values fluctuate much less than perpendicular to the ridges, see figure 4.14. This means that the standard deviation is larger in the direction perpendicular to the ridges orientation. This information is used to determine the orientation of the ridge/valley pattern.

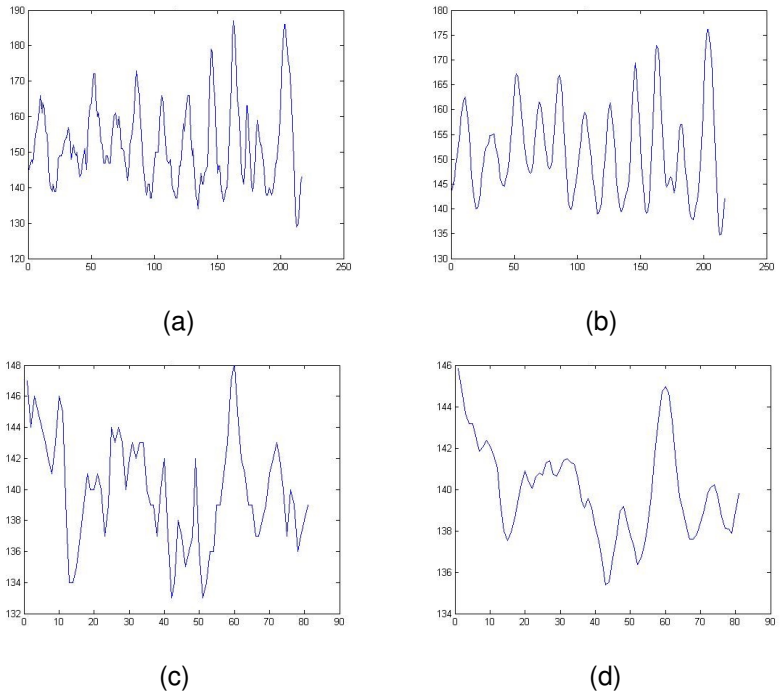



Figure 4.14: Image values along some lines. a) line perpendicular to the ridges, b) same line, but smoothed, c) line in the same direction as the ridges, d) same line but smoothed. It should be noted that the values on the line perpendicular to the ridges fluctuate much more than on the line in the same direction as the ridges

First of all the standard deviation along horizontal and vertical lines is determined in a 5x5 block size smoothed image, where for the horizontal standard deviation calculation an empirically determined block size of 9x3 is used and for the vertical orientation a block size of 3x9 is used. Then the image is rotated 45 degrees and the same calculation is done. This results in 4 different standard deviation matrices calculated for the lines in 0, 45, 90 and 135 degrees orientations. To get a rotation matrix, it is needed to see which values of the standard deviation are the largest at which points. Comparing these values and extracting the maximum results in the matrix seen in figure 4.15.

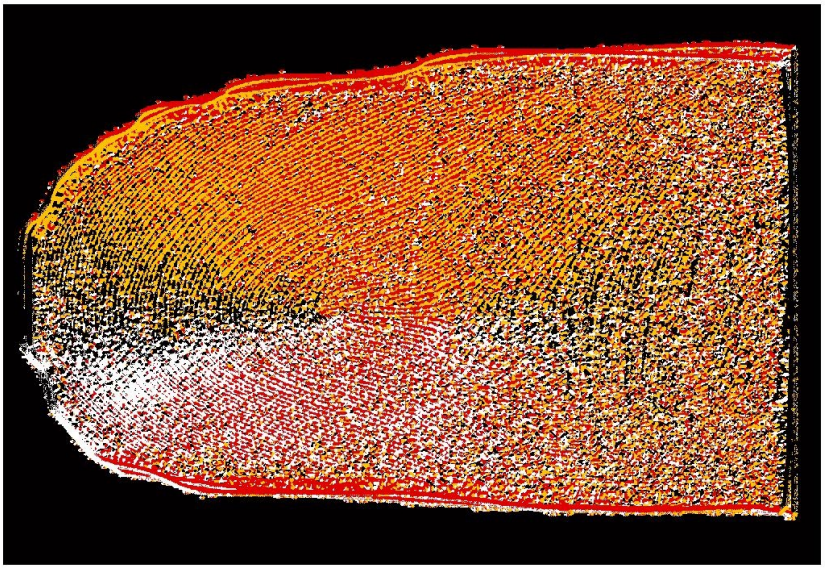


Figure 4.15: The orientation image with the highest standard deviations, black is for 0 degrees, white is 45 degrees, red is 90 degrees and yellow is 135 degrees.

As can be seen the result is quite crude. It can however also be seen that most of the orientations are correct. To make a more exact orientation matrix, the most common value (0,45,90 or 135 degrees) in a block of 30x30 is taken and averaged with the second most common value in the block based on the number of occurrences of that orientation in the block. So if a block has 30% of 0 degrees orientation values and 70% of 45 degrees, the final orientation matrix will show an orientation of $0.3*0+0.7*45 = 31.5$ degrees. Using this averaging scheme results in the final orientation matrix that, as can be seen in figure 4.16, looks much better now.

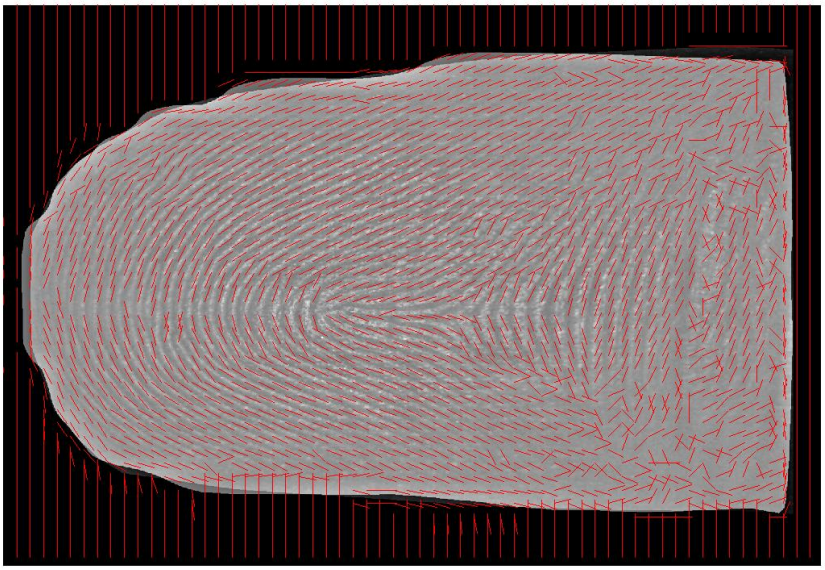


Figure 4.16: The orientation matrix on top of the input image.

Also for more difficult fingerprint images with high curvature, the algorithm shows good results, see figure 4.17.

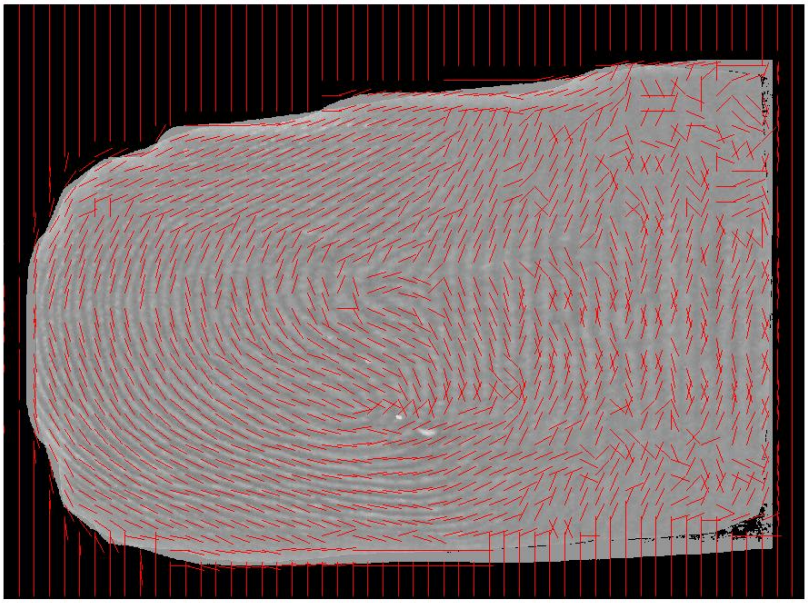


Figure 4.17: The orientation matrix on top of the input image for a highly curved fingerprint pattern.

## 4.5 Enhancement

Enhancement is the final step in this process. While most related work now describes steps involving gabor filters or short time fourier transforms (STFT). It is chosen not to employ these here. This is done because the VeriFinger software for enhancing and extracting fingerprints from DSCF already has the capabilities to enhance the images in the same way as gabor filters or STFT does. It is therefore completely unnecessary in trying to improve these kind of filters that are already in development for a long time. Instead it is chosen to enhance the fingerprint image in such a way that it shows more similarities with DSCF images.

The ridge/valley pattern of DSCF images are such that the ridges and valleys are distinctly visible, but with gaps in the pattern. Multiple trials are done in enhancing the images. First of all an attempt is made to map the ridges to a black value and the valleys to a white value. The algorithm is described and the results are displayed. Summarizing what went well and what not, the enhancement algorithm is altered to show more similarities with DSCF and the result of that algorithm is described as well. It is furthermore shown that providing the VeriFinger software with the input images for the enhancement algorithm has worse results than supplying the software with the enhanced images. But first we will take a look at DSCF images in more detail.

### 4.5.1 DSCF images in detail

Examples of DSCF can be seen in figure 1.1. To make it more clear what these images look like at pixel level, a 2D section perpendicular to the ridge orientation is extracted. The extracted lines can be found in figure 4.19, these correspond to the the images in figure 4.18. The lines are made by plotting the pixel values of the sample images on a line from top to bottom in the center of the shown image.

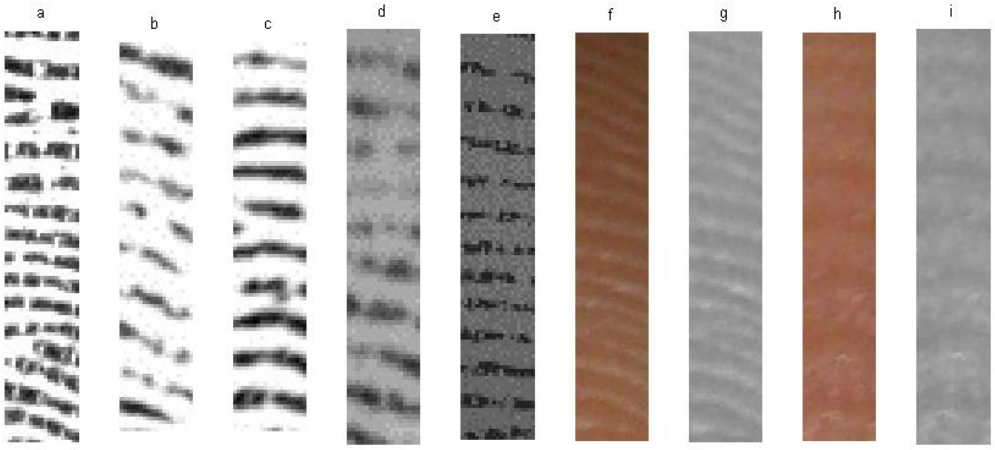



Figure 4.18: Samples from DSCF and SCF images. e - f correspond to e - f from figure 1.1. h is from the same image as f but f is taken from the edge and h is more towards the core. g and i are the normalized images of f and h.

From the DSCF image a few things can be noted. There is an almost perfect basis level, the white background of DSCF images. This basis level can most clearly be seen in figure 4.19a, 4.19b and 4.19c at pixel value 250. From this level, peaks can be seen downwards, with the more probable ridges having bigger peaks. In figure 4.19d and 4.19e we see more noise at the basis level. From these images it becomes clear that a steady basis level is needed, or if there is noise at this basis level, a larger peak swing is

needed.

Looking at the SCF lines in figure 4.19f and 4.19h it first of all becomes clear that the red channel of the image shows bad quality, this is why the choice is made to remove this channel for the gray scale conversion. Furthermore the line in figure 4.19f shows an upward moving mean due to the shadow. Removing this shadow is done with the normalization step, the improvement for this can be seen in figure 4.19g and 4.19i. While due to this improvement the line now shows a clear oscillating pattern, there is no basis level established as seen in the DSCF lines and there are also few distinct peaks in figure 4.19h.

To improve the images two different approaches are taken. First an approach with mapping the values to either a black or a white value is tried. In this way a clear basis level will be obtained and the peaks will be distinct. As a second approach, a basis level will also be used, but the peaks will be enhanced according to their strength instead of mapping to a black value.

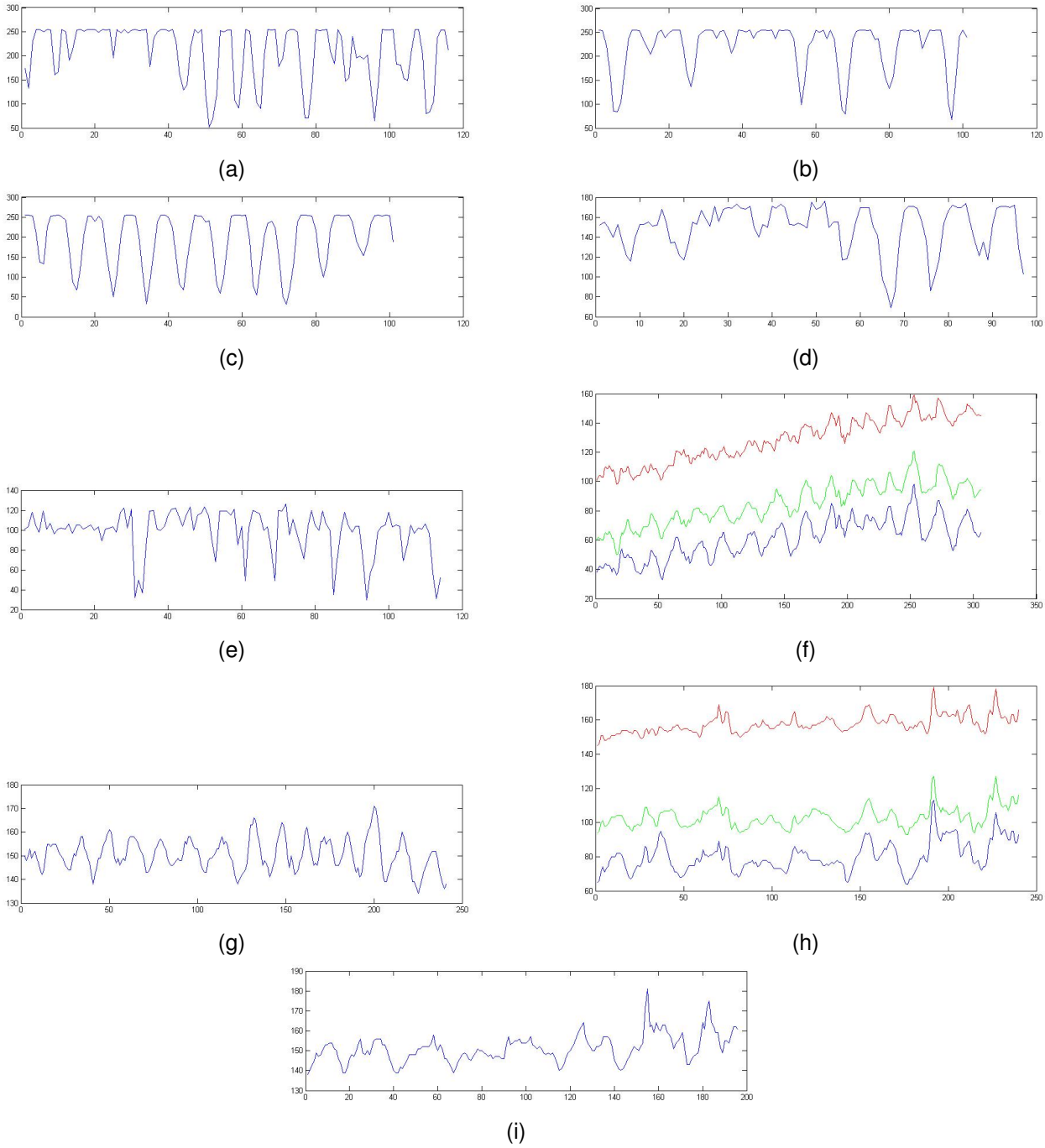



Figure 4.19: Lines corresponding with the images from figure 4.18. For the images f and h, the red green and blue channel components are plotted seperately.

### 4.5.2 First enhancement algorithm

As said before, the first enhancement algorithm will map the pixel values to white or black. To do this, a measure for a pixel belonging to a ridge or valley should be set. We know from figure 4.19g and figure 4.19i that an oscillating pattern can be found on lines in the direction perpendicular to ridge/valley pattern. Since this pattern is a bit erratic, it is chosen to use the smoothed image which is already calculated during the ridge orientation with a 5x5 block size smoothing filter. The measure for a ridge or valley could then be obtained by calculating the mean along those lines. To reduce the risk of mapping towards the wrong

value, the standard deviation along the line is also measured and only pixels which are larger or smaller than the mean +/- 0.4 * standard deviation are mapped. This process will be described in more detail below.

The orientation information is used to enhance the image. First of all the mean and standard deviation along a line perpendicular to the orientation is measured. The mean and standard deviation are measured along the line with an empirically determined 29 points. To keep the cpu power low, not all exact orientations are used. There are four different orientations measured, 0, 45, 90 and 135 degrees. With this information about the mean and standard deviation the image can be enhanced. An example of the calculated lines can be observed in figure 4.20.

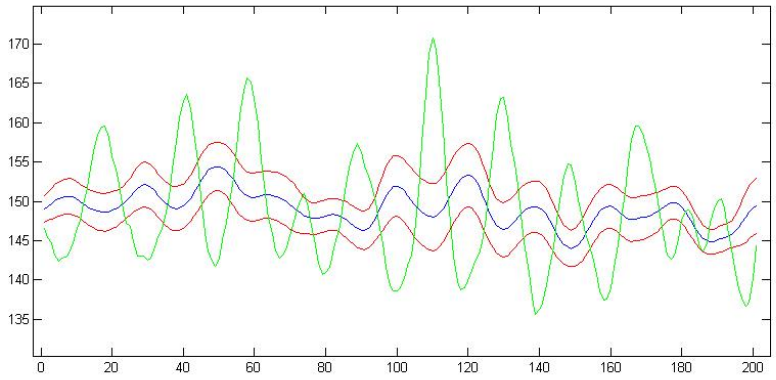



Figure 4.20: Line of the image (g), mean of the line over 29 points (b), mean +/- 0.4*standard deviation of the line over 29 points(r) for a line perpendicular to the ridge/valley pattern

The points of the line above the upper red line are mapped to an intensity value of 220, while points below the lower red line are mapped to 20. Doing this for all four orientations and using only the right orientation for each block results in figure 4.21.

Figure 4.21: Image after first enhancement

As can be seen, most of the pixels are mapped to 220 or 20 intensity values. For the pixels that are not

mapped yet, a next enhancement step is required. This enhancement step is the same as before, but now the pixels of the closest other orientation are used to map the pixels. In this way slight variations in the direction calculation are compensated for and dirty spots in the exact direction are circumvented by taking the values in this other direction. Using this enhancement step, the image in figure 4.22 is created.

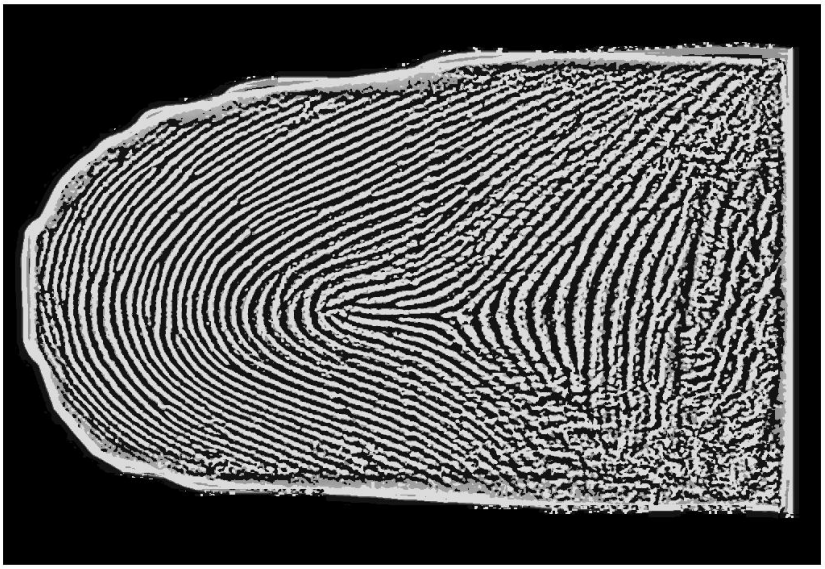
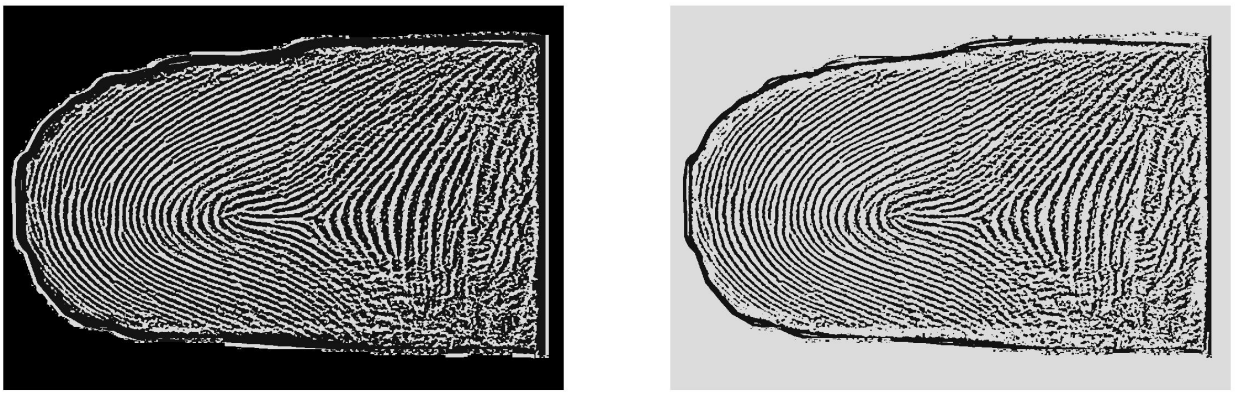


Figure 4.22: Image after second enhancement

At this point it can be chosen to map the remaining pixels all towards black or white. With mapping towards white resulting in connecting lines that shouldn't be connected and with mapping towards black resulting in disconnecting lines. Since fingerprint enhancement algorithms can better handle discontinuations in lines (this is also common in fingerprints acquired using sensor systems or ink), it is a better choice to map towards black. Since normal fingerprint images have black lines on ridges and white lines on valleys and this SCF has it vice versa, the image has to be inverted. This results in the images seen in figure 4.23.



(a) (b)

Figure 4.23: a) image mapped towards white, b) image mapped towards black

It can however also be chosen to use the line calculation again on the enhanced image. This time the calculation is performed without using the standard deviation such that al pixels above the mean are mapped to 220 and below to 20. This means that only the mean has to be computed on the enhanced

image across the orientation lines. The results are slightly better than those from direct mapping, but the computation costs are also higher. The resulting image can be found in figure 4.24

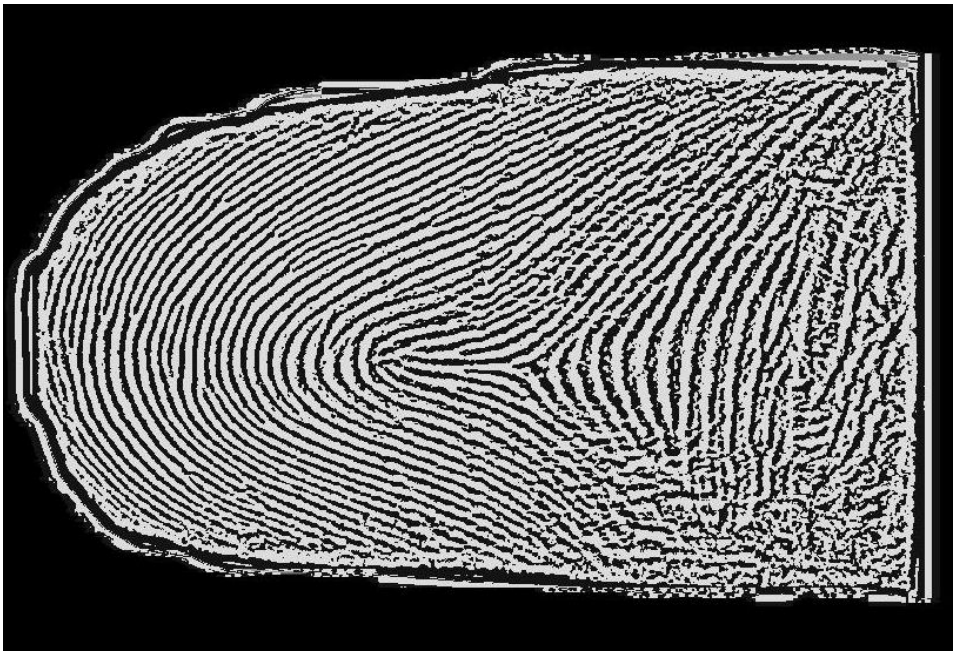


Figure 4.24: Image after second enhancement

Since the verifinger software uses filters which rely on the ridge frequency the images are scaled to a ppi value of 500 by knowing the distance from the camera to the finger and the resolution of the camera. In this way, the ridge frequency corresponds to the images gathered from the dedicated sensor which are also 500 ppi. Furthermore the images are rotated to an upright position, this is also done to make the images from both sensors the same.

### 4.5.3 Results of first enhancement on the test set

The enhanced images are saved and split into three groups. Group 1 contains 16 fingerprint templates of 16 different fingers, group 2 contains 16 images of the same fingers and should thus be matched, while group 3 contains 8 images of other fingerprints that should not be matched.

The software to extract minutiae points and do the comparison is the Verifinger 7.0/MegaMatcher 5.0 Identification Technology Algorithm Demo [5]. The options for the software are set as:

- Minimum minutia count: 10
- Quality Treshold: 90
- Template size: Large
- Extracted ridge counts: Eight neighbors with indexes
- Return processed image: true
- Fast Extraction: false
- FAR: $0.01\%$
- Maximal rotation: 59
- Matching speed: Low

- Maximal Result Count: 1000
- First Result Only: False

Using these options, the results below are returned for the black and white mapped images using the algorithm for calculation of the mean along lines twice (the last mentioned algorithm):

Table 4.1: The results for making a template of the images from group 1

| | enrolled | failed | reason |
|---|---|---|---|
| Group 1 (enrolled) | 15 | 1 | bad quality fingerprint |

Table 4.2: The results of the identification process using the image from group 2

| Group 2 (identify) | image number | identified (True/False) | reason | score |
|---|---|---|---|---|
| | 17 | T | | 165 |
| | 18 | T | | 83 |
| | 19 | T | | 425 |
| | 20 | T | | 406 |
| | 21 | T | | 612 |
| | 22 | T | | 507 |
| | 23 | T | | 426 |
| | 24 | T | | 466 |
| | 33 | T | | 621 |
| | 34 | T | | 299 |
| | 35 | F | Corresponding fingerprint not enrolled | |
| | 36 | T | | 303 |
| | 37 | T | | 82 |
| | 38 | T | | 163 |
| | 39 | T | | 275 |
| | 40 | T | | 115 |
| **Average** | | | | **329.8667** |

As can be seen in the tables, the software shows perfect results, except for one image which is not enrolled. This is however a good thing, since the image - figure 4.25 - is completely out of focus.

Table 4.3: The results of the images from group 3 that should not be identified

| | image number | identified (True/False) |
|---|---|---|
| Group 3 (don't identify) | 9 | F |
| | 10 | F |
| | 11 | F |
| | 12 | F |
| | 13 | F |
| | 14 | F |
| | 15 | F |
| | 16 | F |

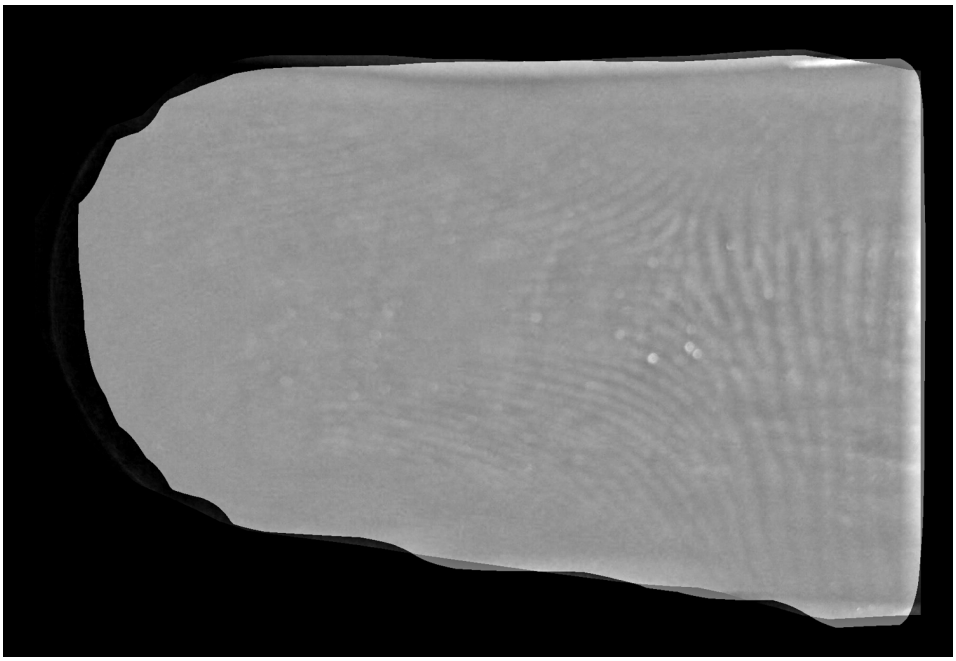


Figure 4.25: Not enrolled image

That does however not mean that all defocused images show bad results, as can be seen in figure 4.26.

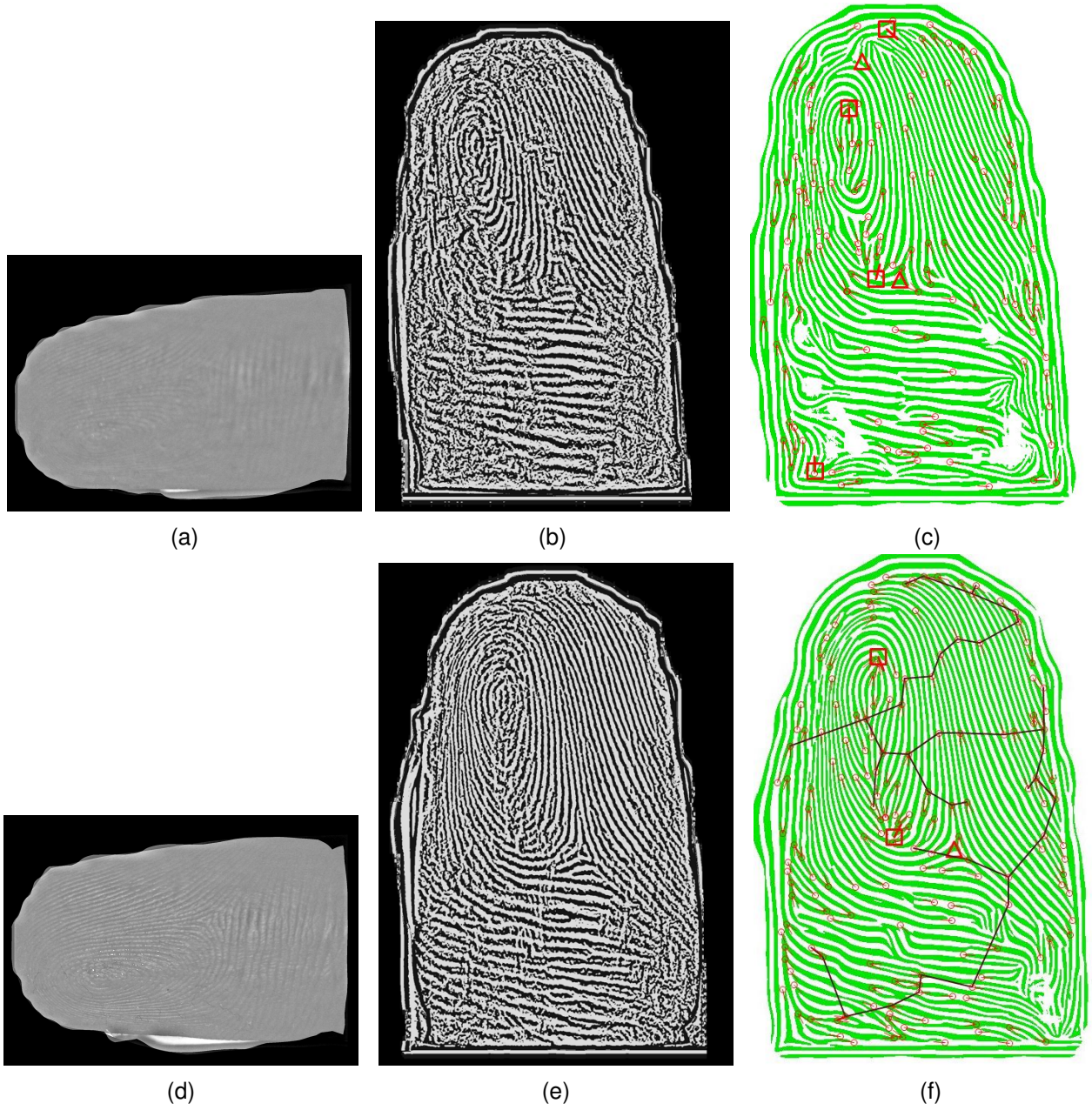



Figure 4.26: Processing and comparison of defocused fingerprint images from the same finger. Top row shows the normalized, enhanced and processed image template. While the bottom row shows the same for the comparison image.

### 4.5.4 Results of the test set without enhancing

To show that the enhancement process does indeed have a positive effect on the image quality, a comparison test is done using the normalized images that are rotated and scaled to the same size as the enhanced set. The only processing done on these images is segmentation, normalization and scaling. The results can be found in the tables below.
As can be seen many more fingerprint images are rejected because of bad quality. The fingerprints that are identified however show a higher score. This is probably mainly because the enhanced images have artefacts around the edges resulting in false minutiae points, while the non enhanced images do not get these artefacts.

Table 4.4: The results for making a template of the images from group 1 for the only normalized images

| | enrolled | failed | reason |
|---|---|---|---|
| Group 1 (enrolled) | 10 | 6 | bad quality fingerprint |

Table 4.5: The results of the identification process using the image from group 2 for the only normalized images

| Group 2 (identify) | image number | identified (True/False) | reason | score |
|---|---|---|---|---|
| | 17 | T | | 400 |
| | 18 | T | | 106 |
| | 19 | T | | 1717 |
| | 20 | T | | 1071 |
| | 21 | T | | 1300 |
| | 22 | T | | 736 |
| | 23 | T | | 766 |
| | 24 | T | | 1005 |
| | 33 | F | Corresponding fingerprint not enrolled and bad fingerprint | |
| | 34 | F | Not identified | |
| | 35 | F | Corresponding fingerprint not enrolled | |
| | 36 | F | Corresponding fingerprint not enrolled and bad fingerprint | |
| | 37 | F | Corresponding fingerprint not enrolled | |
| | 38 | F | bad fingerprint | |
| | 39 | F | Corresponding fingerprint not enrolledt and bad fingerprint | |
| | 40 | F | Corresponding fingerprint not enrolled and bad fingerprint | |
| **Average** | | | | **887.625** |

Table 4.6: The results of the images from group 3 that should not be identified for the only normalized images

| | image number | identified (True/False) |
|---|---|---|
| Group 3 (don't identify) | 9 | F |
| | 10 | F |
| | 11 | F |
| | 12 | F |
| | 13 | F |
| | 14 | F |
| | 15 | F |
| | 16 | F |

### 4.5.5 Second enhancement algorithm

Looking at the results, it becomes clear that the Verifinger software performs great in enhancing fingerprint images. The enhancement algorithm however has some problems in low contrast areas. This is because the algorithm has to make a choice to map to either white or black. This mapping might be unnecessary since the DSCF also shows a grayscale and the VeriFinger algorithm already does this better than we can. Furthermore there is no difference in strength between the ridges, while this can be seen in the DSCF images. To account for these problems, a second enhancement algorithm is made. This algorithm will also make a basis level as can be seen in DSCF images, but will enhance the ridges according to their relative strength. Such behaviour can also be seen in figure 4.19.

The algorithm still maintains the same orientation and mean calculation but it won't calculate the standard deviation and has been adapted to enhance the ridge patterns that are already well visible. To do this, the enhanced image is calculated in the four different directions according to:

$$g_d(x,y) = f(x,y) - 24 + \Re((f(x,y) - m_{fd}(x,y))^{1.5}) \tag{4.4}$$

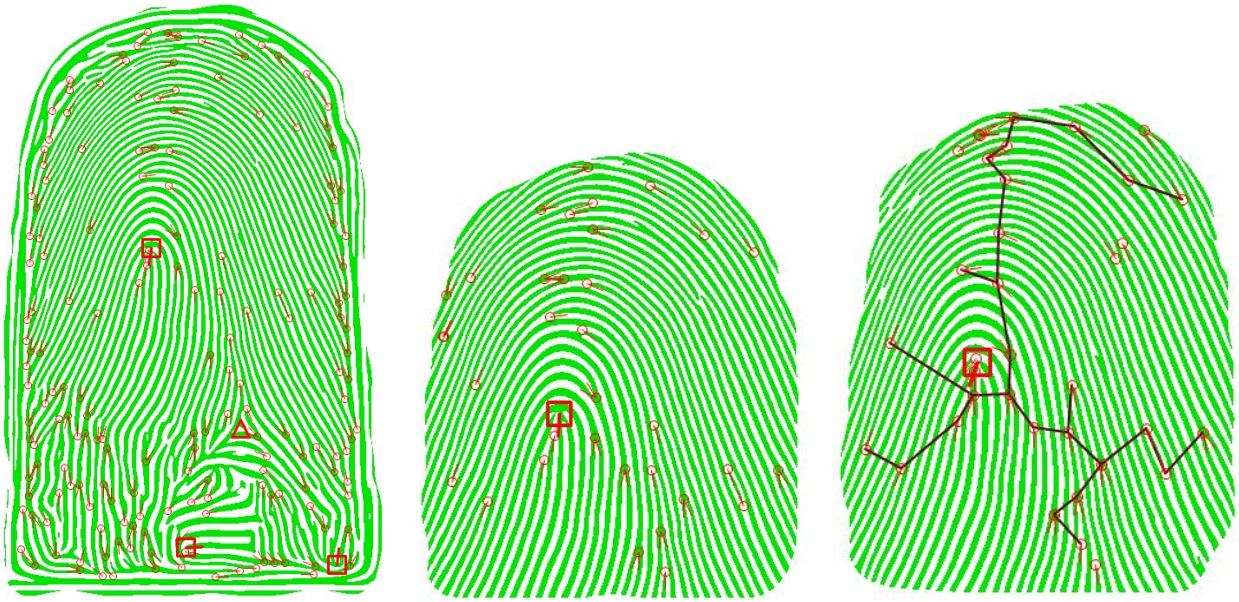
where
$f(x,y)$ : the original gray scale image which is smoothed with a 5x5 smoothing filter
$m_{fd}(x,y)$ : the local mean of the image in one of the four orientations
$\Re()$ : the real part of operator

The $-24$ is used to get the image mean to be around 122 instead of 150 as first calculated during normalization, this is done so the background is more white and enhancement is less likely to be outside the boundaries of 0-255. It is however recommended to this already during normalization. Running the algorithm will create four images, which are then stitched together according to the orientation estimation. It should be noted that only the pixels above the local mean are amplified, because the ones below the mean become purely imaginary when using this non integer power. It could be chosen to map the pixels that are under the local mean to the basis level, but this resulted in slightly worse results than not doing this.

As can be seen in figure 4.20, the calculated mean line is not very steady. It has small downward peaks when upward peaks occur in the image line and vice versa. This is because the mean line is calculated over 29 pixels, which is about 1.5 times the ridge frequency. While this behaviour seems bad, it makes the enhancement actually better due to a larger difference between the mean line and the image line on the top of peaks and valleys.

Now the image is created, the complement is taken to get an image with black ridges and white valleys as normal with images from dedicated sensors. From the test set it is noted that many false minutiae points are created near the border of the fingerprint, see figure 4.27a. These points lower the overall quality of the fingerprint and makes the chance of false fingerprint acceptance higher. To remove this edge of the fingerprint an erosion on the fingerprint border is done with a kernel of 140 by 140 pixels. It is also observed that the lower part of the fingerprint has a lower quality than the rest of the fingerprint - see figure 4.27a - and is normally invisible in DSCF images. It is therefore chosen to remove the lower 35% of the fingerprint images. Another source of false minutiae points is a whiter background around the fingerprint than the basis line of the fingerprint. To eliminate this effect, the background is set to the same pixel value as the basis line of the fingerprint. Finally all pixel values are lifted by 70, this is done only for human perception. Using this procedure, images that look like figure 4.29c are produced. While a line extracted from the image as seen in figure 4.28 now has close resemblance to the lines from some DSCF images as found in figure 4.19d and figure 4.19e. Far fewer false minutiae points can now be observed in figure 4.27b. In figure 4.27c another image of the same finger in 4.27b can be seen. The minutiae points that are the same in both images are connected by a line. As can be seen, most of the minutiae points match,

giving rise to a high similarity score.



(a) (b) (c)

Figure 4.27: Images further enhanced and minutiea points detected by the VeriFinger software. a) image as enhanced using the first algorithm, many false minutiae points are detected. b) image as enhanced using the new alogithm c) other enhanced image of the same finger as seen in b, the lines indicate the simillar points.

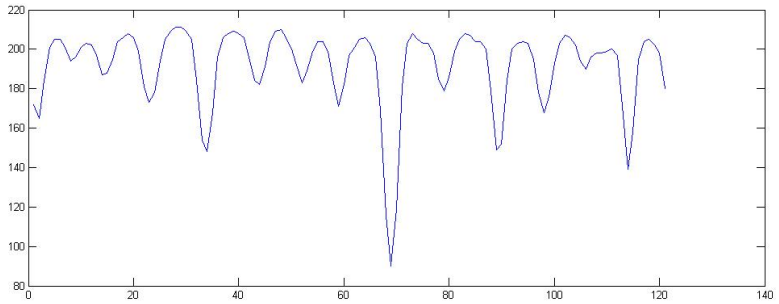



Figure 4.28: Line extracted from the enhanced image perpendicular to the ridge orientation.

(a) (b) (c)

Figure 4.29: a) image from the Cross Match Verifier 300 dedicated sensor, image taken from the verifinger sample database [5] , b) image from the FX2000 optical sensor, image taken from the FVC2002 database 2 [1] c) enhanced image taken with smartphone camera

Comparing the image from figure 4.29c with the old enhancement algorithm in figure 4.24 it might seem that the contrast is lower. While this is certainly true and therefore looks like a bad quality image to a human, the computer readability actually increases. To show that this type of enhancement indeed shows better results, the images are tested again. These results can be found in de tables below.

Table 4.7: The results for making a template of the images from group 1 for images enhanced with the second algorithm

| | enrolled | failed | reason |
|---|---|---|---|
| Group 1 (enrolled) | 15 | 1 | bad quality fingerprint |

As can be seen in the tables, all fingerprints are recognized except for the defocused one. It was even possible to set the fingerprint quality option to 100 out of 100 (instead of 90 with the first enhancement). The overall results are much better in comparison to both other tests. The scores given by the VeriFinger software improved from an average 329 to 3165. This means the results are improved while the computational power of the algorithm is reduced. It can therefore be concluded that it is important to let the images look more like images from dedicated sensors since this greatly improves the comparison score of the fingerprint recognition software.

Table 4.8: The results of the identification process using the image from group 2 for images enhanced with the second algorithm

| Group 2 (identify) | image number | identified (True/False) | reason | score |
|---|---|---|---|---|
| | 17 | T | | 2133 |
| | 18 | T | | 185 |
| | 19 | T | | 5463 |
| | 20 | T | | 2801 |
| | 21 | T | | 6686 |
| | 22 | T | | 3281 |
| | 23 | T | | 3871 |
| | 24 | T | | 5130 |
| | 33 | T | | 6461 |
| | 34 | T | | 2525 |
| | 35 | F | Corresponding fingerprint not enrolled | |
| | 36 | T | | 4389 |
| | 37 | T | | 129 |
| | 38 | T | | 719 |
| | 39 | T | | 3422 |
| | 40 | T | | 273 |
| **Average** | | | | **3164.5** |

Table 4.9: The results of the images from group 3 that should not be identified for images enhanced with the second algorithm

| | image number | identified (True/False) |
|---|---|---|
| Group 3 (don't identify) | 9 | F |
| | 10 | F |
| | 11 | F |
| | 12 | F |
| | 13 | F |
| | 14 | F |
| | 15 | F |
| | 16 | F |

# 5 Results

To provide reliable results, the best algorithm is used on all SCF images in the fingerprint image database, which consists of 176 SCF and 175 DSCF images. These enhanced SCF images are matched against the other SCF images and against DSCF images. In this way, we will be able to see whether it is possible to cross match SCF and DSCF images and if SCF images form a reliable fingerprint template source. Furthermore DSCF images are compared to the other DSCF images, so we will be able to compare the performance of SCF and DSCF images.

Both the SCF and DSCF images will be further enhanced by the VeriFinger 6.2 MegaMatcher software. This software will also extract the minutiae points and make a template for each image. These templates are then matched against the other templates in the database by the VeriFinger software, which results in similarity scores [1]. These similarity scores determine if the two images are from the same finger. By lowering the threshold of when a certain score is a match, the false rejection rate (FRR) decreases but the false acceptance rate (FAR) will increase. With this score it is possible to determine if the system performs well. Images of fingerprints which are not the same, should have a low similarity score (near 0), while images of fingerprints that are the same, should have a high similarity score.

Before the fingerprint template is made, the VeriFinger software performs a quality check on the images to ensure the used images are good enough for detecting the minutiae points. Images that do not pass this check are rejected. Out of the 176 SCF images, 6 images do not pass the default quality check. For the DSCF images, 5 images do not pass the quality check, this is a good indication that the SCF images have a comparable quality to DSCF images.

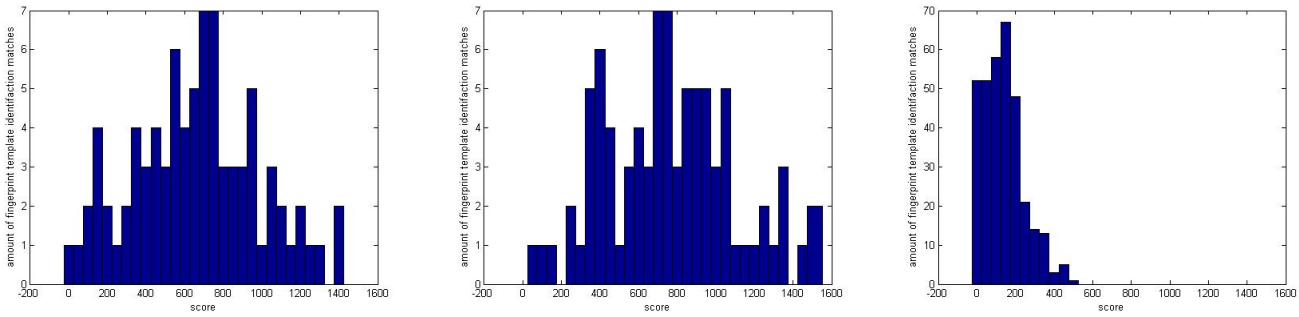
The created templates can now be matched against the other templates and the scores as seen in figure 5.1 are obtained. To give an indication of the meaning of the scores, the score is - according to the VeriFinger SDK documentation [18]- related to the FAR with:

$$FAR = 10^{\frac{score}{-12}} \tag{5.1}$$

This means that a score of 48 should be equal to a FAR of 0.01%.

It can be seen in figure 5.1 that the score distribution of DSCF against DSCF templates is about the same as SCF against SCF, while the DSCF against SCF histogram shows a distribution with far lower scores. While this indicates that the matching of DSCF against SCF images performs worse than the other two, it does not mean that the comparison fails. In figure 5.1 it can also be seen that almost all false match scores are below 30, while almost all genuine scores are above 30 - even for DSCF against SCF matching -.

[1] It should be noted that the scores from these results are much lower than in the previous chapter, this is because the 6.2 version of the software as used in this chapter gives lower scores than the 7.0 version used in the previous chapter.




(a) DSCF against DSCF genuine match scores

(b) SCF against SCF genuine match scores

(c) DSCF against SCF genuine match scores

(d) DSCF against DSCF false match scores

(e) SCF against SCF false match scores

(f) DSCF against SCF false match scores

Figure 5.1: Histogram of scores obtained by matching the fingerprint templates with other templates of the same finger (top row), and by matching with templates of other fingers (bottom row). For DSCF against DSCF (a,d), SCF against SCF (b,e) and DSCF against SCF (c,f)

For a better comparison between the different matching cases, a plot is made which shows the percentage of matches with a score higher than the score shown on the x-axis, see figure 5.2a and 5.2b for the genuine match score and figure 5.2c for the false match score. This is useful because it shows how much percent of the genuine and false matches are detected above a certain score. The percentage of genuine templates that are accepted should be high for an as high score as possible, while it should rapidly become lower for false matches. It can be seen that the SCF images perform as good as the DSCF fingerprints. The SCF images have a slightly higher genuine match score, but they also have a slightly higher score for false matches. The DSCF against SCF comparison performs worse than the others, but looking at the scores of the false matches it is still usable.

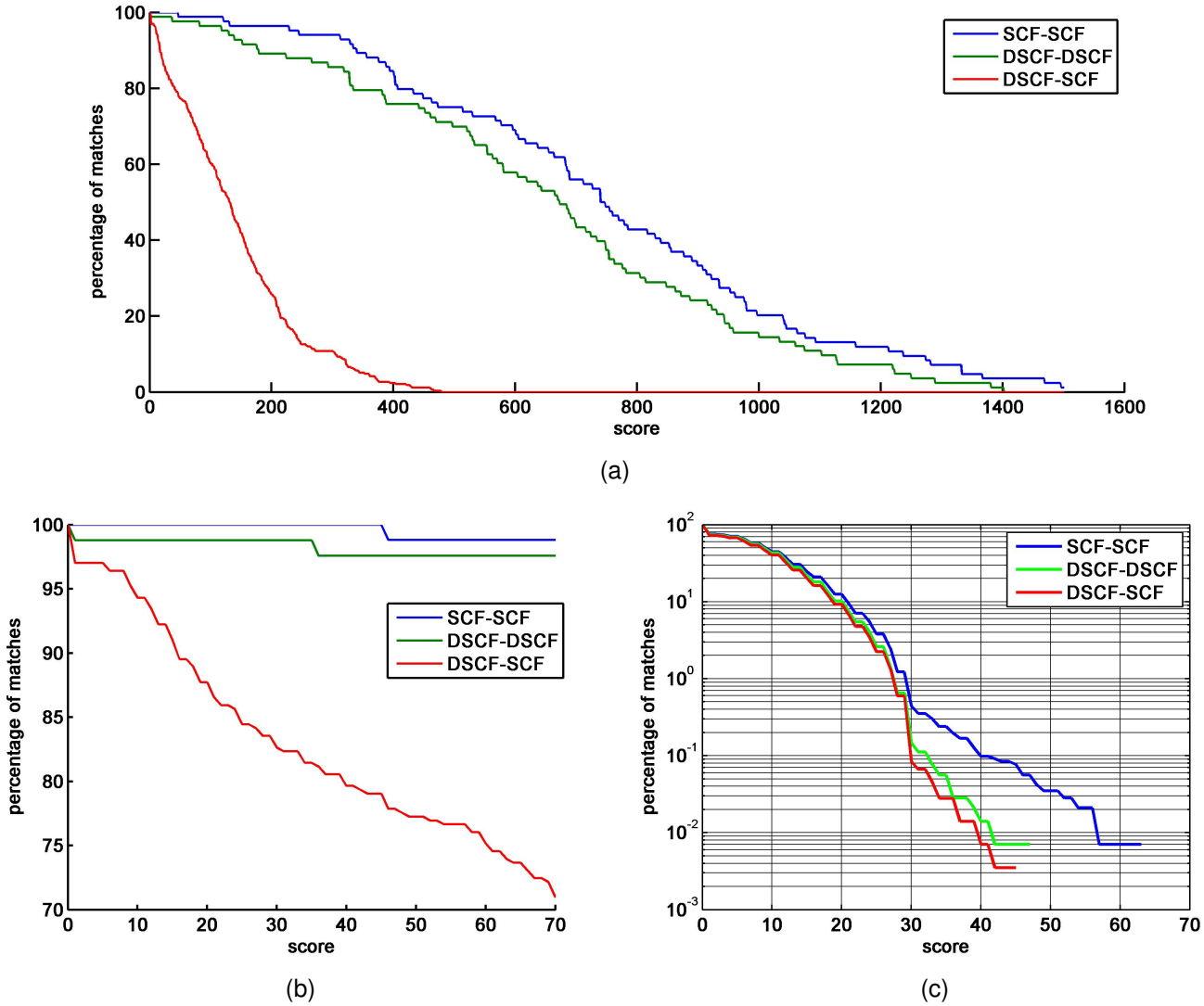



Figure 5.2: Graphs displaying the percentage of fingerprint template comparisons with a score equal or higher than the score on the x-axis. a) for fingerprint templates that should match, b) the same as a but zoomed in, c) for fingerprint templates that should not match. The blue line is for comparing SCF against SCF templates, the green for DSCF against DSCF and the red one for comparing DSCF against SCF fingerprint templates. In c, the end of a line indicates that there are zero fingerprint template matches with a score higher than the one at the end of the line.

There are a couple of reasons why the score for DSCF against SCF is less than the others. First of all the SCF image shows a larger fingerprint surface than DSCF images, resulting in a larger number of detected minutiae. While some of these minutiae can be matched against the DSCF images, others are not captured by the dedicated sensor. Since the score is not only based on matched minutiae, but also on non matched minutiae the score will be lower. Furthermore the capture methods are not the same. Sensor specific behaviour causes differences in the fingerprint image, these are among others the convex form of the finger which causes ridges to be closer together in SCF images, the way in which scars are handled, the effect of dirt on a finger, the amount of detail in the image, the effect of humidity and how much pressure is applied. Next to this, the VeriFinger enhancement algorithm tends to create false minutiae points near the edges of the fingers, since DSCF images are about the same size these false minutiae points can improve the score falsely, the same goes for SCF images, but not for the cross match. And finally, the DSCF images tend to create ridge bifurcation minutiae (a point where two ridges join into one ridge) instead of

ridge endings as can be seen in figure 5.3. It cannot be said with certainty if the dedicated sensor or the SCF has it right, but looking at the images it seems that the SCF image does it correct while the DSCF image falsely identifies the point as an bifurcation. The result of this can be that some minutiae points do not get connected by the VeriFinger software, resulting in lower scores.

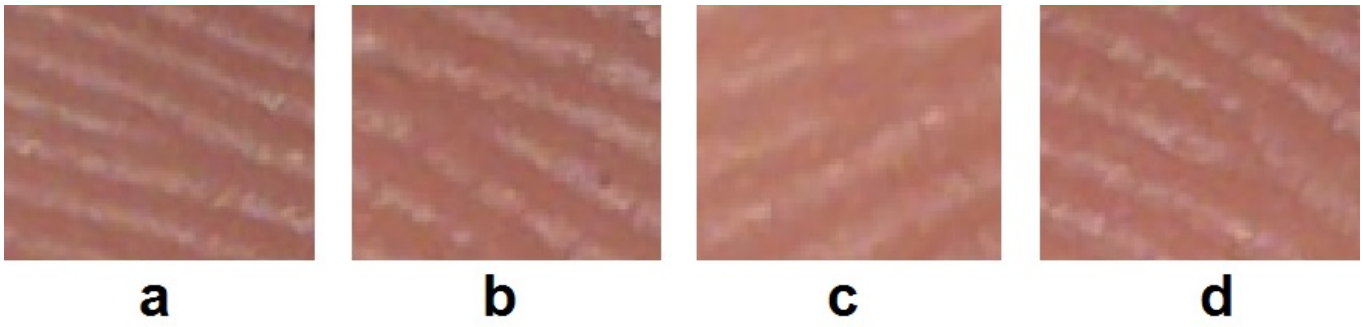



Figure 5.3: a-c) ridge endings according to the SCF image while they are classified as ridge bifurcations according to the DSCF image. d) ridge bifurcation according to both the SCF and DSCF image

For the DSCF against SCF comparison, a detection error trade-off (DET) curve is plotted in figure 5.4. This is only done for the cross match, because there are too few images to correctly plot a DET curve for SCF against SCF and DSCF against DSCF. According to the website of neurotechnology, the used U are U 4500 fingerprint scanner has an equal error rate (EER) of 0.15% [3], this means that only 15 out of 10,000 fingerprints are falsely accepted when the same amount is falsely rejected. Since the database only has 175 DSCF and 176 SCF images, the calculated EER would be highly uncertain. Because the SCF images show the same performance, the EER uncertainty is also too high for these images. This uncertainty is also reflected in figures 5.2b and 5.2c where the maximum score of the false comparisons is about 60, while more than 96 percent of the genuine comparisons have a higher score than 60.

It is however possible to calculate the EER for the comparison of DSCF against SCF images. This is because the number of genuine comparisons done is four times higher - every SCF against two same DSCF fingers, instead of one versus one with SCF or DSCF images which - so there are a total of 340 $(170 * 2)$ genuine finger comparisons and 28,560 $(170 * 168)$ false finger comparisons. Since the calculated EER as seen in figure 5.4 is 11.37%, around 11 out of 100 images are falsely accepted for the same false rejection rate, making the result reliable according to the amount of images used for the comparison. While an EER of 11.37% seems high, it is low considering the differences in capture method. And from the DET curve it can be found that the FRR rises to just 24.8% at a FAR of 0.0001% [2]. This means that 3 out of 4 fingers can be correctly recognized when only 1 in 1,000,000 images is falsely accepted. This score can be improved by combining a group of DSCF or SCF images of the same finger into one more reliable template allowing for higher scores and a better EER. It can also be improved by rejecting SCF and DSCF images with a low quality.

[2] This is extrapolated data. The extrapolation is based on the scores from the genuine comparisons as given by the VeriFinger software. According to the documentation of the software [18], the FAR can be calculated using the formula: FAR $= 10^{\frac{score}{-12}}$. By knowing the percentage of genuine comparisons with a score higher or equal to a certain number, the corresponding FAR can be approximated with the given formula.

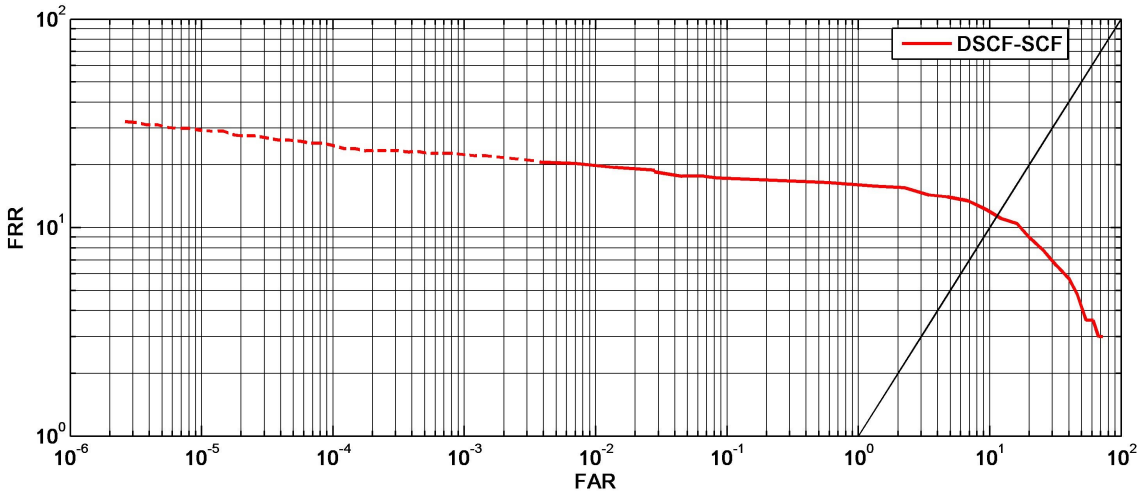



Figure 5.4: Detection error trade-off (DET) curve for DSCF against SCF images. The FAR and FRR are in percentages. The dashed line is extrapolated data. As seen in the figure, the equal error rate (EER) is 11.37%

It is remarkable that the images which should match but have a low score come mostly from the same persons. Out of the 340 genuine comparisons between SCF and DSCF templates, 45 comparisons have a score of 20 or less. From these 45, the distribution over the different persons can be found in the table below 5.1.

Table 5.1: Distribution of genuine scores <21 over the persons

| **Person number** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Number of low scores** | 0 | 0 | 1 | 10 | 0 | 8 | 0 | 8 | 12 | 0 | 11 |

The table makes clear that some persons are harder to cross match than others. Looking at the DSCF images in the database, it can be seen that these DSCF images show a lower quality than the images from other persons. It seems that two factors reduce the comparison quality among these persons. One is the small surface of the fingerprint pattern that is captured, probably due to a bad placement of the finger on the dedicated sensor. The other is a less distinct pattern, probably due to too much pressure of the finger on the dedicated sensor. It should also be noted that the fingerprints from person 8 show a high false match score in the SCF against SCF comparison. The aforementioned problems have a smaller impact on DSCF against DSCF or SCF against SCF comparisons because a person is more likely to make a fingerprint the same way as before in terms of placement and pressure, making the differences between successive fingerprint images less.

Comparing the performance of the approach taken in this article to the related work as seen in table 2.4, it can be concluded that the approach as described here shows the best results. Whether this is due to the used minutiae extractor, the capture process, the used smartphone, the pre-processing algorithm or a combination can however not be said with certainty. Though it can be concluded that in a well controlled environment the performance is comparable to that of a dedicated sensor. For the DSCF against SCF matching the performance cannot be compared to any related work, since this has not been done before. But it can be said that in order to use it in real life scenarios, the performance of the system should be improved.

# 6 Conclusion

The question whether it is possible to process smartphone captured fingerprint (SCF) images by commercial software made for dedicated sensor captured fingerprint (DSCF) images has been answered in this report. It can be concluded that this is certainly possible. Preprocessing is however required before the VeriFinger software is able to process the images in a reliable manner. The processing steps described involve the segmentation of the finger to remove the background, gray scale conversion, normalization, ridge orientation estimation and an enhancement step.

The segmentation of the finger has been written specifically for this application and cannot be used in real life applications with an erratic background. Since this was not the focus of the report, a simple finger segmentation algorithm is made. The segmentation algorithm does compensate for finger rotation, which is also advisable for real life applications.

The gray scale conversion is done by rejecting the red channel of the true color image and averaging the blue and green channel. In other work a blue light source is used to produce better quality images [12, 20, 23], since a smartphone does not possess a blue led it is chosen to use the blue channel of the image. But since the green channel shows an equally good pattern, the green channel is used as well. The image is furthermore scaled such that is has 500 ppi. This scaling is done based on the distance of the camera and the amount of pixels in the image. The fingerprint image is also cropped and an erosion on the border of the fingerprint image is done to remove the parts of the fingerprint that have a bad quality and produce false minutiae.

In the normalization step, the shadows on the fingerprint image are removed. This is needed for further processing and makes the image more consistent. Furthermore the VeriFinger software also relies on a consistent background color, which is partly achieved by the normalizing step.

A ridge orientation estimation step is required for the enhancement step to be able to enhance the pattern. Ridge orientation has been done in several different ways in related work. In this report is is shown that is is possible to do this estimation without using a Fourier transform, but simply by calculating the standard deviation along different directions in the image. The orientation estimation shows good results, even for highly curved fingerprint patterns and at minutiae points. It is however computational expensive.

The goal for the enhancement step was to make the images look more like images from dedicated sensors. The dedicated sensor images show a baseline (the background) from which peaks occur at the ridges as shown in figure 4.19. To make the images look like DSCF images, different enhancement steps are tried. The first algorithm tried to map all pixels to either black or white. While these images show a high contrast for the human eye, the computer readability is lower. This is mainly because all pixels are either black or white, having no nuances for the highly developed VeriFinger software which is better able to map uncertain pixels than the own developed enhancement algorithm. Therefore a second enhancement algorithm is made which enhances the pixel values above a mean line, the assumed ridges, with a power of 1.5. This makes the ridges more distinct, keeps the pixels which are lower than the mean line in a narrow band of possible values and makes the peaks for the more certain ridges higher than for those that are more uncertain. While this algorithm shows less contrast for a human, the VeriFinger software performs better

on these images.

The performance improvement is shown in the report. While the database was too small for a DET curve for the SCF-SCF and DSCF-DSCF comparison, the performance on the basis of the score provided by the VeriFinger algorithm shows results indicating that the SCF against SCF comparison performs as good as DSCF against DSCF, which should have an EER of 0.15%. This kind of performance has not been shown before in related work.

A cross match performance between SCF and DSCF images is also shown in the report. This cross-matching produces worse results than the other comparison tests due to sensor specific behaviour. An equal error rate of 11.37% is acquired in this report. For a more secure false acceptance rate of 1 in 1 million, the false rejection rate rises to 24.8%.

# 7 Discussion and Future Work

While this report shows that it is possible to use software written for dedicated sensors to extract a minutiae template for SCF images, the process can be improved. In the discussion possible improvements are handed and what could have been done better will be discussed.

Capturing the fingerprint images is done with the LED of the phone on, at the closest focus distance and with the maximum amount of pixels. The LED allows for uniform and consistent lighting in most circumstances and the close focus distance and large amount of pixels make the details better visible. It should be remarked that most processing steps in this report are based on the focus distance and number of pixels from the smartphone camera. When other smartphones are used for capturing the photos, the algorithm should be adjusted to account for the possibly different amount of vertical and horizontal pixels and the different focus distance.

The database acquired consists of 176 images. While this is good enough to show the performance of the sensors, a database with more photos would produce more reliable results. For future work, a performance comparison between the sensors with multiple images captured over a period of several weeks will make more reliable results as well.

It is clear that preprocessing is required before the VeriFinger software can be used. This preprocessing needs to consist of segmentation, gray scale conversion and depending on the enhancement step also normalization and ridge orientation estimation.

The segmentation as described in this report only works with a non erratic white background. For mobile phone applications it is recommended to use another segmentation process. It is however recommended to use the rotation compensation as shown in this report to make the final image better. For an even better image, convex form compensation could be considered.

For a gray scale conversion, it is highly recommended to do this using an averaging of the blue and green channel. In this way, the red channel - which shows the fingerprint pattern much less clear - is removed while the averaging of the green and blue channel reduces the amount of noise which would occur by taking only the blue channel.

The normalization step makes the image much more consistent and removes the shadows. In this report a computational inexpensive way of normalizing the image with interpolating is shown. There might be ways to normalize the image even better, but considering the amount of processing, it is recommended to keep the normalization simple.

Ridge orientation estimation as done here will probably be needed by any enhancement algorithm. While the approach mentioned in this work shows good results, other algorithms in related work produce similar results. Which one should be used cannot be said on forehand. In this report it was chosen to do it using the standard deviation because this looked like a promising new algorithm and did not involve Fourier transformations. Since the VeriFinger software would also need some kind of ridge orientation information, it can be considered to combine these two steps in one, preventing from doing the same computation twice.

The enhancement step has the most impact on the score provided by the VeriFinger software. In this report the images are enhanced in such a way that they show more similarities with DSCF images. This is done because the VeriFinger software is specially developed to enhance these kind of images. These DSCF images have a uniform background color and black peaks on the ridges. The approach in this report by enhancing the ridges which are more certain more than less likely ridges and thereby creating a more or less uniform background shows good results. While the enhancement as described in the report does this, it can be done better. As can be seen in figure 4.20, the calculation of the mean line above which the pixel values are classified as a ridge can be improved. This could be done by calculating a ridge frequency estimation instead of assuming that this frequency is the same over the whole image. With this frequency calculation it would also be easier to detect the start and ending of a ridge. It is however still recommended to enhance the peaks that have a higher amplitude more than smaller peaks. This should be done such that the VeriFinger software can make a better decision if a peak is a ridge or not.

The results obtained by the test set are promising, but it should be stressed that a larger database would provide more reliable results. Looking at the scores provided by the VeriFinger software there are however no extreme scores and the SCF score shows a same pattern as the DSCF images. Since the used dedicated sensor is tested much more extensively by others, the score pattern can be used to say something about the performance of the system.

# Bibliography


[1] Fvc2002 database. http://bias.csr.unibo.it/fvc2002/download.asp. Accessed: 16-6-2014.

[2] Lowess smoothing. http://www.mathworks.nl/help/curvefit/lowess-smoothing.html. Accessed: 18-6-2014.

[3] Neurotechnology det curve u are u 4000 sensor. http://www.neurotechnology.com/res/VeriFinger-ROC-U-are-U-4000.png. Accessed: 7-7-2014.

[4] Touch id - iphone fingerprintscanner. http://www.iphoneclub.nl/dossiers/touch-id/. Accessed: 17-6-2014.

[5] Verifinger demo software. http://www.neurotechnology.com/download.html#vf. Accessed: 15-6-2014.

[6] Fengling Han, Jiankun Hu, M Alkhathami, and Kai Xi. Compatibility of photographed images with touch-based fingerprint verification software. In *Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on*, pages 1034–1039. IEEE, 2011.

[7] Bee Yan Hiew, Andrew Beng Jin Teoh, and David Chek Ling Ngo. Preprocessing of fingerprint images captured with a digital camera. In *Control, Automation, Robotics and Vision, 2006. ICARCV'06. 9th International Conference on*, pages 1–6. IEEE, 2006.

[8] Bee Yan Hiew, Andrew Beng Jin Teoh, and Ooi Shih Yin. A secure digital camera based fingerprint verification system. *Journal of Visual Communication and Image Representation*, 21(3):219–231, 2010.

[9] BY Hiew, Andrew BJ Teoh, and YH Pang. Digital camera based fingerprint recognition. In *Telecommunications and Malaysia International Conference on Communications, 2007. ICT-MICC 2007. IEEE International Conference on*, pages 676–681. IEEE, 2007.

[10] Lin Hong, Yifei Wan, and Anil Jain. Fingerprint image enhancement: algorithm and performance evaluation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):777–789, 1998.

[11] MOHAMMED S Khalil and FK Wan. A review of fingerprint pre-processing using a mobile phone. In *Wavelet Analysis and Pattern Recognition (ICWAPR), 2012 International Conference on*, pages 152–157. IEEE, 2012.

[12] R Donida Labati, Angelo Genovese, Vincenzo Piuri, and Fabio Scotti. Contactless fingerprint recognition: a neural approach for perspective and rotation effects reduction. In *IEEE Workshop on Computational Intelligence in Biometrics and Identity Management*, 2013.

[13] Ruggero Donida Labati, Angelo Genovese, Vincenzo Piuri, and Fabio Scotti. Contactless fingerprint recognition: A neural approach for perspective and rotation effects reduction. In *CIBIM*, pages 22–30, 2013.

[14] Chulhan Lee, Sanghoon Lee, Jaihie Kim, and Sung-Jae Kim. Preprocessing of a fingerprint image captured with a mobile camera. In *Advances in Biometrics*, pages 348–355. Springer, 2005.

[15] Dongjae Lee, Kyoungtaek Choi, Heeseung Choi, and Jaihie Kim. Recognizable-image selection for fingerprint recognition with a mobile-device camera. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(1):233–243, 2008.

[16] Guoqiang Li, Bian Yang, R Raghavendra, and Christoph Busch. Testing mobile phone camera based fingerprint recognition under real-life scenarios. *Norsk informasjonssikkerhetskonferanse (NISK)*, 2012, 2012.

[17] Robert Mueller and Raul Sanchez-Reillo. An approach to biometric identity management using low cost equipment. In *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP'09. Fifth International Conference on*, pages 1096–1100. IEEE, 2009.

[18] Neurotechnology. *VeriFinger SDK 6.2 Developer's Guide*, 6.2.0.10 edition, 4 2011.

[19] Vincenzo Piuri and Fabio Scotti. Fingerprint biometrics via low-cost sensors and webcams. In *Biometrics: Theory, Applications and Systems, 2008. BTAS 2008. 2nd IEEE International Conference on*, pages 1–6. IEEE, 2008.

[20] Emiko Sano, Takuji Maeda, Takahiro Nakamura, Masahiro Shikai, Koji Sakata, Masahito Matsushita, and Koichi Sasakawa. Fingerprint authentication device based on optical characteristics inside a finger. In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on*, pages 27–27. IEEE, 2006.

[21] Chris Stein, Vincent Bouatou, and Christoph Busch. Video-based fingerphoto recognition with anti-spoofing techniques with smartphone cameras. In *Biometrics Special Interest Group (BIOSIG), 2013 International Conference of the*, pages 1–12. IEEE, 2013.

[22] Chris Stein, Claudia Nickel, and Christoph Busch. Fingerphoto recognition with smartphone cameras. In *Biometrics Special Interest Group (BIOSIG), 2012 BIOSIG-Proceedings of the International Conference of the*, pages 1–12. IEEE, 2012.

[23] Yongchang Wang, Laurence G Hassebrook, and Daniel L Lau. Data acquisition and processing of 3-d fingerprints. *Information Forensics and Security, IEEE Transactions on*, 5(4):750–760, 2010.

# Appendix

The matlab files as used for the different enhancement steps are shown below in the appendix. The used matlab version is R2013b.

## A: wrapper code

This code loads the SCF image, calls the segmentation function, converts the image to a gray scale image, calls the normalization function, rotates the image if needed and saves it.

```
PersonIDs = {'P01','P02','P03','P04','P05','P06','P07','P08','P09','P10','P11'};

for IDs = 1:length(PersonIDs);

    % Get the image
    %%%%%%%%%%%%%%%%%%%%
    folder = ['C:\Users\Wout\Pictures\Smartphone\' PersonIDs{IDs} '\'];
    foldersave = ['C:\Users\Wout\Pictures\Smartphone\' PersonIDs{IDs} '\mean\'];
    imagefiles = dir([folder '*.jpg']);
    nfiles = length(imagefiles);    % Number of files found

    for ii=1:nfiles
        % Read the input image
        %%%%%%%%%%%%%%%%%%%%%%
        currentfilename = [folder imagefiles(ii).name]
        im = imread(currentfilename);

        % Segment the input image and show it
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        [im_segmented,finger_width_avg,rotation] = segment2(im);

        % Convert to grayscale
        %%%%%%%%%%%%%%%%%%%%%%
        im_gray = im_segmented(:,:,2)/2 + im_segmented(:,:,3)/2;

        % Apply local normalization
        %%%%%%%%%%%%%%%%%%%%%%
        [im_mean] = nomalize_local(im_gray,30);

        % Rotate the input image
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        im_scaled = im_mean;

        if rotation
            im_scaled = imrotate(im_scaled,rotation);       % Compensate for horizontal rotation
        end

        % Save the image
        %%%%%%%%%%%%%%%%%%%%%%%%%
        imagename = [foldersave imagefiles(ii).name];
        imwrite(uint8(im_scaled),imagename,'jpeg','Quality',100,'BitDepth',8);
    end
end
```

## B: segmentation

This algorithm gets as input the true color SCF image and segments the finger. As output this function provides the segmented images, the calculated finger width and the calculated finger rotation.

```
% Segment the input color image based on its green values
% Output is the segmented image, the average finger width calculated
% from the average finger width 300—1000 pixels from the tip of the finger
% and the rotation of the finger
%
% Example:
```

```
% [im_segmented,finger_width_avg,rotation] = segment(image);

function [im_segmented,finger_width_avg,rotation] = segment2(im)

% Take only the green image
%%%%%%%%%%%%%%%%%%%%%%%%%
img_g = im(:,:,2);

[h,w] = size(img_g);        % Calculate the height and width of the image
Cy = round(0.5*h);          % Center of the image y values
Cx = round(0.5*w);          % Center of the image x values

% Find the minimum in green values to detect the finger tip edge
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
in_line_hor = smooth(double(img_g(Cy,:)),30);   % Get a line along the center
[~, max_index_hor] = min(in_line_hor(1632:end));% Find the minimum value of the line
max_index_hor = max_index_hor + 1631;           % Because the start derivative is removed

hand = max_index_hor <= Cx;     % 1 for right hand, 0 for left hand
if ~hand
    im = flipdim(im,2);
    img_g = im(:,:,2);
    max_index_hor = w - max_index_hor;
end

step = 50;                                  % Distance in pixels between measurement lines
varloop = 0;
no_steps = abs(floor((max_index_hor-w)/step));   % Number of lines considered
y_finger_min_index = zeros(no_steps+1,1);   % For saving the minimum values
y_finger_max_index = y_finger_min_index;    % For saving the maximum values

for x_index = max_index_hor:step:w
    varloop = varloop + 1;
    in_line = smooth(double(img_g(:,x_index)),40);          % The smoothed pixel values on that line

    [mini,min_index] = min(in_line(Cy:end-700));            % Get the index of the minimum value
    min_index2 = find(in_line((min_index + Cy - 1):end-700)>mini*1.3,1,'first');
    if isempty(min_index2)
        [mini,min_index] = min(in_line(Cy:end-500));            % Get the index of the minimum value
        min_index2 = find(in_line((min_index + Cy - 1):end-500)>mini*1.3,1,'first');
        if isempty(min_index2)
            [mini,min_index] = min(in_line(Cy:end-300));            % Get the index of the minimum value
            min_index2 = find(in_line((min_index + Cy - 1):end-300)>mini*1.3,1,'first');
            if isempty(min_index2)
                min_index2 = 0;
            end
        end
    end
    y_finger_max_index(varloop) = min_index2 + min_index + Cy - 1;

    [mini,min_index] = min(in_line(900:Cy));                % Get the index of the minimum value
    min_index2 = find(in_line(850:(min_index + 899))>mini*1.3,1,'last');
    plus = 849;
    if isempty(min_index2)
        [mini,min_index] = min(in_line(700:Cy));                % Get the index of the minimum value
        min_index2 = find(in_line(650:(min_index + 699))>mini*1.3,1,'last');
        plus = 649;
        if isempty(min_index2)
            [mini,min_index] = min(in_line(500:Cy));                % Get the index of the minimum value
            min_index2 = find(in_line(450:(min_index + 499))>mini*1.3,1,'last');
            plus = 449;
            if isempty(min_index2)
                [mini,min_index] = min(in_line(300:Cy));                % Get the index of the minimum value
                min_index2 = find(in_line(250:(min_index + 299))>mini*1.3,1,'last');
                plus = 249;
                if isempty(min_index2)
                    min_index2 = 0;
                end
            end
        end
    end

    diff_in_line = diff(double(in_line(min_index2 + plus - 40:min_index2 + plus + 160)));
    if max(diff_in_line) > 1.5 && min(diff_in_line) < -1.5 % There are two fingers close to each other, take the right one
        [~,min_index3] = min(diff_in_line);
        min_index3 = min_index3 + 40; % Compensation
        y_finger_min_index(varloop) = min_index2 + plus + min_index3 - 41;
    else
        y_finger_min_index(varloop) = min_index2 + plus;
    end

    % Compensate for the effect that a shift of the finger causes a wrong
    % middle point at the end
    if varloop > 10
       mean_dist = mean(y_finger_max_index(6:10) - y_finger_min_index(6:10));
       if  (y_finger_max_index(varloop) - y_finger_min_index(varloop)) < 0.8*mean_dist % Wrong point extracted due to shift of finger
           mean_min = mean(y_finger_min_index(6:10));
           mean_max = mean(y_finger_max_index(6:10));
           if abs(mean_min -  y_finger_min_index(varloop)) < abs(mean_max -  y_finger_max_index(varloop))
                y_finger_max_index(varloop) = y_finger_min_index(varloop) + 1.1*mean_dist;
                if y_finger_max_index(varloop) > h;
                    y_finger_max_index(varloop) = h;
                end
           else
                y_finger_min_index(varloop) = y_finger_max_index(varloop) - 1.1*mean_dist;
                if y_finger_min_index(varloop) < 1;
                    y_finger_min_index(varloop) = 1;
```

```
                end
            end
        end
    end

    % Compensate for the effect of a too large offset used
    if y_finger_max_index(varloop) - y_finger_min_index(varloop) < 400 && varloop > 1
        [mini,min_index] = min(in_line(Cy:end-300));            % Get the index of the minimum value
        min_index2 = find(in_line((min_index + Cy - 1):end-300)>mini*1.3,1,'first');
        if isempty(min_index2)
            min_index2 = 0;
        end
        y_finger_max_index(varloop) = min_index2 + min_index + Cy - 1;
    end
end

% Smooth the min and max indexes to remove outliers
smooth_min = smooth(y_finger_min_index,'rlowess',5);
smooth_max = smooth(y_finger_max_index,'rlowess',5);

% If they are all the same value, something went wrong with the rlowess method
if round(smooth_min(10)) == round(smooth_min(11)) && round(smooth_min(10)) == round(smooth_min(12)) && round(smooth_min(10)) == round(smooth_min(13))
    smooth_min = smooth(y_finger_min_index,5);
end

y_finger_min_index(5:end) = round(smooth_min(5:end));
y_finger_max_index(5:end) = round(smooth_max(5:end));

y_finger_min_index(1) = round((y_finger_min_index(2) + y_finger_max_index(2))/2)-80;
y_finger_max_index(1) = y_finger_min_index(1)+80;
y_finger_min_index(end) = y_finger_min_index(end-1);
y_finger_max_index(end) = y_finger_max_index(end-1);

y_finger_min_index = smooth(y_finger_min_index,2);

% Crop based on the indexes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hor_crop1 = max_index_hor;
vert_crop1 = min(y_finger_min_index);
vert_crop2 = max(y_finger_max_index);
if vert_crop2 > h
    vert_crop2 = h;
end

finger_width_avg = round(mean(y_finger_max_index(6:20) - y_finger_min_index(6:20)));
finger_width = max(y_finger_max_index - y_finger_min_index);
finger_height = round(1.6 * finger_width);

rot_top = atand(mean(diff(y_finger_min_index(6:20)))/step);     % Calc the rotation of the top edge
rot_bottom = atand(mean(diff(y_finger_max_index(6:20)))/step);  % Calc the rotation of the bottom edge
rotation = 0;
if sign(rot_top) == sign(rot_bottom)                            % If both have an angle in the same direction
    rotation = rot_top * (rot_top <= rot_bottom) + rot_bottom * (rot_top > rot_bottom);
end

hor_crop2 = hor_crop1 + finger_height;
if hor_crop2 > w
    hor_crop2 = w;
end

im = im(vert_crop1:vert_crop2,hor_crop1:hor_crop2,:);           % Crop the image

% Create the background seperation mask
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mask = false(vert_crop2-vert_crop1+1,hor_crop2-hor_crop1+1);
y_finger_min_index = y_finger_min_index - vert_crop1 + 1;
y_finger_max_index = y_finger_max_index - vert_crop1;
varloop = 0;

for index = 1:step:(hor_crop2-hor_crop1+1-step)
    varloop = varloop + 1;

    if varloop > 1
    ymin = round(linspace((y_finger_min_index(varloop-1)+y_finger_min_index(varloop))/2,...
        (y_finger_min_index(varloop)+y_finger_min_index(varloop+1))/2,step));
    ymax = round(linspace((y_finger_max_index(varloop-1)+y_finger_max_index(varloop))/2,...
        (y_finger_max_index(varloop)+y_finger_max_index(varloop+1))/2,step));
    else
    ymin = round(linspace(y_finger_min_index(varloop),(y_finger_min_index(varloop)+y_finger_min_index(varloop+1))/2,step));
    ymax = round(linspace(y_finger_max_index(varloop),(y_finger_max_index(varloop)+y_finger_max_index(varloop+1))/2,step));
    end

    for index2 = 1:step
        mask(ymin(index2):ymax(index2),index+index2) = true;
    end
end

% Use the mask
%%%%%%%%%%%%%%
im_segmented = im.*uint8(repmat(mask,1,1,3));
im_segmented = im_segmented(:,step:end-step,:);

% Pad 60 pixels at either side of the image so it can be better used later on
im_segmented = padarray(im_segmented,[60 60]);

end
```

# C: normalization

This function normalizes the gray scale image which it gets as input. It furthermore takes a block size as input for the normalization, in this report a block size of 30x30 is used. The output of the function is the normalized image.

```
% Normalize the image based on its mean
%
% Input is the grayscale image and the block size used for normalizing
% Output is the normalized image
%
% Example:
% [img_mean] = nomalize_local(img_gray,block_size)

function [img_mean] = nomalize_local(img_gray,block_size)

x = block_size;   % Set block size in x direction
y = block_size;   % Set block size in y direction
I = img_gray;
[w,h] = size(I);  % Calculate the width and height of the image
w1=floor(w/x)*x;
h1=floor(h/y)*y;
mean_mask = zeros(size(I)); % Initialize the mean mask

% Calculate the mean for the 30x30 blocks
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:2*x:w1
    for j=1:2*y:h1
        a=i+x-1;
        b=j+y-1;
        imgmean = mean2(I(i:a,j:b));
        mean_mask(i+x-round(x/2),j+y-round(y/2)) = imgmean;
    end
end

% Do the interpolation in horizontal direction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:2*x:w1
    xi = i+x-round(x/2);
    for j=1:2*y:h1-2*y
        yi = j+y-round(y/2);
        step = (mean_mask(xi,(yi+2*y)) - mean_mask(xi,yi))/(2*y+1);
        if step==0
            mean_mask(xi,yi+1:yi+2*y-1) = meshgrid(mean_mask(xi,yi),1:2*y-1);
        else
            grid = mean_mask(xi,yi):step:mean_mask(xi,yi+2*y);
            mean_mask(xi,yi:yi+2*y) = grid(1:2*y+1);
        end
    end
end

% Do the interpolation in vertical direction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:2*x:w1-2*x
    xi = i+x-round(x/2);
    for j=1:h1
        yi = j;
        step = (mean_mask(xi+2*x,yi) - mean_mask(xi,yi))/(2*x+1);
        if step==0
            mean_mask(xi+1:xi+2*x-1,yi) = meshgrid(mean_mask(xi,yi),1:2*x-1);
        else
            grid = mean_mask(xi,yi):step:mean_mask(xi+2*x,yi);
            mean_mask(xi:xi+2*x,yi) = grid(1:2*x+1);
        end
    end
end

% Apply the mask
img_mean = double(img_gray) + (150 - mean_mask - (mean_mask<30)*150);

% Convert to type uint8
img_mean = uint8(img_mean);

end
```

# D: enhancement

This function loads the normalized image, does the rotation calculation, enhances the image and saves the enhanced image.

```
PersonIDs = {'P01','P02','P03','P04','P05','P06','P07','P08','P09','P10','P11'};

for IDs = 11:length(PersonIDs);

foldersave = ['C:\Users\Wout\Pictures\Smartphone\' PersonIDs{IDs} '\enhanced\'];\begin{lstlisting}
folder = ['C:\Users\Wout\Pictures\Smartphone\' PersonIDs{IDs} '\mean\'];
imagefiles = dir([folder '*.jpg']);
nfiles = length(imagefiles);    % Number of files found

for ii=1:nfiles

% Get the image
%%%%%%%%%%%%%%%%%%%%
currentfilename = [folder imagefiles(ii).name]
im = imread(currentfilename);
im = im(:,:,1);

% Remove the dirty spots
im(im<130 & im > 1) = 150;

im_mean = conv2(im,ones(5)/25,'same'); % Smooth the image

Std1 = 9;   % size of the std calculation block
Std2 = 3;   % size of the std calculation block

gx = stdfilt(im_mean,ones(Std2,Std1)); % calc the standard deviation
gy = stdfilt(im_mean,ones(Std1,Std2)); % calc the standard deviation

im_rot = imrotate(im_mean,45);         % rotate the image
gx2 = stdfilt(im_rot,ones(Std2,Std1)); % calc the standard deviation
gy2 = stdfilt(im_rot,ones(Std1,Std2)); % calc the standard deviation
gx2 = imrotate(gx2,-45,'crop');        % rotate back
gy2 = imrotate(gy2,-45,'crop');        % rotate back

% Remove the borders of the rotated calculation to make it the same size
% as the image again
%%%%%%%%%%%%%%%%%%%%
[w2,h2] = size(im_mean);
[wx2,hx2] = size(gx2);
Wborder = round((wx2 - w2)/2);
Hborder = round((hx2 - h2)/2);
gx2 = gx2(Wborder:wx2-Wborder,Hborder:hx2-Hborder);
gy2 = gy2(Wborder:wx2-Wborder,Hborder:hx2-Hborder);
[w,h] = size(gx);
gx2 = gx2(1:w,1:h);
gy2 = gy2(1:w,1:h);

im_std_gx = gx;
im_std_gy = gy;
im_std_gx2 = gx2;
im_std_gy2 = gy2;

% Calculate the rotation mask by finding which seperate rotation mask
% has the highest standard deviation at which pixel values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    rot_mask = zeros(size(im_std_gx));
rot_mask_real = im_std_gx;
    rot_mask(im_std_gx<im_std_gy) = 0.33;
rot_mask_real(im_std_gx<im_std_gy) = im_std_gy(im_std_gx<im_std_gy);
    rot_mask(rot_mask_real<im_std_gx2) = 0.66;
rot_mask_real(rot_mask_real<im_std_gx2) = im_std_gx2(rot_mask_real<im_std_gx2);
    rot_mask(rot_mask_real<im_std_gy2) = 1;
rot_mask_real(rot_mask_real<im_std_gy2) = im_std_gy2(rot_mask_real<im_std_gy2);

% Calculate the final rotation matrix by finding the largest amount
% of found rotations in a 30x30 block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
block_size = 30;
x = block_size;
y = block_size;
I = rot_mask;
[w,h] = size(I);
w1=floor(w/x)*x;
h1=floor(h/y)*y;
mean_rot_mask = zeros(size(I));
for i=1:x:w1-x
    for j=1:y:h1-y
        a=i+x;
        b=j+y;
        deg0 = sum(sum(I(i:a,j:b)==0));          % amount of 0 degree values in 30x30 block
        deg90 = sum(sum(I(i:a,j:b)==0.33));      % amount of 90 degree values in 30x30 block
        deg45 = sum(sum(I(i:a,j:b)==0.66));      % amount of 45 degree values in 30x30 block
        deg135 = sum(sum(I(i:a,j:b)==1));        % amount of 135 degree values in 30x30 block
        array = [deg0, deg90, deg45, deg135];

        [max1,index1] = max(array);              % Find the most common rotation
        array(index1) = -1;
```

```
        [max2,index2] = max(array);             % Find the seccond most common rotation

        % Average the 2 most common found rotations based on the amount of
        % occurances in the 30x30 block
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        switch index1
            case 1
                switch index2
                    case 2
                        rotation = 0;
                    case 3
                        rotation = max2/(max1+max2) * 45;
                    case 4
                        rotation = max1/(max1+max2) * 45 + 135;
                end
            case 2
                switch index2
                    case 1
                        rotation = 90;
                    case 3
                        rotation = max1/(max1+max2) * 45 + 45;
                    case 4
                        rotation = max2/(max1+max2) * 45 + 90;
                end
            case 3
                switch index2
                    case 1
                        rotation = max1/(max1+max2) * 45;
                    case 2
                        rotation = max2/(max1+max2) * 45 + 45;
                    case 4
                        rotation = 45;
                end
            case 4
                switch index2
                    case 1
                        rotation = max2/(max1+max2) * 45 + 135;
                    case 2
                        rotation = max1/(max1+max2) * 45 + 90;
                    case 3
                        rotation = 135;
                end
        end

        % Make the rotation mask
        mean_rot_mask(i:a,j:b) = rotation;
    end
end

% Calculate the width and height of the rotation mask
[w,h] = size(mean_rot_mask);

% Image that has to be enhanced
im2enhance = im_mean;

% Setup the mask for the convolution mean orientation calculation
conv_mask0 = ones(1,29);
conv_mask90 = ones(29,1);
conv_mask45 = diag(ones(29,1),0);
conv_mask135 = fliplr(diag(ones(29,1),0));

% Calculate the mean along the orientations
mean_0 = conv2(im2enhance,conv_mask0,'same')./29;
mean_90 = conv2(im2enhance,conv_mask90,'same')./29;
mean_45 = conv2(im2enhance,conv_mask45,'same')./29;
mean_135 = conv2(im2enhance,conv_mask135,'same')./29;

% Initialize variables
im_enh = im2enhance;
im_enh0 = im2enhance;
im_enh45 = im2enhance;
im_enh90 = im2enhance;
im_enh135 = im2enhance;

% Enhance the image according to the orientation and its mean using the
% power of 1.5
im_enh0 = real(im_enh0 - 24 + sign(im_enh0-mean_0).*((im_enh0-mean_0).^1.5));
im_enh45 = real(im_enh45 - 24 + sign(im_enh45-mean_45).*((im_enh45-mean_45).^1.5));
im_enh90 = real(im_enh90 - 24 + sign(im_enh90-mean_90).*((im_enh90-mean_90).^1.5));
im_enh135 = real(im_enh135 - 24 + sign(im_enh135-mean_135).*((im_enh135-mean_135).^1.5));

% Get the indexes at which place which rotation occurs
mask0 = (mean_rot_mask < 23 | mean_rot_mask > 157);
mask45 = (mean_rot_mask >= 23 & mean_rot_mask < 68);
mask90 = (mean_rot_mask >=68 & mean_rot_mask < 112);
mask135 = (mean_rot_mask >=112 & mean_rot_mask <=157);

% Make the enhanced image
im_enh(mask0) = im_enh0(mask0);
im_enh(mask45) = im_enh45(mask45);
im_enh(mask90) = im_enh90(mask90);
im_enh(mask135) = im_enh135(mask135);

% Invert the image so ridges become black and valleys white
im_enh(im_enh<0) = 0;
im_enh2 = imcomplement(uint8(im_enh))+70;
```

```
% Remove the border which causes many false minutiae
imbw = im2enhance>0;
mask_border = imerode(imbw,ones(140));
im_enh2(~mask_border) = 204;
top_crop = find(imbw(:,end-700),1,'first');
bottom_crop = find(imbw(:,end-700),1,'last');

% Crop resize and rotate the image
im_enh2 = im_enh2(top_crop:bottom_crop,1:round(0.65*h));
im_enh3 = imresize(im_enh2,0.38);   % Resize the image, so the PPI matches the DSCF one
im_enh4 = imrotate(im_enh3,-90);    % Rotate 90 degrees

% Save the image
imagename = [foldersave imagefiles(ii).name];
imwrite(uint8(im_enh4),imagename,'jpeg','Quality',100,'BitDepth',8);

end
end
```

# E: old enhancement algorithm

This function enhances the image according to the first described algorithm with mapping to black or white. For comments I would like to refer to appendix D, since most of the code is the same.

```
% Get the image
folder = 'C:\Users\Wout\Dropbox\BSC eindopdracht\Matlab\img\mean\';
foldersave = 'C:\Users\Wout\Dropbox\BSC eindopdracht\Matlab\img\enhanced_2\';
imagefiles = dir([folder '*.jpg']);
nfiles = length(imagefiles);    % Number of files found

for ii=1:nfiles
currentfilename = [folder imagefiles(ii).name]
im = imread(currentfilename);

% Find the dirty spots and remove them
im_dirt = im;
im_dirt(im>130) = 220;

% Remove the dirty spots
im(im<130 & im > 1) = 150;

im_mean = conv2(im,ones(5)/25,'same');

Std1 = 9;
Std2 = 3;

gx = stdfilt(im_mean,ones(Std2,Std1));
gy = stdfilt(im_mean,ones(Std1,Std2));

im_rot = imrotate(im_mean,45);
gx2 = stdfilt(im_rot,ones(Std2,Std1));
gy2 = stdfilt(im_rot,ones(Std1,Std2));
gx2 = imrotate(gx2,-45,'crop');
gy2 = imrotate(gy2,-45,'crop');
[w2,h2] = size(im_mean);
[wx2,hx2] = size(gx2);
Wborder = round((wx2 - w2)/2);
Hborder = round((hx2 - h2)/2);
gx2 = gx2(Wborder:wx2-Wborder,Hborder:hx2-Hborder);
gy2 = gy2(Wborder:wx2-Wborder,Hborder:hx2-Hborder);
[w,h] = size(gx);
gx2 = gx2(1:w,1:h);
gy2 = gy2(1:w,1:h);

im_std_gx = gx;
im_std_gy = gy;
im_std_gx2 = gx2;
im_std_gy2 = gy2;

    rot_mask = zeros(size(im_std_gx));
rot_mask_real = im_std_gx;
    rot_mask(im_std_gx<im_std_gy) = 0.33;
rot_mask_real(im_std_gx<im_std_gy) = im_std_gy(im_std_gx<im_std_gy);
    rot_mask(rot_mask_real<im_std_gx2) = 0.66;
rot_mask_real(rot_mask_real<im_std_gx2) = im_std_gx2(rot_mask_real<im_std_gx2);
    rot_mask(rot_mask_real<im_std_gy2) = 1;
rot_mask_real(rot_mask_real<im_std_gy2) = im_std_gy2(rot_mask_real<im_std_gy2);

block_size = 30;
x = block_size;
y = block_size;
I = rot_mask;
[w,h] = size(I);
w1=floor(w/x)*x;
h1=floor(h/y)*y;
mean_rot_mask = zeros(size(I));
for i=1:x:w1-x
    for j=1:y:h1-y
```

```
        a=i+x;
        b=j+y;
        deg0 = sum(sum(I(i:a,j:b)==0));
        deg90 = sum(sum(I(i:a,j:b)==0.33));
        deg45 = sum(sum(I(i:a,j:b)==0.66));
        deg135 = sum(sum(I(i:a,j:b)==1));
        array = [deg0, deg90, deg45, deg135];

        [max1,index1] = max(array);
        array(index1) = -1;
        [max2,index2] = max(array);

        switch index1
            case 1
                switch index2
                    case 2
                        rotation = 0;
                    case 3
                        rotation = max2/(max1+max2) * 45;
                    case 4
                        rotation = max1/(max1+max2) * 45 + 135;
                end
            case 2
                switch index2
                    case 1
                        rotation = 90;
                    case 3
                        rotation = max1/(max1+max2) * 45 + 45;
                    case 4
                        rotation = max2/(max1+max2) * 45 + 90;
                end
            case 3
                switch index2
                    case 1
                        rotation = max1/(max1+max2) * 45;
                    case 2
                        rotation = max2/(max1+max2) * 45 + 45;
                    case 4
                        rotation = 45;
                end
            case 4
                switch index2
                    case 1
                        rotation = max2/(max1+max2) * 45 + 135;
                    case 2
                        rotation = max1/(max1+max2) * 45 + 90;
                    case 3
                        rotation = 135;
                end
        end
        mean_rot_mask(i:a,j:b) = rotation;
    end
end

% Show the orientation on the input image
[w,h] = size(mean_rot_mask);
[X,Y] = meshgrid(1:h,1:w);
px_mean = cosd(mean_rot_mask);
py_mean = sind(mean_rot_mask);
L = sqrt(px_mean.^2 + py_mean.^2);
px_mean2 = px_mean ./ L;
py_mean2 = py_mean ./ L;
px_mean = py_mean2;  % Rotate 90 degrees
py_mean = -px_mean2; % Rotate 90 degrees

% Image that has to be enhanced
im2enhance = im_mean;

% Setup the mask for the convolution mean orientation calculation
conv_mask0 = ones(1,29);
conv_mask90 = ones(29,1);
conv_mask45 = diag(ones(29,1),0);
conv_mask135 = fliplr(diag(ones(29,1),0));

% Calculate the mean along the orientations
mean_0 = conv2(im2enhance,conv_mask0,'same')./29;
mean_90 = conv2(im2enhance,conv_mask90,'same')./29;
mean_45 = conv2(im2enhance,conv_mask45,'same')./29;
mean_135 = conv2(im2enhance,conv_mask135,'same')./29;

% Calculate the standard deviation along the orientations
std_0 = stdfilt(im2enhance,conv_mask0);
std_90 = stdfilt(im2enhance,conv_mask90);
std_45= stdfilt(im2enhance,conv_mask45);
std_135 = stdfilt(im2enhance,conv_mask135);

% Enhance the image according to the orientation and its mean and std
std_amount = 0.4;

im_enh = im2enhance;
im_enh0 = im2enhance;
im_enh45 = im2enhance;
im_enh90 = im2enhance;
im_enh135 = im2enhance;

im_enh0(im_enh0 > (mean_0 + std_0*std_amount)) = 220;
im_enh0(im_enh0 < (mean_0 - std_0*std_amount)) = 20;
```

```
im_enh45(im_enh45 > (mean_45 + std_45*std_amount)) = 220;
im_enh45(im_enh45 < (mean_45 - std_45*std_amount)) = 20;

im_enh90(im_enh90 > (mean_90 + std_90*std_amount)) = 220;
im_enh90(im_enh90 < (mean_90 - std_90*std_amount)) = 20;

im_enh135(im_enh135 > (mean_135 + std_135*std_amount)) = 220;
im_enh135(im_enh135 < (mean_135 - std_135*std_amount)) = 20;

mask0 = (mean_rot_mask < 23 | mean_rot_mask > 157);
mask45 = (mean_rot_mask >= 23 & mean_rot_mask < 68);
mask90 = (mean_rot_mask >=68 & mean_rot_mask < 112);
mask135 = (mean_rot_mask >=112 & mean_rot_mask <=157);

im_enh(mask0) = im_enh0(mask0);
im_enh(mask45) = im_enh45(mask45);
im_enh(mask90) = im_enh90(mask90);
im_enh(mask135) = im_enh135(mask135);

% Fill in most of the 'holes' (~= 220 | 20 regions) of the image using the
% closest other orientation
mask0 = (mean_rot_mask > 135 & im_enh135 < 220 & im_enh135 > 20) |...
    (mean_rot_mask < 45 & im_enh45 < 220 & im_enh45 > 20);
mask45 = (mean_rot_mask > 0 & mean_rot_mask < 45 & im_enh0 < 220 & im_enh0 > 20) |...
    (mean_rot_mask >= 45 & mean_rot_mask <= 90 & im_enh90 < 220 & im_enh90 > 20);
mask90 = (mean_rot_mask > 45 & mean_rot_mask < 90 & im_enh45 < 220 & im_enh45 > 20) |...
    (mean_rot_mask >= 90 & mean_rot_mask <= 135 & im_enh135 < 220 & im_enh135 > 20);
mask135 = (mean_rot_mask > 90 & mean_rot_mask < 135 & im_enh90 < 220 & im_enh90 > 20) |...
    (mean_rot_mask >= 135 & mean_rot_mask <= 180 & im_enh0 < 220 & im_enh0 > 20);

im_enh(mask0) = im_enh0(mask0);
im_enh(mask45) = im_enh45(mask45);
im_enh(mask90) = im_enh90(mask90);
im_enh(mask135) = im_enh135(mask135);

% Use the enhancement algorithm again
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Image that has to be enhanced
im2enhance = im_enh;

% Setup the mask for the convolution mean orientation calculation
conv_mask0 = ones(1,29);
conv_mask90 = ones(29,1);
conv_mask45 = diag(ones(29,1),0);
conv_mask135 = fliplr(diag(ones(29,1),0));

% Calculate the mean along the orientations
mean_0 = conv2(im2enhance,conv_mask0,'same')./29;
mean_90 = conv2(im2enhance,conv_mask90,'same')./29;
mean_45 = conv2(im2enhance,conv_mask45,'same')./29;
mean_135 = conv2(im2enhance,conv_mask135,'same')./29;

% Enhance the image according to the orientation and its mean and std
im_enh2 = im2enhance;
im_enh0 = im2enhance;
im_enh45 = im2enhance;
im_enh90 = im2enhance;
im_enh135 = im2enhance;

im_enh0(im_enh0 > mean_0) = 220;
im_enh0(im_enh0 < mean_0) = 20;

im_enh45(im_enh45 > mean_45) = 220;
im_enh45(im_enh45 < mean_45) = 20;

im_enh90(im_enh90 > mean_90) = 220;
im_enh90(im_enh90 < mean_90) = 20;

im_enh135(im_enh135 > mean_135) = 220;
im_enh135(im_enh135 < mean_135) = 20;

mask0 = (mean_rot_mask < 23 | mean_rot_mask > 157);
mask45 = (mean_rot_mask >= 23 & mean_rot_mask < 68);
mask90 = (mean_rot_mask >=68 & mean_rot_mask < 112);
mask135 = (mean_rot_mask >=112 & mean_rot_mask <=157);

im_enh2(mask0) = im_enh0(mask0);
im_enh2(mask45) = im_enh45(mask45);
im_enh2(mask90) = im_enh90(mask90);
im_enh2(mask135) = im_enh135(mask135);

% Invert the image
mask220 = im_enh2==220;
mask20 = im_enh2==20;
im_enh2(mask220) = 20;
im_enh2(mask20) = 220;

% Save the image
%%%%%%%%%%%%%%%%%%%%%%%%%
imagename = [foldersave strrep(imagefiles(ii).name,'mean','enh')];
imwrite(uint8(im_enh2),imagename,'jpeg','Quality',100,'BitDepth',8);

end
```