

**VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY**



**FINAL SEMESTER PROJECT FOR INFORMATION
SECURITY**

**FINGERPRINT RECOGNITION USING SIFT DETECTION
(MINUTIAE BASED ALGORITHM)**

INSTRUCTOR: **MRS. HUỖNH NGỌC TÚ**

STUDENT: **MAI BẢO THẠCH-520H0490**

HUỖNH NHẬT BẢO-520H0605

TRẦN MINH PHÚC-520H0668

CLASS: **20H50304**

20H50303

20H50304

COURSE: **24**

HỒ CHÍ MINH CITY, YEAR 2022

**VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY**



**FINAL SEMESTER PROJECT FOR INFORMATION
SECURITY**

**FINGERPRINT RECOGNITION USING SIFT DETECTION
(MINUTIAE BASED ALGORITHM)**

INSTRUCTOR: **MRS. HUỖNH NGỌC TÚ**

STUDENT: **MAI BẢO THẠCH-520H0490**

HUỖNH NHẬT BẢO-520H0605

TRẦN MINH PHÚC-520H0668

CLASS: **20H50304**

20H50303

20H50304

COURSE: **24**

HỒ CHÍ MINH CITY, YEAR 2022

THANKFUL WORDS

After working for a whole semester with the enthusiastic help and support of Mrs. Huỳnh Ngọc Tú, I was able to complete the report in the most complete and effective way. Her teaching has given our students a lot of knowledge as well as full skills in the specialized subject. Although couple of months is quite short, but that time has also helped me to easily approach the major step by step with a solid foundation, especially with the encouragement and help from seasoned lecturers.

I sincerely thank

Hồ Chí Minh City, November 28, 2022

Author

(Full name and signature)

THẠCH

MAI BẢO THẠCH

BẢO

HUỲNH NHẬT BẢO

PHÚC

TRẦN MINH PHÚC

REPORT COMPLETED AT TON DUC THANG UNIVERSITY

I hereby declare that this is my own report and is under the guidance of Mrs. Huỳnh Ngọc Tú. The research contents and results in this topic are honest and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the author himself from different sources, clearly stated in the reference section.

In addition, the project also uses a number of comments, assessments as well as data of other authors, other agencies and organizations, with citations and source annotations.

If I find any fraud I take full responsibility for the content of my report. Ton Duc Thang University is not related to copyright and copyright violations caused by me during the implementation process (if any).

Hồ Chí Minh city, November 28, 2022

Author

(Full name and signature)

THẠCH

MAI BẢO THẠCH

BẢO

HUỲNH NHẬT BẢO

PHÚC

TRẦN MINH PHÚC

TEACHER'S CONFIRMATION AND ASSESSMENT SECTION

Confirmation section of the instructors

Ho Chi Minh city, day month year
(sign and write full name)

The evaluation part of the lecturer marks the report

Ho Chi Minh city, day month year
(sign and write full name)

SUMMARY

The assignment is built for the purpose of demo fingerprint recognition system knowledge about preprocessing techniques and sift detection (minutiae algorithm), but also for consolidating knowledge after studying at school. The content of the assignment revolves around the basic knowledge of the Fingerprint recognition system of this subject. Regarding the research method, the assignment is based on the learned knowledge, lecture slides, and textbooks given by the lecturer during the learning process. The result obtained after this assignment is the knowledge that can be consolidated in the most general way as well as in place of the assignment of the 1st term during the period of extremely stressful epidemic developments.

TABLE OF CONTENT

THANKFUL WORDS	i
REPORT COMPLETED AT TON DUC THANG UNIVERSITY	ii
TEACHER’S CONFIRMATION AND ASSESSMENT SECTION	iii
SUMMARY	iv
TABLE OF CONTENT	1
LIST OF ABBREVIATION	3
LIST OF DIAGRAMS, CHARTS AND TABLES	4
CHAPTER 1: ABOUT THE PROJECT	6
1.1 Overview of the topic, assessment of the topic and resources:	6
1.1.1 Description:	6
1.1.2 Assessment:	6
1.1.3 Dataset:	6
CHAPTER 2: GENERAL THEORY ABOUT FINGERPRINT RECOGNITION	8
2.1 What is Fingerprint:	8
2.2 Fingerprint recognition:	9
2.3 Objectives:	10
CHAPTER 3: SYSTEM DESIGN	11
3.1 System level design:	11
3.2 Algorithm level design:	12
CHAPTER 4: IMAGE PREPROCESSING	14
4.1 Background removing:	14
4.2 Image enhancement:	15
4.2.1 Image sharpening:	16
4.2.2 Grayscale:	16

4.2.3 Histogram Equalization:.....	17
4.3 Image binarization and ridge detection:.....	20
CHAPTER 5: MINUTIAE EXTRACTION USING SKELETONIZING	
ALGORITHM.....	23
5.1 Skeletonizing algorithm or thinning:	23
CHAPTER 6: MINUTIAE MATCHING	
6.1 SIFT DETECTION:.....	24
6.1.1 SIFT image preprocessing:.....	24
6.1.2 Descriptors extraction:	25
6.1.3 Improved All Descriptor-Pair matching (iADM):.....	27
CHAPTER 7: CODING EXPERIMENT	
7.1 Preprocessing for images:	31
REFERENCES.....	44
APPENDIX.....	45

LIST OF ABBREVIATION

LIST OF DIAGRAMS, CHARTS AND TABLES

Image 1 Sample database templates	7
Image 2 Sample input data	7
Image 3 A sample fingerprint scan	8
Image 4 Minutiae	9
Image 5 Flowchart of algorithm.....	13
Image 6 Background removal flowchart	14
Image 7 Before and after removing background	15
Image 8 Sharpened image.....	16
Image 9 Raw image.....	16
Image 10 Grayscale image.....	17
Image 11 Sharpened image.....	17
Image 12 Example of after using HE	18
Image 13 Example of before using HE	18
Image 14 Image after using HE	20
Image 15 Grayscale image.....	20
Image 16 Image when ridge detection completed	21
Image 17 HE image.....	21
Image 18 Image after binarization and removing noise	22
Image 19 ROI image with noise	22
Image 20 Thinning image	23
Image 21 Binarized image	23
Image 22 Flow chart of image processing.....	24
Image 23 (2)	25
Image 24 (1).....	25
Image 25 (3)	25
Image 26 SIFT descriptor	26

Image 27 (4)	26
Image 28 SIFT-based Minutiae descriptor	27
Image 29 Distribution of two 1-D array in the 1 st step.....	28
Image 30 Distribution of small 2-D accumulator array in the 2 nd step.....	29
Image 31 Code	31
Image 32 Code	31
Image 33 Code	32
Image 34 Code	32
Image 35 Code	33
Image 36 Code	33
Image 37 Code	34
Image 38 Code	34
Image 39 Code	34
Image 40 Code	35
Image 41 Code	36
Image 42 Code	36
Image 43 Code	37
Image 44 Code	37
Image 45 Code	38
Image 46 Code	39
Image 47 Code	40
Image 48 Code	40
Image 49 Code	41
Image 50 Code	42

CHAPTER 1: ABOUT THE PROJECT

1.1 Overview of the topic, assessment of the topic and resources:

1.1.1 Description:

Human fingerprints are full of minutiae, which can be used as identification marks for fingerprint verification. The goal of this project is to create a complete fingerprint verification system by extracting and matching minutiae. To achieve good minutiae extraction in fingerprints of varying quality, fingerprints are preprocessed with image enhancement and binarization before being evaluated. A minutia extractor and a minutia matcher have been built using a variety of methods. For minutiae extraction, we just use sift detection that including processing stages and matching stages. Without resorting to an exhaustive search, this algorithm can find correspondences between the input minutia pattern and the stored template minutia pattern. The developed system's performance is then evaluated against a database of fingerprints from various people.

1.1.2 Assessment:

Thinning template will be stored in the database output folder, the input will be in any kind of image extensions will be processed without error or failure. The image will be background-removal and then preprocessing through some python techniques functions. After that, the system will take the input thinning image and compare with those templates in the database respectively by sift detection with the score match have to be greater than 10 matching points. The system will print out that the the input match with which thinning template in the database.

1.1.3 Dataset:

The images will be preprocessing and put into the folder for database so it will contain a few images in the folder that we declared. And the input is any images that

can take the and scan to see clearly the rigdes cause nowsaday, people often use scanner to scan and get the digital image of a fingerprint.

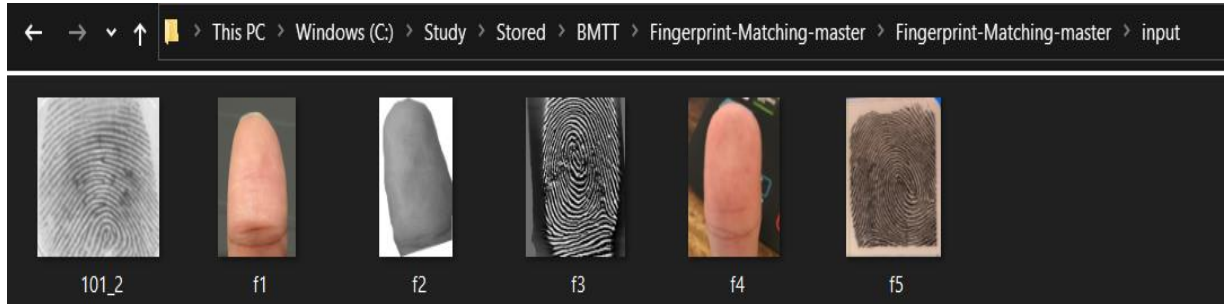


Image 2 Sample input data



Image 1 Sample database templates

CHAPTER 2: GENERAL THEORY ABOUT FINGERPRINT RECOGNITION

2.1 What is Fingerprint:

The skin on human fingertips has ridges and valleys that form distinct patterns. These patterns are fully formed during pregnancy and remain permanent for the rest of one's life. Fingerprints are prints of those patterns. Injuries such as cuts, burns, and bruises can temporarily reduce the quality of fingerprints, but patterns will be restored once fully healed.

Various studies have revealed that no two people have the same fingerprints, making them unique to each individual.



Image 3 A sample fingerprint scan

Fingerprints are widely used as biometric measurements due to the aforementioned properties. Especially in law enforcement, where they have been used to help solve crimes for over a century. Unfortunately, matching fingerprints is a complex pattern recognition problem. Manual fingerprint matching is not only time

consuming, but it also requires extensive education and training of experts. As a result, much effort has been put into developing automatic fingerprint recognition systems since the 1960s.

In forensic applications, automating the fingerprint recognition process proved to be a success. The advancements made in the forensic field have expanded the use of automatic fingerprint recognition into civilian applications. Fingerprints retain their uniqueness and permanence over time. The observations revealed that fingerprints provide more secure and reliable person identification than keys, passwords, or identification cards. Mobile phones and computers outfitted with fingerprint sensing devices for fingerprint-based password protection are being developed to replace traditional password protection methods. These are just a few of the civilian applications for fingerprints.

2.2 Fingerprint recognition:

Sir Francis Galton was the first to discover the method used for fingerprint matching. In 1888, he discovered that fingerprints are full of details, known as minutiae, in the form of discontinuities in ridges. He also noticed that the position of those minutiae remains constant over time. As a result, minutiae matching is a good way to determine whether two fingerprints are from the same person or not.

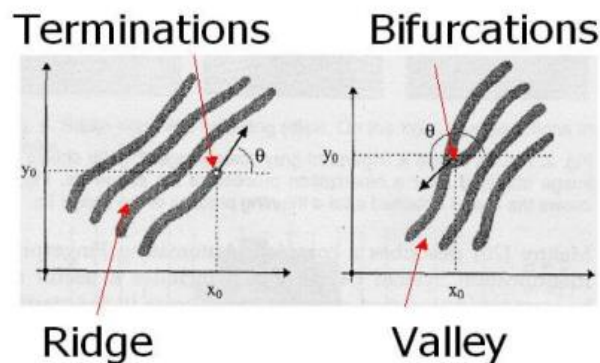


Image 4 Minutiae

The two most important minutiae are termination and bifurcation. Termination is the point on a ridge from which two branches emerge; bifurcation is the point on the ridge from which two branches emerge.

The fingerprint recognition problem is divided into two sub-domains: fingerprint verification and fingerprint identification.

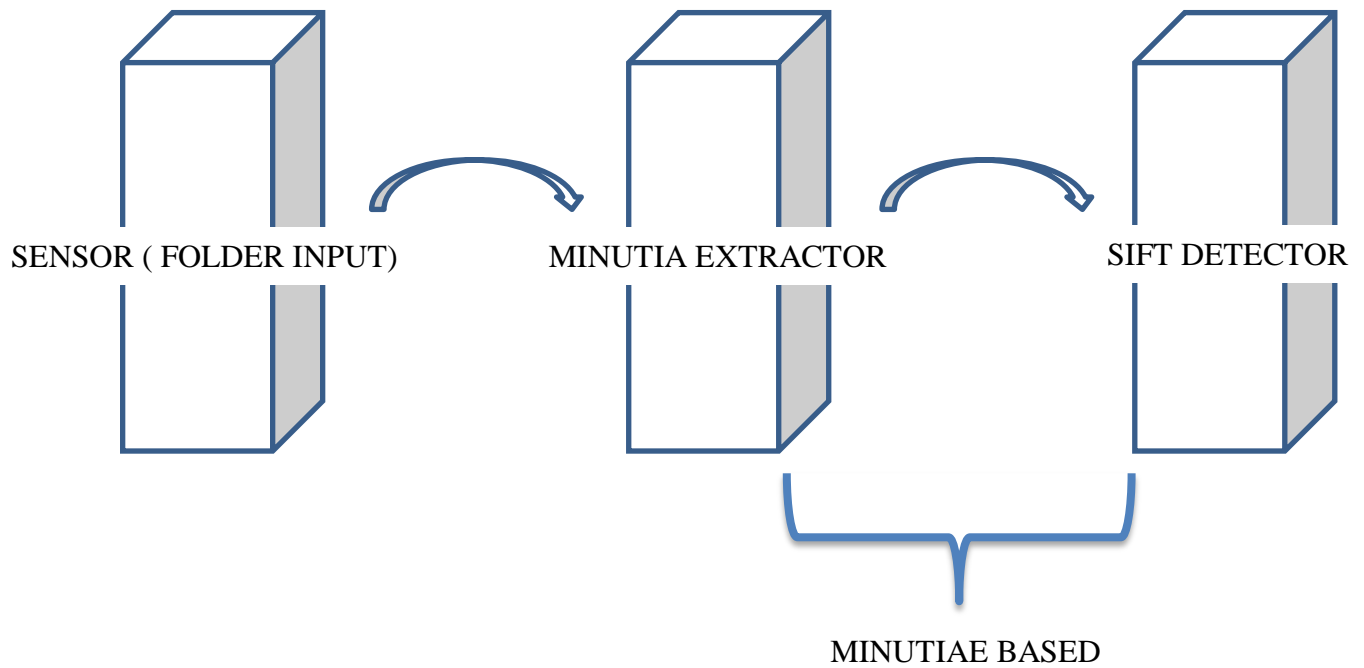
2.3 Objectives:

The goal is to put in place a fingerprint recognition algorithm. After removing the background of each fingerprint image, the preprocessing stages will be implemented to enhance the images. The SIFT-based concept is used to extract the minutia descriptor, which is then followed by the matching technique called iADM (improved all Descriptor-Pair matching).

CHAPTER 3: SYSTEM DESIGN

3.1 System level design:

A fingerprint recognition system is made up of three components: a fingerprint acquiring device, a minutia extractor, and a minutia matcher.



The minutia extractor and minutia matcher modules are detailed in the following section for algorithm design and other subsequent sections.

3.2 Algorithm level design:

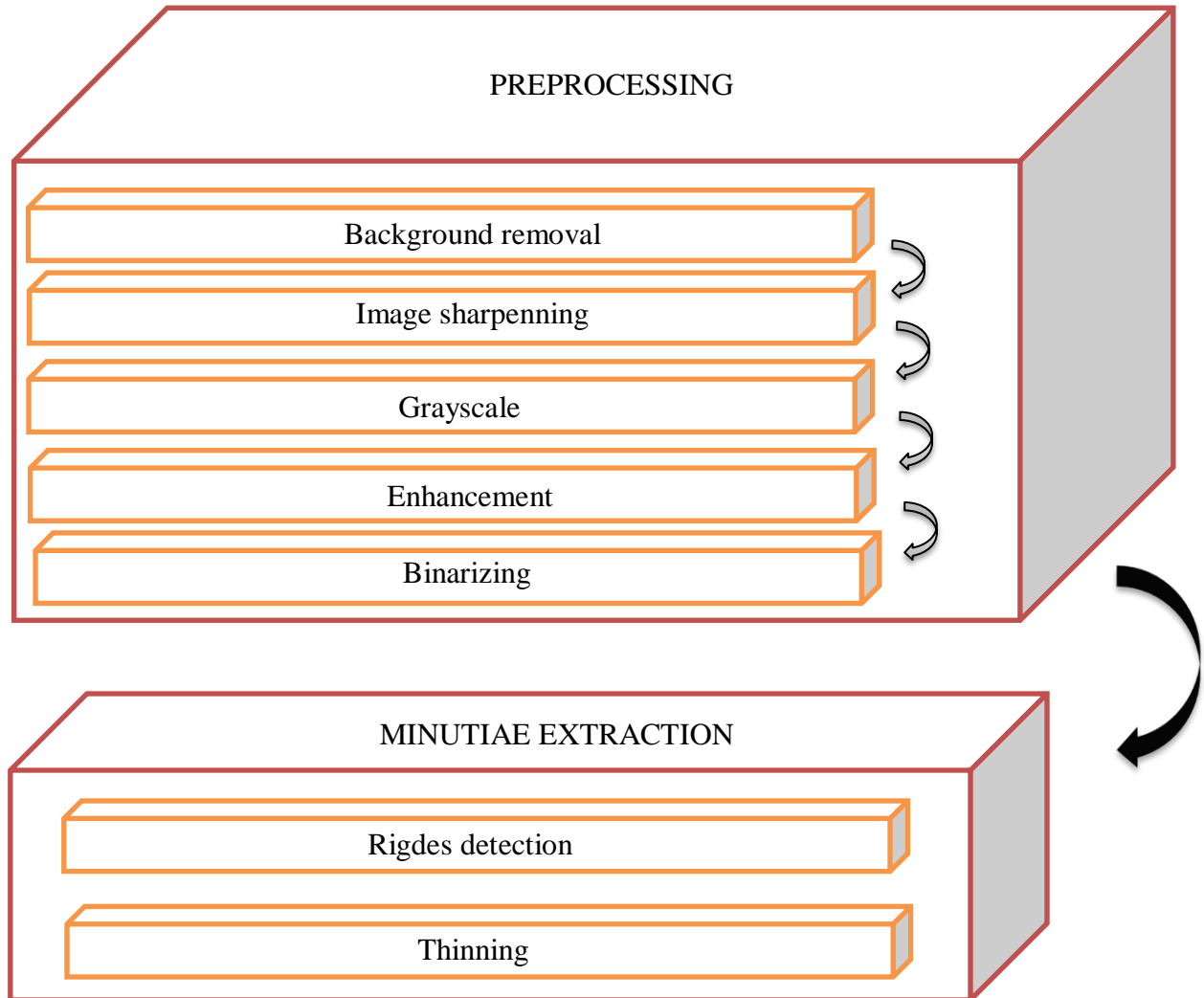


Image enhancement techniques such as Histogram Equalization and Fourier Transform were used during the fingerprint image preprocessing stage. The fingerprint image is then binarized with the locally adaptive threshold method. Moreover, we will remove the background by some library in python.

The iterative parallel thinning algorithm is used for the minutia extraction stage. The Hessian matrix algorithm will be used to detect ridges.

For matching stages, we will use the SIFT detection for marking bifurcation and termination in each input and database's templates. Each matched pair will be viewed in green line and if they overcome above 10 features, the system will provide the notification for matching.

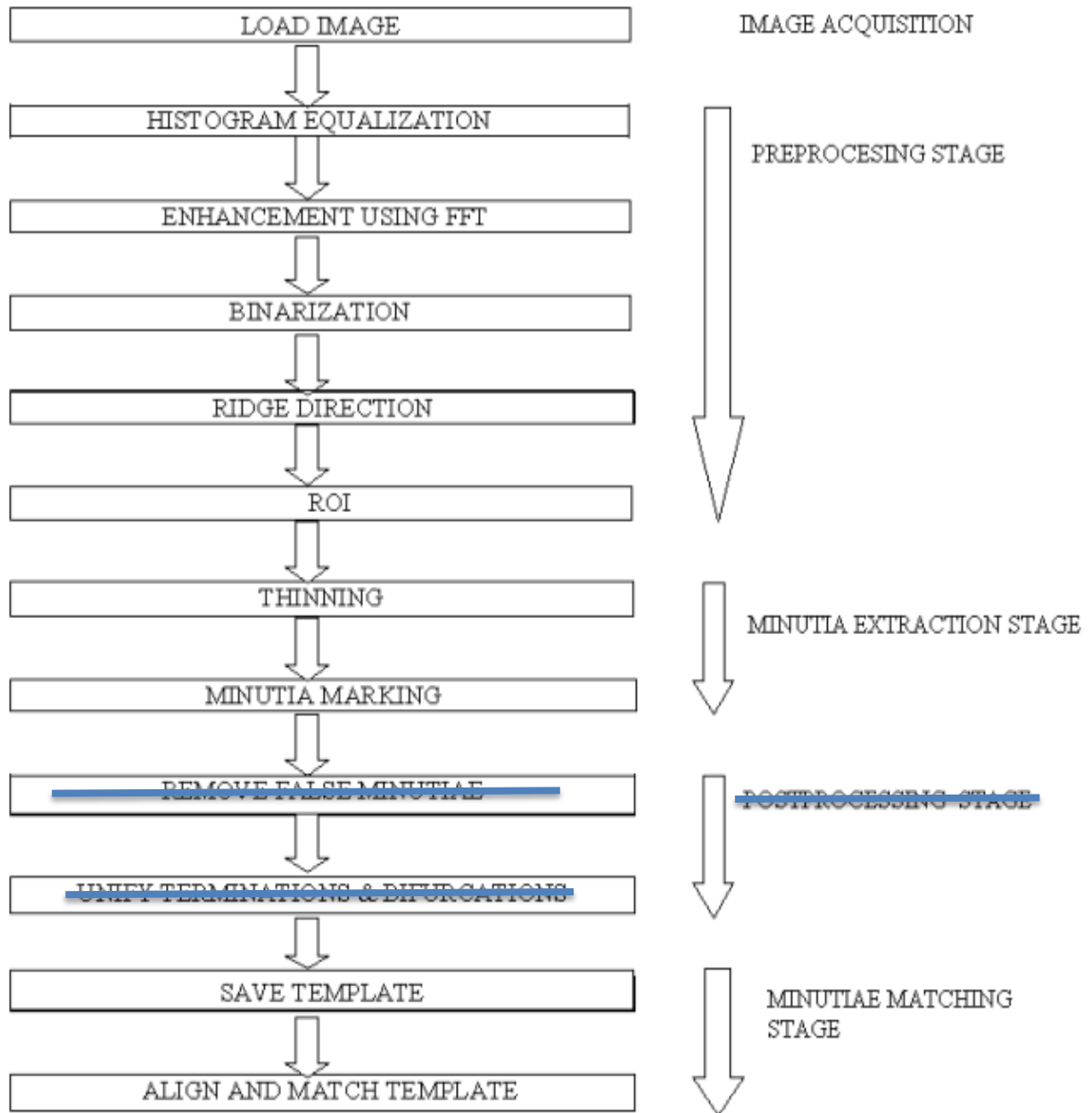


Image 5 Flowchart of algorithm

CHAPTER 4: IMAGE PREPROCESSING

4.1 Background removing:

Background removal is a digital image processing technique that can be used to categorize parts of an image as unwanted or interesting. Many image processing and computer vision applications necessitate background removal prior to further analysis and processing.

Because of the accuracy of detection when matching stages are in process, removing background is crucial when preprocessing an image of fingerprint.

We have use **OpenCV** is an image processing library for python. It is the most popular library for image processing and computer vision tasks because it is open source and very fast.

Foreground detection is a technique that detects changes in the image sequence. The background subtraction method is used here to separate the foreground from an image.

Background removal, also known as fingerprint segmentation, extracts a segmented mask of the image's foreground. The term "foreground" refers to the subject of the image; in this case, humans are in the foreground, and everything else is in the background.

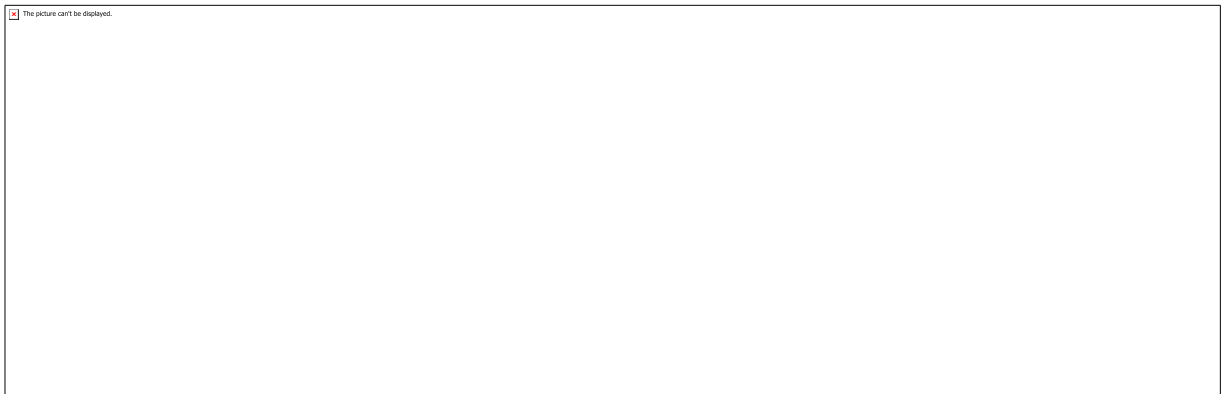


Image 6 Background removal flowchart

Practical result of removing background in our project:



Image 7 Before and after removing background

4.2 Image enhancement:

The purpose of fingerprint image enhancement is to make the image clearer for easier subsequent operations. Because fingerprint images acquired from sensors or other media are not guaranteed to be of perfect quality, enhancement methods such as increasing the contrast between ridges and furrows and connecting false broken points of ridges due to insufficient ink are very useful for maintaining a higher accuracy in fingerprint recognition.

We will use Histogram Equalisation method and convolution kernel for sharpening the images.

4.2.1 Image sharpening:

Image sharpening is a technique used to make digital images appear sharper. Sharpening improves the definition of an image's edges. The dull images are those that have poor edges. The background and edges are nearly identical.

First define a custom 2D kernel, and then use the `filter2D()` function to apply the convolution operation to the image. In the code below, the 3×3 kernel defines a sharpening kernel. Check out this resource to learn more about commonly used kernels.

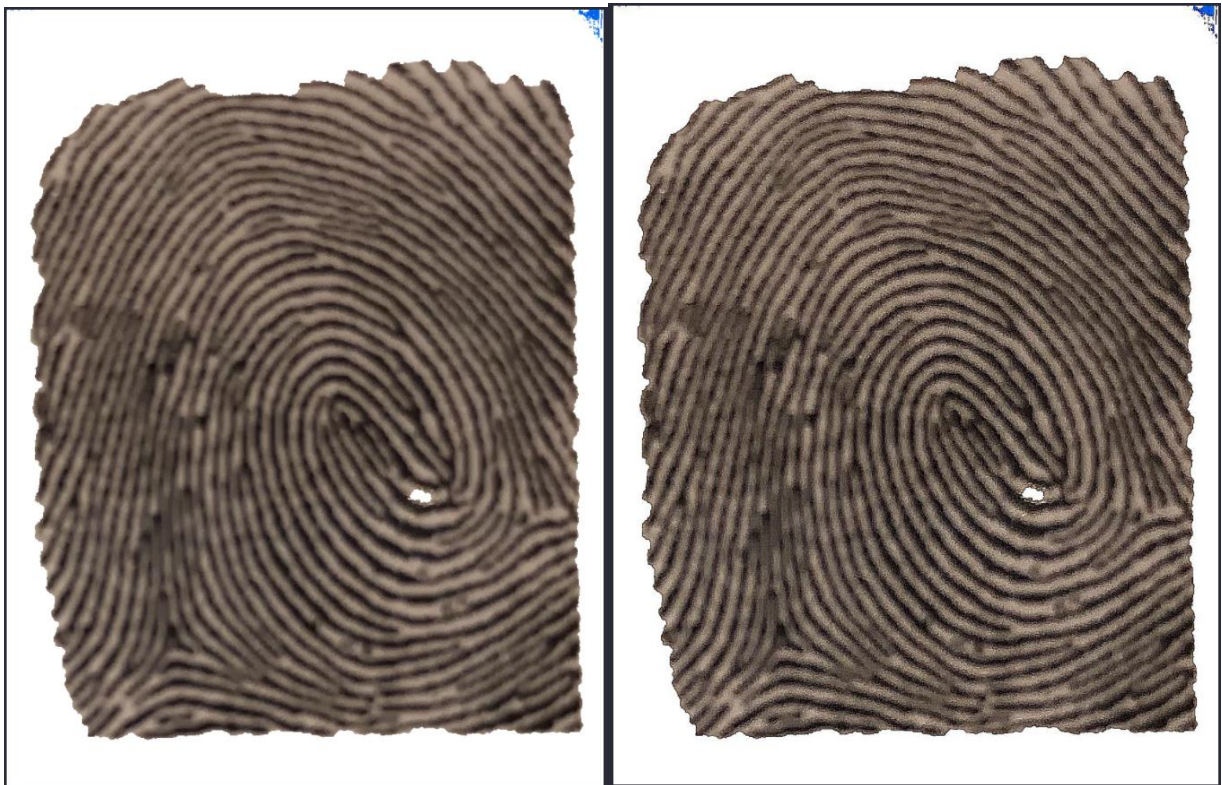


Image 9 Raw image

Image 8 Sharpened image

4.2.2 Grayscale:

Grayscale in digital images means that the value of each pixel only represents the intensity of the light. Typically, such images show only the darkest black to the brightest white. In other words, the image only contains black, white, and gray colors, each of which has multiple levels of gray. We want to turn the image into the color scale that

from black to white for the accuracy in matching. The technique will be done by `cv2` library named `cvtColor`.



Image 11 Sharpened image



Image 10 Grayscale image

4.2.3 Histogram Equalization:

Histogram equalization is the process of expanding an image's pixel value distribution in order to increase perceptual information. The original histogram of a fingerprint image is bimodal; after histogram equalization, the histogram occupies the entire range from 0 to 255, and the visualization effect is enhanced.

Consider an image in which the pixel values are limited to a specific range of values. A brighter image, for example, will have all pixels confined to high values. A good image, on the other hand, will contain pixels from all regions of the image. As a result, you must stretch this histogram to either end.

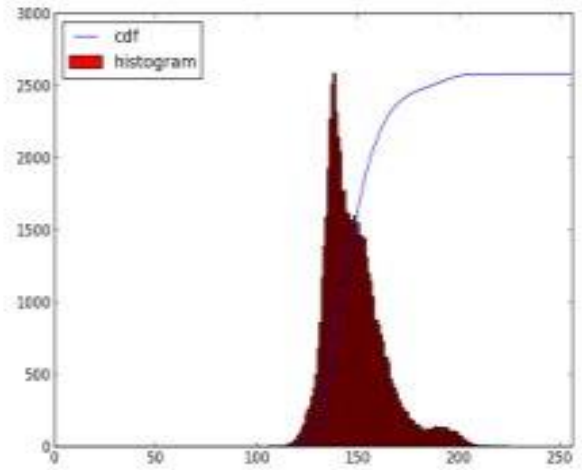


Image 13 Example of before using HE

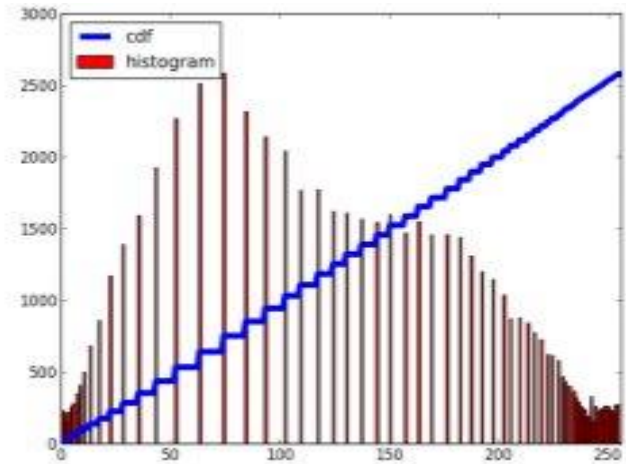


Image 12 Example of after using HE

Using Histogram Equalization in enhancing fingerprint image, and the equation that we use to code is based on the general formula of Cumulative Distribution function:

Consider a discrete gray scale $\{x\}$ and let n_i be the number of occurrences of gray level i . The probability of an occurrence of a pixel of level i in the image is:

$$p_x(i) = p(x = i) = \frac{n_i}{n}, \quad 0 \leq i < L$$

L being the total number of gray levels in the image (typically 256), n being the total number of pixels in the image, and $p_x(i)$ being in fact the image's histogram for pixel value I , normalized to $[0,1]$.

Let us also define the CDF corresponding to I as

$$\text{cdf}_x(i) = \sum_{j=0}^i p_x(x = j),$$

which is also the image's accumulated normalized histogram.

We would like to create a transformation of the form $y = T(x)$ to produce a new image $\{y\}$, with a flat histogram. Such an image would have a linearized cumulative distribution function across the value range.

$$\text{cdf}_y(i) = (i + 1)K \text{ for } 0 \leq i < L$$

for some constant K . The properties of the CDF allow us to perform such a transform, it is define as:

$$y = T(k) = \text{cdf}_x(k)$$

where k is in range $[0, L-1]$. Notice that T maps the levels into range $[0,1]$, since we used a normalized histogram of $\{x\}$. In order to map values back into their original range, the following simple transform needs to be applied on the results:

$$y' = y \cdot (\max\{x\} - \min\{x\}) + \min\{x\} = y \cdot (L - 1).$$

y is a real value while y' has to be an integer. An intuitive and popular method is applying the round operation:

$$y' = \text{round}(y \cdot (L - 1)).$$

However, detailed analysis results in slightly different formulation. The mapped value y' should be 0 for the range of $0 < y \leq 1/L$. And $y' = 1$ for $1/L < y \leq 2/L$, $y' = 2$ for $2/L$

$< y \leq 3/L$,, and finally $y' = L - 1$ for $(L - 1)/L < y \leq 1$. Then the quantization formula from y to y' should be:

$$y' = \text{ceil}(L \cdot y) - 1.$$

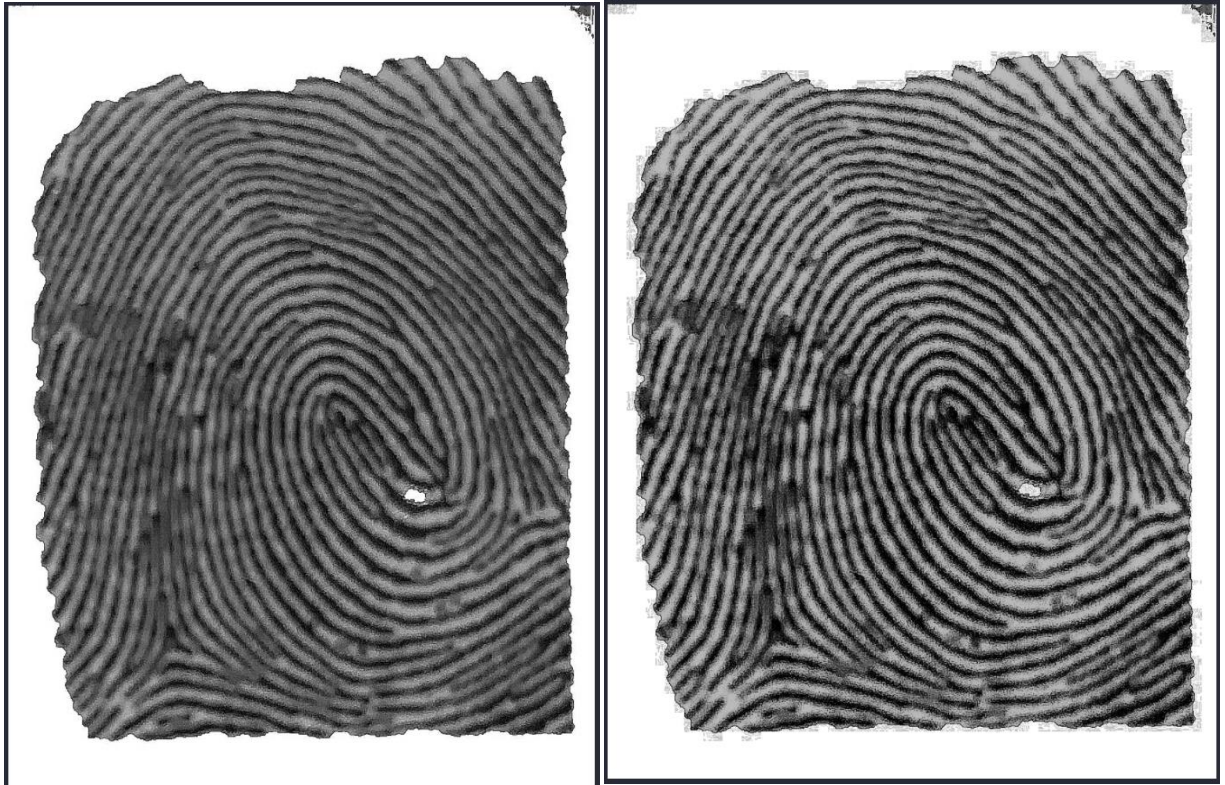


Image 15 Grayscale image

Image 14 Image after using HE

4.3 Image binarization and ridge detection:

Fingerprint Image Binarization is the process of converting an 8-bit grayscale fingerprint image to a 1-bit image with 0 for ridges and 1 for furrows. Following the operation, ridges in the fingerprint are highlighted in black, while furrows are highlighted in white.

To binarize the fingerprint image, a locally adaptive binarization method is used which called adaptive Threshold algorithm. This method gets its name from the

mechanism that converts a pixel value to 1 if it is greater than the mean intensity value of the current block (16x16) to which the pixel belongs.

But first, we will use Hessian matrix algorithm operations for to detect the rigdes of the fingerprint images. We provide a hessian_matrix function of skimage library an grayscale image and then return a list which contains ndarray that we have define as an image. After that, we will use to variable to contain the most contrast among rigdes and valleys from the matrix and the least contrast to plot up 2 image with 2 different constrast, this will help us to reduce less noise of the images. The result of this process will provide us an image with ROI.



Image 17 HE image



Image 16 Image when rigde detection completed

Next step is to binarize the image for easy to extract minutiae and turn the image to the most efficiently color to extract which is black and white. The binarization will follow 4 steps in total. First is dilation the image with max contrast, then Gaussian Blur

will be applied for reducing noise of the image containing the background of it. After that, we will detach the source with the background and get the distinct source image. Finally, using a threshold to transform the the pixel to the max value when they are higher than that limitation and the other pixel will be replaced with 0.

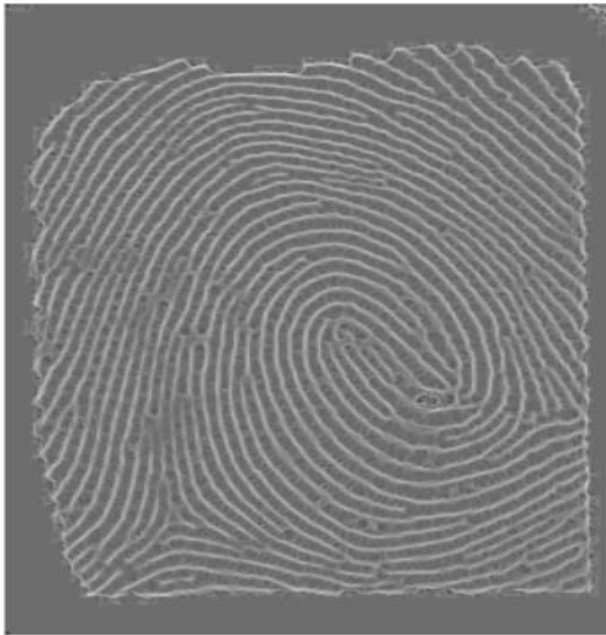


Image 19 ROI image with noise



Image 18 Image after binarization and removing noise

CHAPTER 5: MINUTIAE EXTRACTION USING SKELETONIZING ALGORITHM

5.1 Skeletonizing algorithm or thinning:

Ridge Thinning is the process of removing redundant pixels from ridges until the ridges are only one pixel wide. The thinning algorithm is iterative and parallel. The algorithm marks down redundant pixels in each small image window during each scan of the full fingerprint image (3x3). After several scans, it finally removes all of the marked pixels.



Image 21 Binarized image



Image 20 Thinning image

The thinned ridge map is then filtered by other three Morphological operations to remove some H breaks, isolated points and spikes.

CHAPTER 6: MINUTIAE MATCHING

At this matching stages, with the templates we store in the database and the image of minutiae that we have extracted, sift based will be including throughout this process.

6.1 SIFT DETECTION:

6.1.1 SIFT image preprocessing:

When there are differences in finger pressure or skin characteristics, the SIFT descriptor becomes unstable. As a result, grayscale fingerprint images that have not been pre-processed are unsuitable for original SIFT extraction. Filters are used to enhance the grayscale image of the original fingerprint image. The image below depicts the image processing flow used by FISiA. It is divided into four major stages: highpass filtering, lowpass filtering, ridge direction detection, and ridge enhancement.



Image 22 Flow chart of image processing

The highpass filter is used to calibrate the brightness. If the gray value of the image at position (x,y) is denoted by $I(x,y)$, Equation (1) is used to calculate the highpass filter $I_H(x,y)$, with the size of the highpass window k set to 16 and the bias value b set to 128. It computes the average intensity within the $k \times k$ window and subtracts it from the average of the center pixel biased at b . The lowpass filter $I_L(x,y)$ is used to reduce noise, as shown in Equation (2), with the sizes of the two lowpass windows m and n set to 4 and 2, respectively.

Because the pixel in the smaller window is more similar to the target pixel, the lowpass filter employs two windows and the average of two averages. After performing accurate ridge detection for each pixel with a look-up table, the ridge direction detection for each block (8×8) can be estimated. To enforce the fingerprint pattern, ridge

enhancement with a Gabor-like filter can be performed. It removes low frequency components in the orthogonal direction to the ridge direction.

$$I_H(x, y) = I(x, y) - \frac{1}{k^2} \sum_{i=1}^k \sum_{j=1}^k I(i, j) + b$$

Image 24 (1)

$$I_L(x, y) = \frac{1}{2} \left(\frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m I_H(i, j) + \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n I_H(i, j) \right)$$

Image 23 (2)

6.1.2 Descriptors extraction:

Binarization and thinning are performed based on the image processing from the previous sub-section. The thinning image is used to detect minutiae. There are two types of minutiae: ridge bifurcation and ridge ending. A ridge ending minutia is a point at which a ridge ends, whereas a ridge bifurcation minutia is a point at which a ridge splits into two paths. Equation (3) defines the minutia m , which includes its x coordinate, y coordinate, and direction by tracing.

$$D_M(m) = (x_m, y_m, \theta_m)$$

Image 25 (3)

Section 6.1 calculates the SIFT descriptor based on the processed image. The skeleton image should not be used to extract minutiae because it lacks the texture information required by the SIFT operator. The gradient magnitude and orientation at each point in the region around the sampling point are computed to generate a SIFT descriptor, as shown on the left of the image below. These samples are then aggregated

into orientation histograms that summarize the contents across the sub-regions, as shown on the right of the image below. The proposed descriptor consists of the minutia's SIFT descriptor and SIFT descriptors at various sampling points around the minutia.

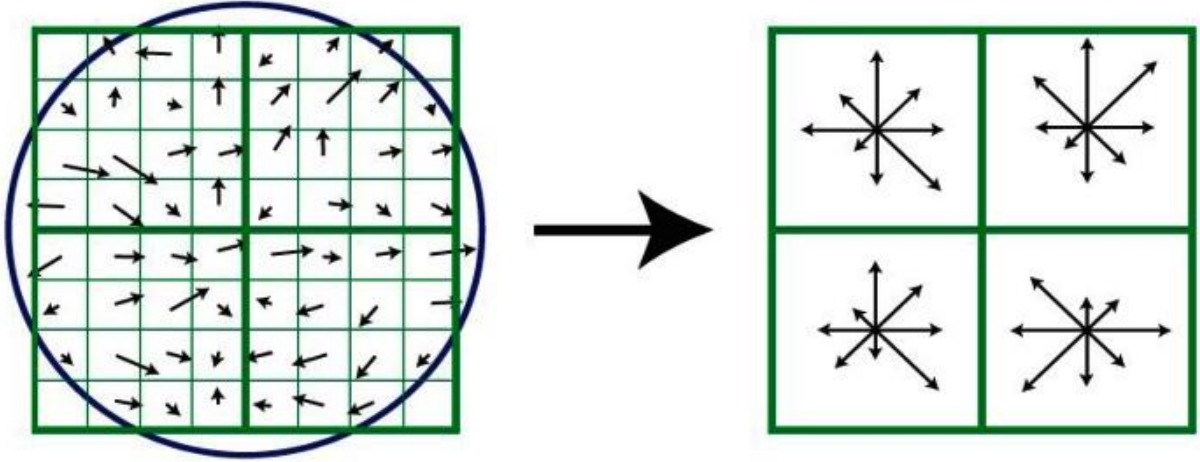


Image 26 SIFT descriptor

Let m and p_i represent the detected minutia and the sampling point, respectively. The detected minutia descriptor is defined as follows:

$$D(m) = \{S(m), \{S(p_i)\}_{i=1}^d\}$$

Image 27 (4)

where $S(m)$ is the SIFT descriptor of minutia, d is the number of sampling points, and $S(p_i)$ is the SIFT descriptor of sampling points surrounding minutia m . The structure is shown below when d equals 4.

In previous work, we demonstrated that the SIFT descriptor around minutiae points plays a significant role when the SIFT algorithm is applied to fingerprint verification. As a result, we believe that the texture information of the region surrounding minutia can be included in this descriptor using this definition. The choice of parameter d represents a tradeoff between verification accuracy and speed. Over 2,000 key points can be extracted from a fingerprint image using the original SIFT extraction. Both

extraction and verification will necessitate significant computational resources. According to our findings, the value of d is 4. The proposed extraction has $5M$ SIFT descriptors, where M denotes the number of minutiae in the fingerprint image. In other words, the number of SIFT descriptors in a single fingerprint image is only about 50200, which significantly reduces computation complexity.

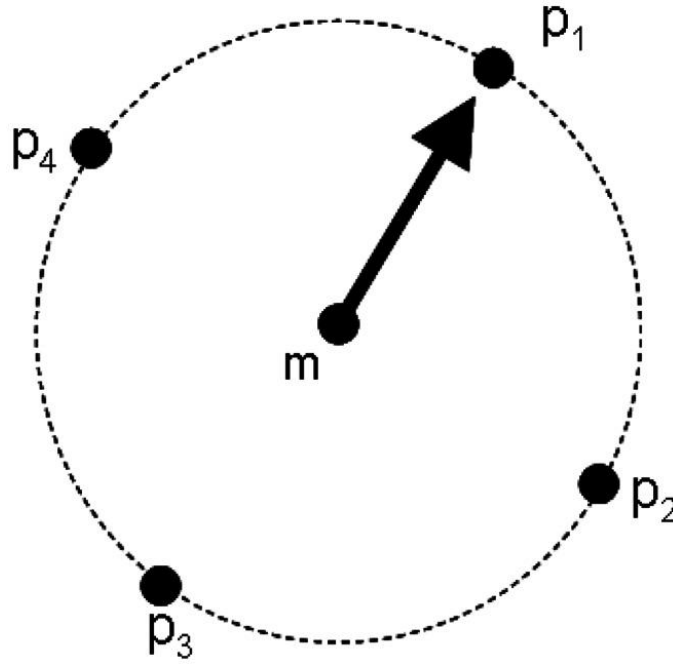


Image 28 SIFT-based Minutiae descriptor

6.1.3 Improved All Descriptor-Pair matching (iADM):

Using the proposed SIFT-based minutia descriptor (SMD), we developed a two-step fast matching method, called improved All Descriptor-Pair Matching (iADM).

The optimized ADM performs well for accurate matching, but it may still suffer from computation time constraints when one-to-many verifications are required. As a result, based on the optimized ADM, we propose the improved All Descriptor-Pair Matching (iADM), a two-step fast matching method that combines global and local search.

The optimized ADM is only performed in the first step based on the SIFT descriptor of minutiae positions. In other words, in Equation, only the $S(m)$ of $D(m)$ is used (4). This step yields the peak of the Hough Transform corresponding to the best alignment transformation (x, y, θ) . The rough alignment transformation produced this result, which will be used for local search. Because the SIFT descriptor of minutiae positions is only $N/5$, where N is the number of SIFT points, the Euclidean Distance is calculated $(N^2/25)$ times. In practice, a more efficient method can be used. Instead of using a 2-D accumulator array, two 1-D accumulator arrays, BX and BY , are used as the displacement vector's histogram arrays. These histograms accumulate scores for each axis, x and y , as shown in the image below. The distribution score, C_x and C_y , can then be calculated by analyzing these histograms. The simplest method is to extract the locations with the highest histogram values. The best rotation can also be obtained by using Optimized ADM to repeat all possible rotation steps. There are two advantages to using two 1-D arrays. It can shorten the time spent searching for a peak in the accumulator array. Furthermore, the memory requirement can be reduced.

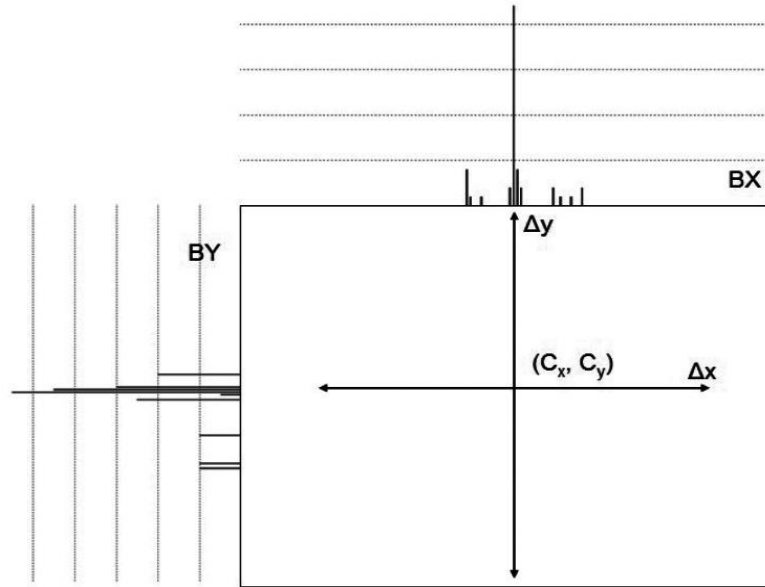


Image 29 Distribution of two 1-D array in the 1st step

The local search is used in the second step. For local matching, only the SIFT descriptor of sampling points around minutiae is used. Similarly, for all closest pairs calculated by Euclidean Distance in this local area, the matching is performed by employing the Hough Transform. To accumulate the score, a small 2-D accumulator array C with (C_x, C_y) as the center is used. The Euclidean Distance is not required for descriptor pairs with displacement vectors outside of C . Notably, each minutia has four sampling SIFT descriptors that are rotation invariant. As a result, the sampling SIFT descriptor only matches descriptors from its own class. $S0(m0)$, $S0(p1)$, $S0(p2)$, $S0(p3)$, for example, belong to one minutia of Image #0, whereas $S1(m0)$, $S1(p1)$, $S1(p2)$, $S1(p3)$ belong to one minutia of Image #1. $S0(p1)$ must only match with $S1(p1)$; the Euclidean Distance computation of $(S0(p1), S1(p2))$ and $(S0(p1), S1(p3))$ is not required. The image below depicts the distribution of scores in the array C . The peak value is found in this distribution of scores and used as the final matching score. The size of C in our experiment is set to 16.

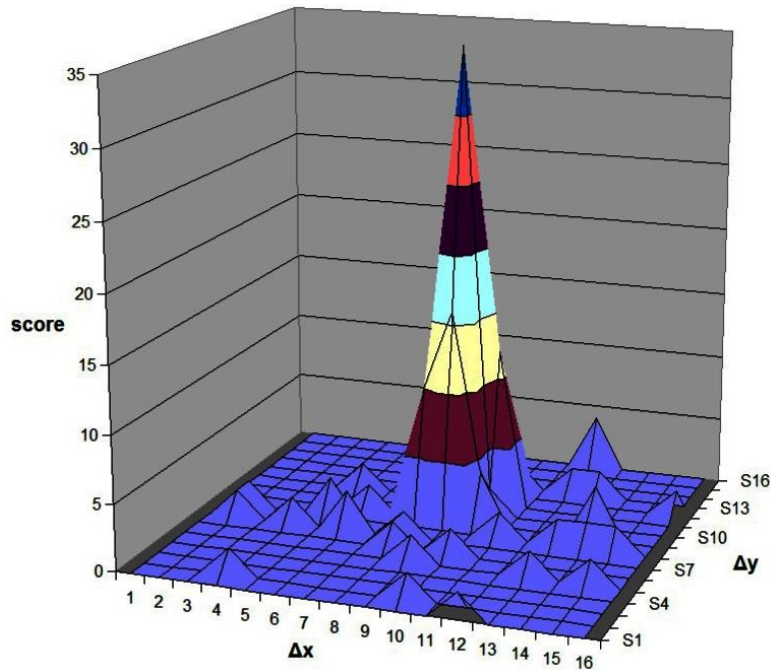


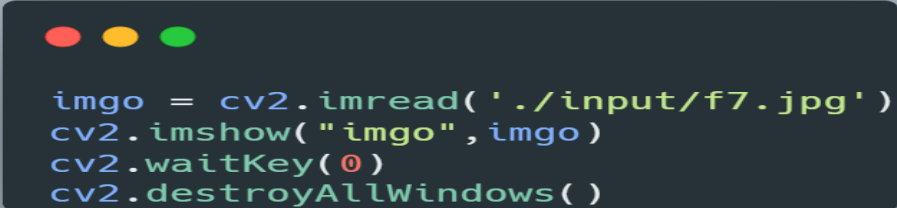
Image 30 Distribution of small 2-D accumulator array in the 2nd step

Finally, in coding environment, we set up a threshold that if the point overcome it, the input fingerprint will match with the template in the database.

CHAPTER 7: CODING EXPERIMENT

We will divide the code into two part for explanation.

7.1 Preprocessing for images:



```
imgo = cv2.imread('./input/f7.jpg')
cv2.imshow("imgo", imgo)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Image 31 Code


After import essential library, we will input the data that need to match, all the data will be stored in the input directory and the `imread()` function is returning the image through the path parameter. The `imshow()` function is to plot the image with the window form and other two functions are used to keep the window open until we close it.



```
height, width = imgo.shape[:2]
print(height,width)
```

Image 32 Code


This block of code is used to show the size of the image to let us know the height and width of the picture. And this is the first step to remove the background of the image.



```
mask = np.zeros(imgo.shape[:2],np.uint8)
print(mask)
```

Image 34 Code

This block code will turn all the array with the shape given by the function shape() and turn all the value of the array into zero for binarization later of the process, the parameter for the zeros() function is the shape of input and the data type which is uint8 meaned int with 8 bits long.

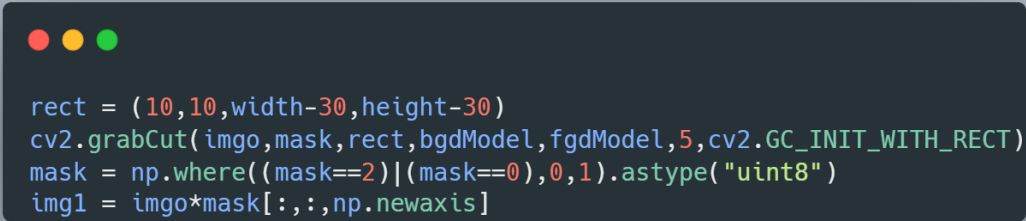


```
bgdModel = np.zeros((1,65),np.float64)
fgdModel = np.zeros((1,65),np.float64)
print(bgdModel)
print(fgdModel)
```

Image 33 Code

The above block of code is used to show what the shape of the original image is, now we will config the background shape and feature shape (which is the fingerprint), both will have an array shape with 1 for height and 65 for width and the data type is float

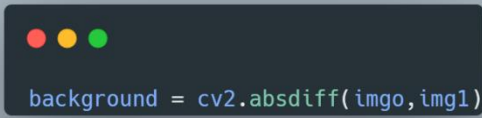
with 64 bit (enough to contain 65 zero values). This stage is call grab cut the object.



```
rect = (10,10,width-30,height-30)
cv2.grabCut(imgo,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)
mask = np.where((mask==2)|(mask==0),0,1).astype("uint8")
img1 = imgo*mask[:, :, np.newaxis]
```

Image 35 Code

We have define mask that hold the foreground and two array for the algorithm to execute. Now we continually define a rectangle for get the foreground image and ignore the background with the size in the code. After that, calling the function grabCut() to cut the rectangle with 5 iterations and provide it with full parameter that we have define before and the mode we use is GC_INIT_WITH_REC which will draw the rectangle. Finally, So we modify the mask such that all 0-pixels and 2-pixels are put to 0 (ie background) and all 1-pixels and 3-pixels are put to 1(ie foreground pixels). When we get the final mask image, multiply it with the original image, the segmented image will born. The mask[:, :, np.newaxis] mean that from the 2d array, we increase by one for the dimension and become the 3d array.




```
background = cv2.absdiff(imgo,img1)
```

Image 36 Code

To get the background we will use the absdiff() function to define a minus equation the differences between first parameter and the second one. In this case, we

minus the original array (original image) by the foreground array that we get after cut (segmented image) and we will get the difference which is the background array (background image).



```
background[np.where((background > [0,0,0]).all(axis = 2))] = [255,255,255]
```

Image 38 Code


We know that the background have random color so we will turn all the color that are not black to white. And then attach the white background with the segmented image following by this code which is the combination of two array.



```
final = background + img1
```

Image 37 Code

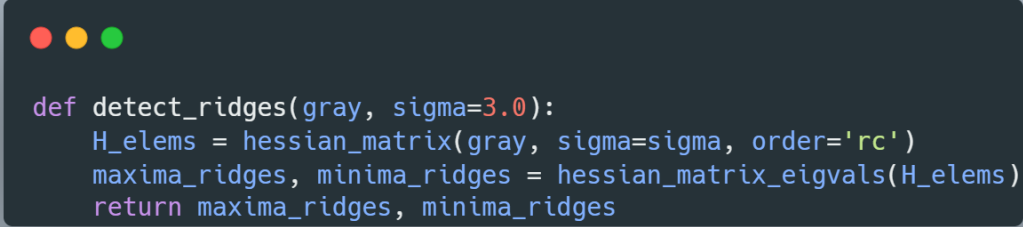
Finally, viewing the result through imshow() function and export the image for the next step of processing.



```
cv2.imshow('image', final )  
cv2.imwrite("input.jpg",final)
```

Image 39 Code

The next content we will go through is define 2 function which detect the ridge of the fingerprint image (ROI) and plot those image respectively.



```
def detect_ridges(gray, sigma=3.0):  
    H_elems = hessian_matrix(gray, sigma=sigma, order='rc')  
    maxima_ridges, minima_ridges = hessian_matrix_eigvals(H_elems)  
    return maxima_ridges, minima_ridges
```

Image 40 Code

This function will need to be provided two parameters, one is the gray images and the other is sigma. We will use the `hessian_matrix` to compute and return the list of ndarray which contain upper-diagonal elements of the hessian matrix for each pixel of the input image. The parameters will be the image, the standard deviation for computing and the order 'rc' is mean using the first axis of the gradient computing. After finishing the computing, we will get the max contrast and the least contrast of the list ndarray which mean from highest array to the lowest array of the image. The outputs are clearly be the two ndarray of the image, one is the most clearly and the other is in contrast.

```

def plot_images(*images):
    images = list(images)
    n = len(images)
    fig, ax = plt.subplots(ncols=n, sharey=True)
    for i, img in enumerate(images):
        ax[i].imshow(img, cmap='gray')
        ax[i].axis('off')
        extent = ax[i].get_window_extent().transformed(fig.dpi_scale_trans.inverted())
        plt.savefig('fig'+str(i)+'.png', bbox_inches=extent)
    plt.subplots_adjust(left=0.03, bottom=0.03, right=0.97, top=0.97)
    plt.show()

```

Image 41 Code

This function has the input is images which “*” let us pass the number of arguments into the function. We will first put all the images into a list variable, then get the length of the list. Defining two variables for set up the images that will be plotted later. Running a loop through the the list of images and show the images, we just need the image so the axis must be off. The “extent” variable is used to transform the height and weight into inches and then we will save that images by savefig() function with parameters are the name of figure we save and the container that we define in inches.

The end of the function is adjusting the position of the figure and show it on the screen.

Now, after finishing the removing background stage and define function for preprocessing, we will toward into the ehancement stages which contains 6 steps in total.

```

img = cv2.imread("input.jpg",1)

```

Image 42 Code

First step is loading the image that we have removed the background and put a flag variable equal to 1 to make it come with color of it.

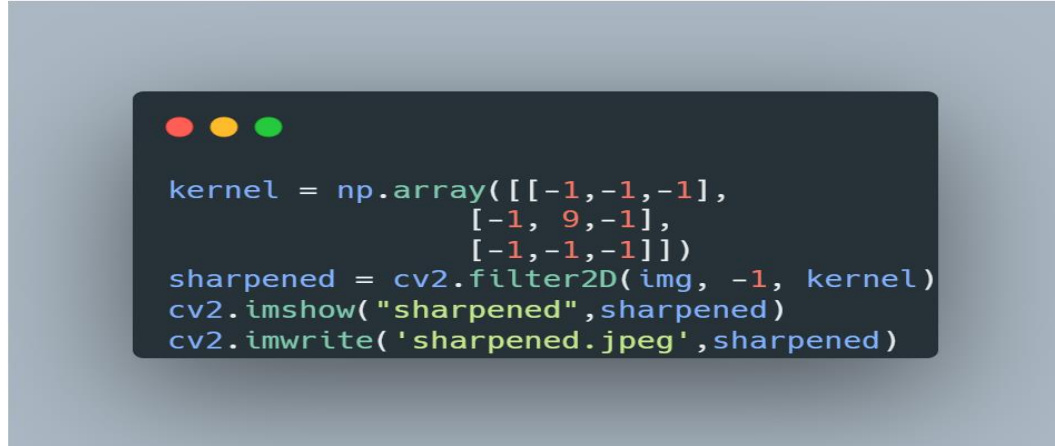


Image 43 Code

Second step is sharpening the image. We need to define a kernel (2d array), because we want to enhance the edge to clearly see the ridges and valleys. All the negative values will be multiplied with the background pixel and positive value will make the edges clearer. Using filter2D() function and passing the source image, kernel for edge detection and sharpening with “-1” which is the similar depth to the source image, the result will be the image with clearer edges.



Image 44 Code

Third step is converting the sharpened image to a grayscale color for unifying the basic color and for later uses. Using `cvtColor()` function with the input image after sharpening stage and the color space conversion code, in this case we turn it to gray color.

```

hist,bins = np.histogram(img.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * hist.max() / cdf.max()

cdf_m = np.ma.masked_equal(cdf,0)
cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
cdf = np.ma.filled(cdf_m,0).astype('uint8')

img2 = cdf[gray]

plt.plot(cdf_normalized, color = 'black')
plt.hist(img.flatten(),256,[0,256], color = 'red')
plt.xlim([0,256])
plt.ylim([0,80000])
plt.legend(('cdf','histogram'))
plt.show()

hist,bins = np.histogram(img2.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * hist.max() / cdf.max()

plt.plot(cdf_normalized, color = 'black')
plt.hist(img2.flatten(),256,[0,256], color = 'red')
plt.xlim([0,256])
plt.ylim([0,80000])
plt.legend(('cdf','histogram'))
plt.show()

```

Image 45 Code

Fourth step is performing Histogram Equalization to make the image become clearer. To perform this algorithm, we need to use correct format for calculation of the cdf. Flattenning the image is crucial because the histogram() function needs an flatten image, after computing the histogram value will be store in an array along with the array of edge. Function cumsum() is used to sum cumulatively the elements (indexes) in the array. And the cdf_normalized will be compute as the above equation, after that we will use the np.ma.mask_equal() function to remove the given mask (in this case is zero). Then we will filled the mask array with 0 to get the array after computing.

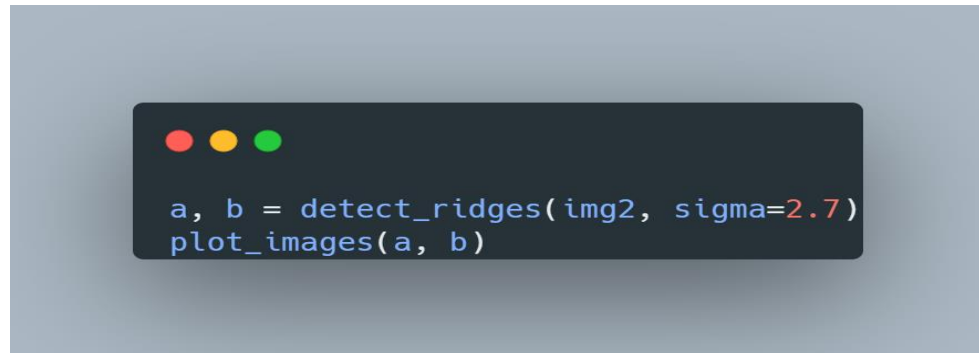


Image 46 Code

As we define the two function `detect_ridges()` and `plt_images()`, we will put the fingerprint image which was enhanced by Histogram Equalization algorithm into the first function to extract the ROI (Region of Interest) as the fifth step. Parameter sigma is just the variable for computing the hessian matrix, which will optimizing the function of interest. After that will save two images and plot them onto the screen with the second function.



Image 48 Code

Sixth stage is binarizing the image, we will use the most contrast figure as (the clearest ridge detection above). Dilate() function use to increase the minutiae area which is the white region of the fingerprint. After that, we will blur the image by a function called GaussianBlur() to reduce the noise of the image. After all, threshold() function will return two outputs, one is the threshold that we have used, other is the image that is binarized.

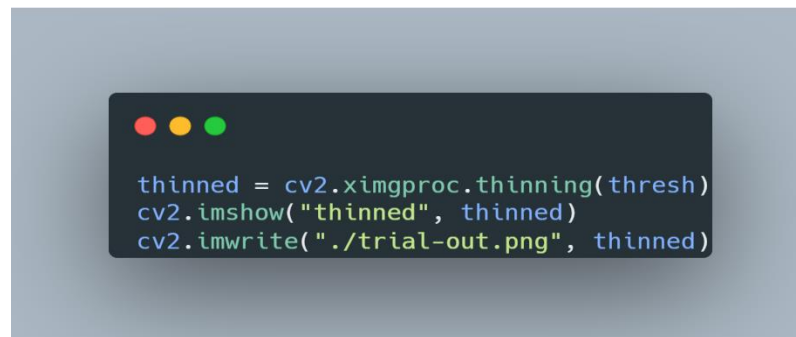


Image 47 Code

Now we will extract the minutiae which is all the ridges like we take an x-ray at the hospital. We will use the ximgproc.thinning() function to transform the binary image to skeletonizing image for marking with sift detection algorithm a matching stage.

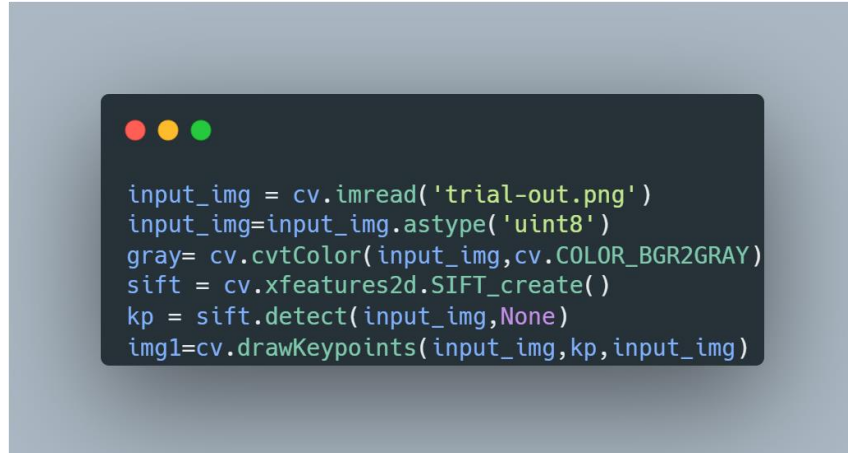


Image 49 Code

The final content we will through now is using SIFT detection for preprocessing for matching input and template in the database. First we will get the inpt image that we had preprocessed before at those previous stages, because the image is an array so we have to change all the value and array's elements into int type with 8 bits long. Because sift preprocessing need grayscale image so we have to convert it into gray color. Xfeatures2d.SIFT_create() is used for create an sift detector, then we will put the input image into the detector. The function detect() will find the keypoint base on the orientation of the ridges and mark it with function drawKeypoints().

```

os.chdir("./output")
for file in glob.glob("*.png"):

    frame=cv.imread(file)
    frame=frame.astype('uint8')
    gray1 = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    sift = cv.xfeatures2d.SIFT_create()
    kp = sift.detect(frame,None)
    img2=cv.drawKeypoints(frame,kp,frame)
    kp1, des1 = sift.detectAndCompute(img1,None)
    kp2, des2 = sift.detectAndCompute(img2,None)

    FLANN_INDEX_KDTREE = 0
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks = 50)
    flann = cv.FlannBasedMatcher(index_params, search_params)
    matches=flann.knnMatch(np.asarray(des1,np.float32),np.asarray(des2,np.float32),k=2)
    good = []
    for m,n in matches:
        if m.distance < 0.7*n.distance:
            good.append(m)
    if len(good)>10:
        src_pts = np.float32([ kp1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
        dst_pts = np.float32([ kp2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
        M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC,5.0)
        matchesMask = mask.ravel().tolist()
        print("Matched "+str(file))
        flag=1
    else:
        matchesMask = None

    draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                        singlePointColor = None,
                        matchesMask = matchesMask, # draw only inliers
                        flags = 2)

    img3 = cv.drawMatches(img1,kp1,img2,kp2,good,None,**draw_params)
    cv.imshow("Match",img3)

    cv.waitKey(0)
    cv.destroyAllWindows()

if flag==0:
    print("No Matches among the given set!!")

```

Image 50 Code

After marking bifurcation and termination of the input image, we will go to the database folder and get every png format templates, turn them into gray color by `cvtColor()` function. And then create another sift detector to contain the recent templates for finding key points and draw the keypoint in circle. Next step, we will get the input image and the template once again find the keypoints and descriptors. And then using `FlannBasematcher` to find the most similar and nearest keypoints between to images with 50 checks. After that a list will be created for containing the good matches which reach the required point of descriptor distance. If the matches reach out of ten, then we will keep the source points and the destination points to find the transformation of two images, all together put them in the match array which will be a mask image. Finally, raising the flag value to 1 mean that there is a match with this recent template in the database folder. To show it up on the screen as an image, we will use function `drawMatches()` which includes input image, templates, keypoints and the matching points with drawing parameters that defined above.

REFERENCES

- [1]: [OpenCV documentation index](#)
- [2]: <https://www.geeksforgeeks.org/>
- [3]: [Fingerprint Identification Using SIFT-Based Minutia Descriptors and Improved All Descriptor-Pair Matching - PMC \(nih.gov\)](#)
- [4]: [Flow chart of fingerprint matching using SIFT operator The feature... | Download Scientific Diagram \(researchgate.net\)](#)

APPENDIX