

Fingerprint Verification System

Yash Malviya

1 How to Run?

1.1 Install Dependencies

Use the following command to install all the dependencies

```
pip install -r requirements.txt
```

It installs all dependencies listed in the *requirements.txt* file. *opencv-python*, *scipy*, *scikit-image* are image processing libraries for *python*. It also installs *fingerprint-enhancer* which is 3rd party library to enhance fingerprints. I have implemented calculation of orientation map myself, but application of gabor filter is taken from the library.

1.2 Execute program

```
python fingerprint_verification.py fingerprint1_path fingerprint2_path
```

fingerprint1 and *fingerprint2* can be taken from *testcases* folder. Path will be entered like - *testcases/106_1.tif*

Use *-h* to print full help.

The output prints out on stdout. The number of minutiae found in each fingerprint, how many of them matched and what percentage of them matched are printed.

A window displaying minutiae on the skeleton of ridges for both images will open after another. To close the windows or open the image press *escape* key.

2 Implementation Details

2.1 fingerprint_verification.py

This is the main file, it parses command line arguments call other scripts present in *src* directory to facilitate fingerprint verification.

Output displays the follows -

Fingerprint 1 has X minutiae points
Fingerprint 2 has Y minutiae points
Z out of those match in between them
 $100 * Z / \min(X, Y)$ percentage of minutiae matched

Once above output is printed, It shows skeleton of ridges along with corresponding minutiae of the input fingerprints. First fingerprint 1 is shown on pressing *escape* key on the keyboard Fingerprint 2 is shown. Finally press *escape* key to close the window and exit the program

2.2 src/fingerprint.py

Contains class for fingerprint which does the following steps -



Figure 1: Original Image: the one inputted to the program straight from the sensor

2.2.1 Segmentation

Segmentation is done using the variance threshold method described in the slides.

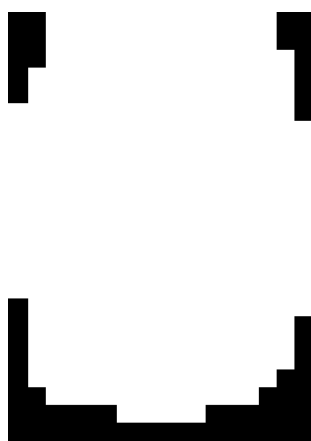


Figure 2: Segmentation Mask: White region is the area of interest for fingerprints further processing

2.2.2 Normalization

Normalization is done using the standard method, also described in slides



Figure 3: Normalized Image: Variance and mean of image adjusted

2.2.3 Orientation Map

Orientation map for each pixel is calculated using sobel operator. Then the same formula as in slides is used.



Figure 4: Orientation Map: Blue lines show direction of ridges

2.2.4 Frequency Map

Frequency map save value of inverse of wavelength in a particular segment. Wavelength is calculate using number of peaks in a segment.

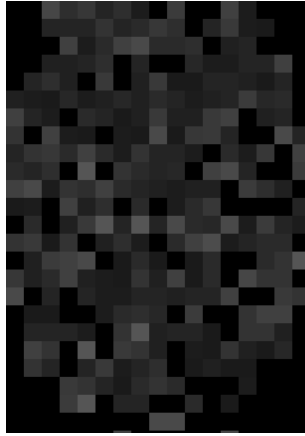


Figure 5: Frequency Map: wavelength of sinusoidal wave for Gabor filter

2.2.5 Enhancement

Frequency map comes out correctly, along with frequency map a gabor filter to enhance parallel lines in the direction of the local ridges can be obtained. But 3rd party implementation for this was taken due not getting proper results from self implementation.



Figure 6: Enhanced Image: broken ridges are joined and ridges are clearer

2.2.6 Thinning

Thinning is done using *skimage*'s *skeltonize* method which uses

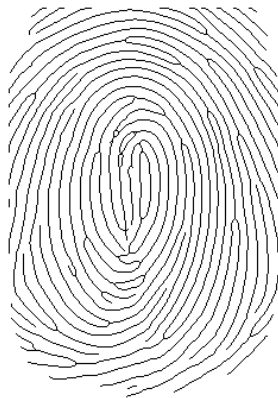


Figure 7: Thinned Image: Skeleton of the ridges from enhanced image

2.2.7 Minutiae Extraction

Cross Number (CN) algorithm is used to filter out pixels with CN values 1 and 3. 1 for ridge ending 3 for bifurcations.

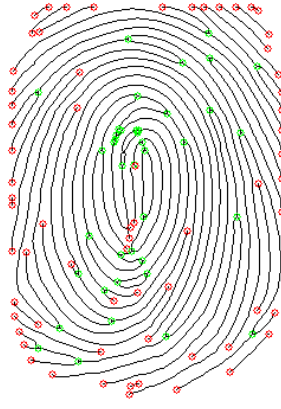


Figure 8: Minutiae Points: All minutiae according to cross number algorithm

2.2.8 False Minutiae Removal

Two type of false minutiae case were removed -

Up, Down, Left, Right neighbourhood is seen if the minutiae point lies in the certain distance threshold the minutiae point is removed.

Clusters containing points with distance less than certain threshold distance were removed.

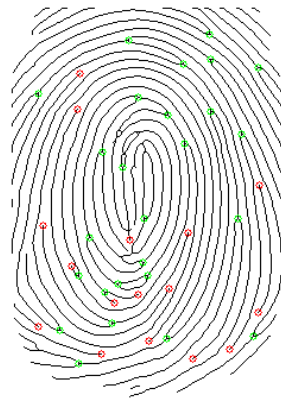


Figure 9: Accurate estimate of Minutiae: After removal of some false minutiae

2.2.9 Alignment

Alignment done using Generalized Hough Transform method, accumulator store discrete values for $\delta\theta$, δx , δy .

2.2.10 Pairing or Matching

During matching first query minutiae points were moved according to best parameters obtained from alignment stage. Then same method as in slides was used.

2.3 src/utils.py

Contains some general use function for stuff like round to closest multiple of 5, displaying image, euclidean distance etc

3 Results

Fingerprint impression from FVC 2000 DB.2B were taken. The *testcases* directory in the repository contains the following fingerprints -

- 104.1.tif
- 104.2.tif
- 106.1.tif
- 106.2.tif
- 110.1.tif
- 110.2.tif

The following minutiae images were extracted from them -

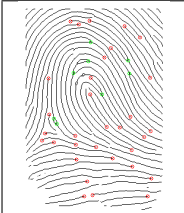
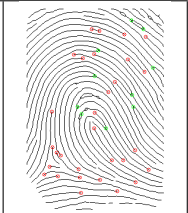
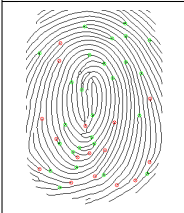
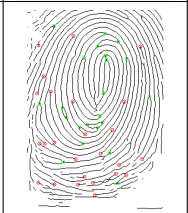
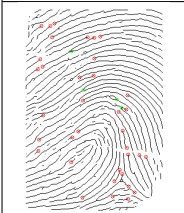
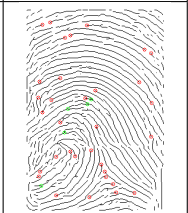
| | |
|--|--|
|  104.1 |  104.2 |
|  106.1 |  106.2 |
|  110.1 |  110.2 |

Table 1: Extracted Minutiae

3.1 Same fingerprint: Good percentage of matched minutiae cases

3.1.1 104.1 vs 104.2

Fingerprint 1 has 40 minutiae points
Fingerprint 2 has 40 minutiae points
30 out of those match in between them
75.000 percentage of minutiae matched

3.1.2 106.1 vs 106.2

Fingerprint 1 has 42 minutiae points
Fingerprint 2 has 47 minutiae points
28 out of those match in between them
66.667 percentage of minutiae matched

3.2 Same fingerprint: Minutiae with high noise

3.2.1 110.1 vs 110.2

Fingerprint 1 has 40 minutiae points
Fingerprint 2 has 38 minutiae points
4 out of those match in between them
10.526 percentage of minutiae matched

3.3 Different fingerprints

3.3.1 104.1 vs 106.1

Fingerprint 1 has 40 minutiae points
Fingerprint 2 has 42 minutiae points
2 out of those match in between them
5.000 percentage of minutiae matched

3.3.2 104.2 vs 106.2

Fingerprint 1 has 40 minutiae points
Fingerprint 2 has 47 minutiae points
0 out of those match in between them
0.000 percentage of minutiae matched