Fingerprint Recognition

Md Maniruzzaman, A. Sooryanarain

***Abstract*—This article is part of the course Pattern Recognition. The main goal of this project is to develop a system that can recognize fingerprints and check whether the given input fingerprint is the same as the reference fingerprint or not. OpenCV was used to code the fingerprint recognition algorithm**

***Index Terms*—Bio-metrics, Feature Matching, Fingerprint Extraction**

I. Introduction

Biometrics is a technology that can identify a person based on his physical characteristics. Fingerprint identification and recognition is one such method that finds great application in a variety of day-today appliances where there is a need for security and/or maintaining a log of multiple personnel. Due to its simplicity, precision and reliability, fingerprint scanner are used in mobile phones, tablets, laptops and other electronic devices.

The overall objective of this project is pretty simple does the given input image exists in the database or not. The input image might not always be looking exactly the same source image used in the database Fig. 1 .So even if there is a small offset in any of the parameters of the image, we will be unable to get proper results from our database search. In order to overcome this the features from the images are extracted and they are compared with the database.

In this project, the images that were used were obtained from a public dataset [1] . Later some pre-processing was done using few digital images processing techniques in order to improve their quality. Once the image were per-processed, key-points and their homing distances were calculated. This computed values are later used for comparing the input data with the ones in the database. .

Fig. 1. Fingerprint image in various positions.

II. Methodology

In order to check if the given input fingerprint is part of the database, the following steps were executed:

- Pre-processing
- Key-points extraction
- Feature Description
- Feature Matching

Note: The data set I used is called FVC2002 [1] and is published by the University of Bologna. There were 4 different data sets, for this project we used the dataset DB1.

A. Pre-processing;

1) CLAHE: Contrast Limited Adaptive Histogram Equalization, this technique is better compared to regular histogram equalization. The Adaptive Histogram Equalization is confined to regions and then it equalizes the histogram of each region. Where as regular histogram equalization takes the global contrast into account. The Contrast Limiting is used to reduce the noise [2]. The difference between CLAHE and global histogram equalization can be seen in Fig. 2. .

2) Ridge processing: First the region of interest ROI (ridges) is segmented out, from the regular image. Then the local orientation of each pixel in the image is calculated. Once the ridges are aligned vertically, inside its small segments, the frequency of the ridges are finally calculated.

3) Thresholding: Image is converted from grayscale to B/W binaries. This allow us to clear the image from unnecessary noise and helps us make the contrast between the very ”wrinkled” surface of the imprint and the other lines. Instead of using global thresholding, we use Otsu thresholding so that the algorithm itself will find the optimal threshold value for the given image [3].

4) Skeletalization: In order to improve the process of extracting key-points on the fingerprint, it is good to skeletonize the image itself Fig. 3. This creates more distinct thin lines of the ridges. Skelitization is based on the Lee algorithm [4]. In order to have a seamless set of lines, without any unwanted noisy dots; the skeletonized image is passed through simple double for-loop function was used to filter out the noise. .

B. Key-points extraction

The corners present in the skeletalized image is found using Harris Corner algorithm [5]. Once the (Harris) corners are found, the image is normalized to a 32 bit single channel float image. Out of all the (Harris) corners, the ones whose value is greater than 125 are accounted as key-points.

C. Feature Description

The features description are extrapolated from the key-points by using a ORB (Oriented FAST and Rotated BRIEF) algorithm [6]

Md Maniruzzaman is a MSc student at Institute of Technology, University of Tartu, Tartu 50411, Estonia e-mail: Maniruzzaman@ut.ee

D. Feature Matching

In order to match the extrapolated features from images, Brute Froce matcher algorithm was used [7]. The Hamming distance was used since we are getting the data in binary format for the BF-Matcher's input. The matched key-points were connected using the cv2.drawMatches() command.

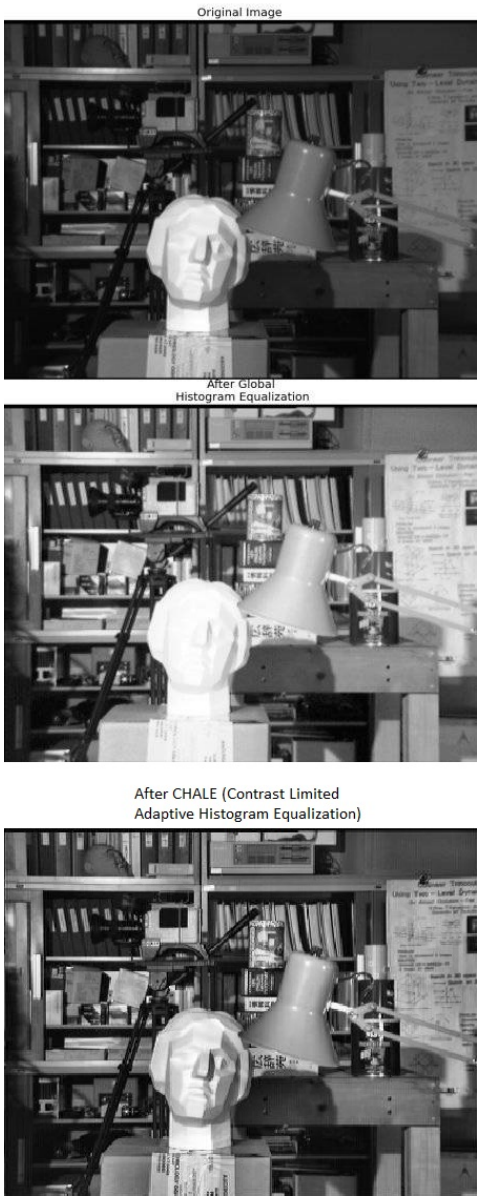

Fig. 2. Fingerprint image in various positions.

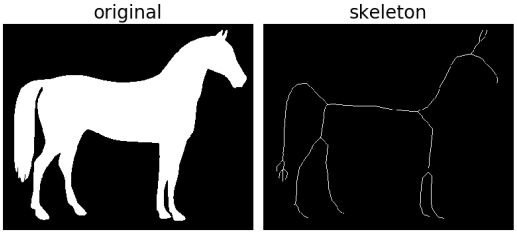

Fig. 3. Fingerprint image in various positions.

III. Results

We have tested the code with multiple fingerprints. For the visualization of the results, we drew the key-points over the original reference image and input image. The results of the skelitalized image along with the reference and input image are visible on Fig. 4. The top 15 possible key-points (Fig. 5) that were used by the feature descriptor were further connected lines with their respective matching pair, from the reference image and input image Fig. 6. The application seems to be reliable and accurate and it can detect the key features correctly.

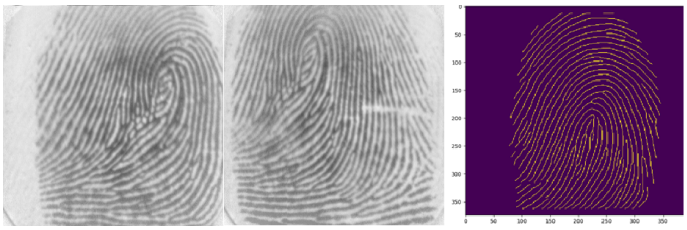

Fig. 4. Reference Image VS Input Image Vs Skeletalized.

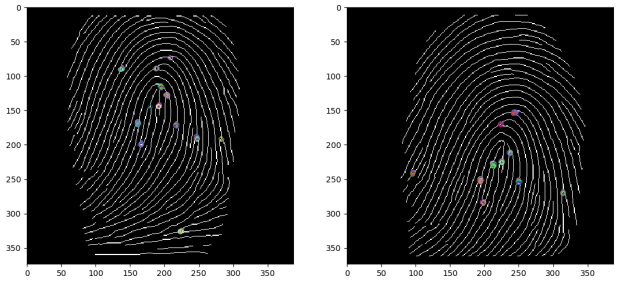

Fig. 5. Key-points Extrapolation.

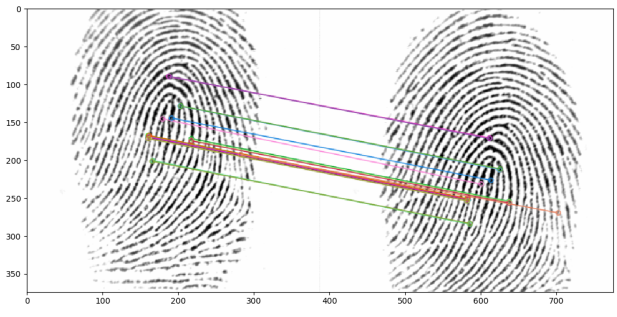

Fig. 6. Feature Matching.

IV. Discussion

One of the difficulties we came across was at times the code is unable to properly form the skeletalized image. Thus at times it fails to match the fingerprints. Also we were only comparing the input image with the reference image instead of searching for the input image in a given database. For future work, it would be nice to complete the code where the fingerprint matcher is able to match the given input fingerprint amongst the ones provided in the database.

V. Conclusion

This project has developed a fingerprint recognition system based on the method of finding key-points. Then these key-points are used to find the formal descriptors of the region around them and thus form a matrix that identifies the fingerprint itself. We tested the system on the data set FCV2002 DB1 to determine if it successfully recognizes the fingerprints.

Acknowledgment

The authors would like to acknowledge that the code was mainly adopted from GitHub. Therefore our Majority of the credit goes to GitHub user kjanko [8].

References

[1] "Fingerprint verification competition 2002." [Online]. Available: http://bias.csr.unibo.it/fvc2002/databases.asp
[2] "Opencv: Clahe (contrast limited adaptive histogram equalization)." [Online]. Available: https://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html
[3] "Opencv: Image thersholding." [Online]. Available: https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html
[4] "scikit-image: Skeletonize." [Online]. Available: https://scikit-image.org/docs/dev/auto_examples/edges/plot_skeleton.html
[5] "Opencv: Harris corner detection." [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html
[6] "Opencv: Orb (oriented fast and rotated brief)." [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html
[7] "Opencv: Bf matcher (brute-force matcher)." [Online]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html
[8] "Github: Kjanko." [Online]. Available: https://github.com/kjanko/python-fingerprint-recognition