Fingerprint minutiae extraction and matching for identification procedure

Philippe Parra
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0443
pparra@ucsd.edu

Abstract

Fingerprints are the most used biometrics in applications where a high level of security is required. This project implements the identification procedure: it matches one fingerprint among N fingerprints. It uses minutiae points based algorithms: in the enrollment step, the points are extracted from the print. Later on, during the authentication step, the points are matched. The first step is implemented using fingerprint enhancement and minutia filtering. The second step is realised using Ransac under an affine transformation model.

1. Introduction

The main objective of this research project was to create a performing and accurate program for fingerprint identification. I chose the subject of fingerprints because it involves different fields: pattern recognition, statistical tests, minutiae detection, performing algorithm programming. This project allowed me to discover computer vision field through a classic computer vision application. Fingerprint recognition stands for me as a stereotype in pattern recognition and detection. Techniques used in this field can be reused in other applications. Added to that, high security issues led to a lot of research on fingerprint recognition, meaning that the techniques used are advanced. Further introduction can be found on the website [1] and a video on [7]. Beyond the scope of this project, a complete identification application involves much more fields: we need to design a sensor, crypt the data, and implement a driver for the authentication device.

The two steps in fingerprint identification are presented on the figure 1. In this paper, I will present the results using the same fingerprint (shown on the figure 1).

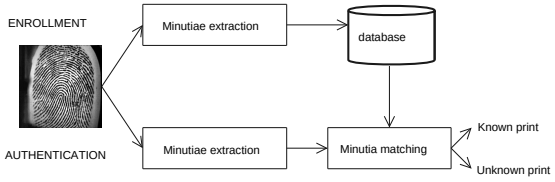

Figure 1. Identification procedure

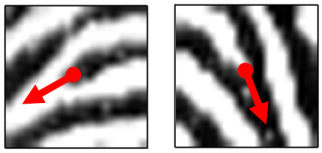

Figure 2. Ridge ending and bifurcation - Position and orientation convention

2. Minutiae points extraction

Minutiae points are extracted during the enrollment process and then for each authentication. In a fingerprint, they correspond to either a ridge ending or a bifurcation (figure 2). As we will see in 2.3, there is a duality between the two types of minutiae: if the pixel brightness is inverted, ridge endings become bifurcations and vice versa. The position of the minutia point is at the tip of the ridge or the valley. The orientation is given by the orientation of the arrow formed by the ridge or the valley according to figure 2. First, the local orientation field need to be computed. This will allow to enhance the print using oriented Gabor filter, and then better detect minutiae point using template matching procedure. This is summarized on the figure 3.

2.1. Orientation estimation

I estimate the orientation using the classic gradient technique (presented in [6]). Let's say we have an image I and we want to compute its gradient ∇I. The gradient in x and y (called I_x and I_y) are computed using Sobel filters (see

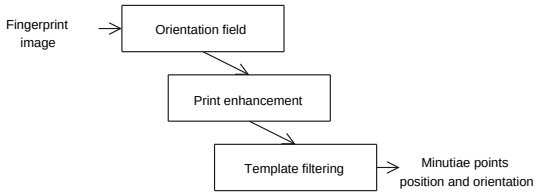

Figure 3. Minutiae points extraction steps

equation 1).

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (1)$$

Then the norm and orientation of the gradient can be estimated using the equations 2 and 3.

$$\|\nabla I\| = \sqrt{I_x^2 + I_y^2} \quad (2)$$

$$\arg(I) = \tan^{-1}(I_y / I_x) \quad (3)$$

In order to reduce the influence of noise, the orientation and norm of the gradient has to be averaged inside a small window (I used a 8×8 window). I tried different techniques to estimate the gradient inside that window.

The first technique is to simply average the gradient of each pixel inside the window. As we are only interested in the direction of the gradient and not the orientation, we cannot directly average the vectors. For example, the two unit vectors (1,0) and (-1,0) are pointing in opposite direction (0 and π) but have the same orientation 0. The direction is a number between 0 and 2π, whereas the orientation lies between 0 and π. The trick to average the gradient vector is to double the angle of their polar decomposition (trick presented in [9]). In the example, (-1,0) will become (1,0), and (1,0) will not be changed. Then, those doubled vectors are averaged to give the average gradient: its angle represents the orientation of the gradient and its norm represents the reliability of the angle. In the example, the average gradient will be (1,0): its orientation is 0 and its norm is 1, that is what we expect. In the fingerprint image, we are looking for the ridge orientation, we simply need to add $\pi/2$ to the angle to get the orthogonal orientation.

The second technique I used is the singular value decomposition of the covariance matrix of the gradient. The covariance matrix C represents how I_x and I_y are correlated and is defined by the equation 4.

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \quad (4)$$

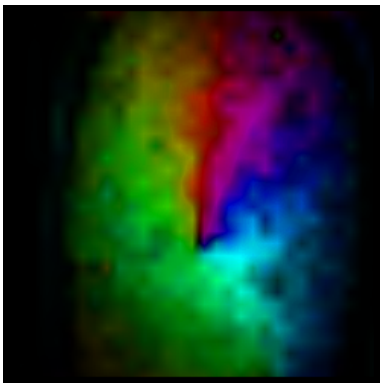

Figure 4. Bilinearly interpolated orientation in Hue color space

The SVD of C gives valuable information on the average gradient. It is given by the eigenvector having the highest eigenvalue. The smallest eigenvalue, which is positive because C is symmetric, represent the error we make by using the eigenvector as the average. The ratio highest over lowest eigenvalue gives the reliability of the orientation: if it is high, then the orientation is reliable, the data is mainly along a line representing the orientation. On the other hand, a ratio close to 1, means that the data is spread around, with no dominant orientation. The orientation is not reliable and it happens in uniform regions. The two eigenvectors are orthogonal: the ridge orientation is given by the eigenvector having the smallest eigenvalue. I keep this method in my program as it gives better results and good reliability information of the orientation.

After having the ridge orientation and strength on a 8×8 mesh, we need to estimate the ridge orientation for each pixel in the image. I bilinearly interpolate the 4 closest pixels where we previously computed the gradient and doubled the angles as presented previously in the first technique. The picture 4 shows the orientation and reliability information. Hue color space is used: the hue gives the orientation (0 ˚ for red,60 ˚ for green and 120 ˚ for blue), the saturation is full and the value gives the reliability (black not reliable, full color is reliable).

2.2. Fingerprint enhancement

The orientation information per pixel is only needed to enhance the fingerprint. This process is done to increase the quality of the print by reducing the noise, filling some small gaps and enhancing the ridges and valleys. The Gabor filters are often used in fingerprint identification algorithms (as presented in [9] and [6]). It is an oriented band pass filter, and a low pass filter in the orthogonal direction. When convolved with a ridge that has the same orientation, it smooths the values along the ridge. It also responds strongly if the

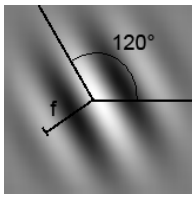

Figure 5. Upscaled oriented Gabor filter: f=0.2; θ=120 $^\circ$

Figure 6. Enhanced fingerprint using oriented Gabor filters

ridge frequency in the orthogonal orientation is the same as the band pass frequency.

I used a 17×17 kernel whose values are determined by the equations 5 and 6.

$$g(x, y, \theta, f) = \exp[-\frac{1}{2}(\frac{x_\theta^2}{\sigma_x^2} + \frac{y_\theta^2}{\sigma_y^2})]\cos(2\pi f x_\theta) \quad (5)$$

$$\begin{bmatrix} x_\theta \\ y_\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (6)$$

The standard deviation of the Gaussian in the two directions σ_x and σ_y were set to the constant value 4 (empirical values from [9]). The frequency of the filter f is a number usually between 0.125 and 0.2. The inverse of the frequency represents distance in pixels between 2 ridges (which is usually between 5 and 8). The figure 5 shows an upscaled version of an oriented Gabor filter.

Using the per pixel ridge angle, (5) and (6), I compute the corresponding oriented Gabor filter and apply it. The figure 6 shows the enhanced print.

2.3. Template filtering

It is more accurate to find the minutiae points on the enhanced print. Two approaches can be found in the literature to deal with this problem. The first is to binarize the enhanced print and run a line thinning algorithm on it. This is pretty straightforward, but requires a lot of false minutia cleaning (further information can be found in [9]). I choose the second method: the goal is to find a filter that, when applied on the enhanced print, pops out the bifurcation and ridge endings.

2.3.1 Filters used

After trying some real minutia point template in various directions, I decided to create a template using maths functions. The main motivation is because it allows to orient the template in any direction without problems in filling the holes of the rotation. It is also convenient because it can be tuned easily. I used interpolated cosine wave functions with different frequencies. In order to rotate the filter in the local orientation of the ridge, I used the equation 6.

2.3.2 Local maxima extraction

Due to the duality of ridge ending and bifurcation, the enhanced image has to be filtered using two filters. This can be done by inverting the image, or the filter, or rotating the filter of 180 $^\circ$. I chose the first solution even if it is not the faster one. Then, the local maxima of the two filtered images need to be extracted and merged together (a threshold can be adjusted here). The local maximum is extracted using a small window of 21×21: if the center of this window is the maximum of all the values inside that window, then it is a local maximum. The figure 7 summarizes this process.

3. Minutiae points matching

Points matching is a common task in computer vision. In the project I choose an affine transformation model between the points and solve it using Ransac (presented in [3] and [4]).

3.1. Problem formulation

The problem formulation is directly inspired from [9]. Let T and I be the template and input fingerprint we are trying to match. Previously, we stored the position and orientation of the minutiae point. I and T may not have the same number of feature points: let m and n be the number of points for T and I.
$T = \{m_1, m_2, ...m_m\}$, with $m_i = \{x_i, y_i, \theta_i\}$ $i = 1..m$
$I = \{m'_1, m'_2, ...m'_n\}$, with $m'_j = \{x'_j, y'_j, \theta'_j\}$ $j = 1..n$

Two minutiae points m_i and m'_j are considered to match, if their position and orientation are close. This can be written according to (7) and (8), using the spatial distance sd and direction difference dd.

$$sd(m_i, m'_j) = \sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2)} < R_0 \quad (7)$$

$$dd(m_i, m'_j) = \min(|\theta'_j - \theta_i|, 360 - |\theta'_j - \theta_i|) < \theta_0 \quad (8)$$

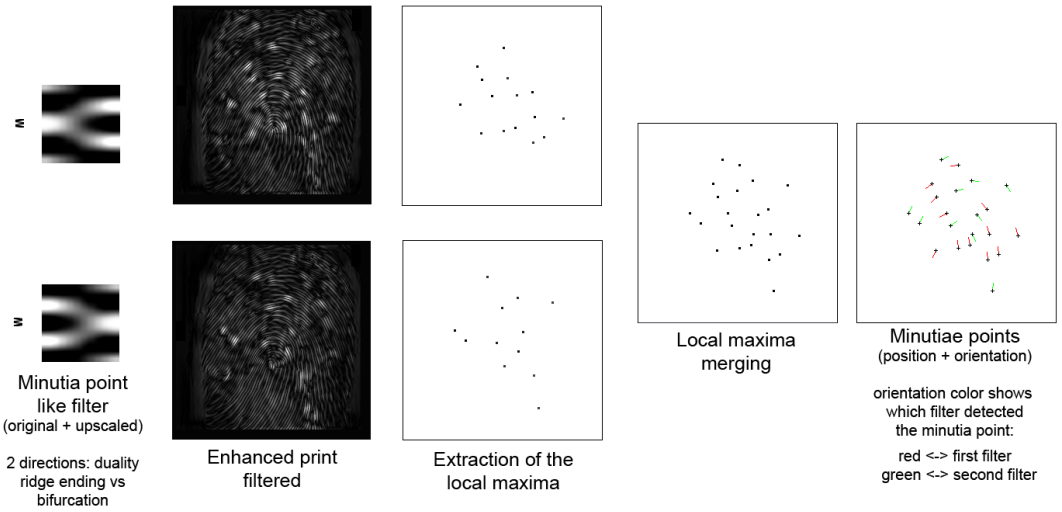

Figure 7. Template filtering algorithm

R_0 and θ_0 are the two tolerance parameters we can adjust. Those inequalities come from the classic euclidean distance in 2D and the circularity of angles.

Let md be the function that says if two minutiae points are close or not i.e. $md(m_i, m'_j) = 1$ if $(sd(m_i, m'_j) < R_0$ and $dd(m_i, m'_j) < \theta_0), 0$ otherwise.

Now, we try to match the points i.e. finding a transformation: $y = map(x)$, for all points x in the 3D space, it returns an other point y. We need a pairing function P: $P(i) = j$ means that the mate of the m_i in T is the minutia m'_j in I.

Finally the matching problem can be written as finding map and P that maximize the equation 9.

$$\sum_{i=1}^{m} md(map(m'_{P(i)}), m_i) \tag{9}$$

3.2. Finding correspondences

As a preprocessing step of Ransac, correspondences need to be found between the input and template points. In other words, we should find for each point in the input image, which points in the template are likely to be the matching point. This is a version of the classic problem of feature correspondence. It is usually solved using Normalized Cross Correlation or SIFT (from [8]) on a small window around the points. The issue here is that a minutia point looks like an other minutia point, thus the different feature vectors will be close and the distance between them not accurate. I present a technique which compute for each minutia point a feature vector, which will be compared using a feature distance. The smallest the distance, the more likely will be the points to match.

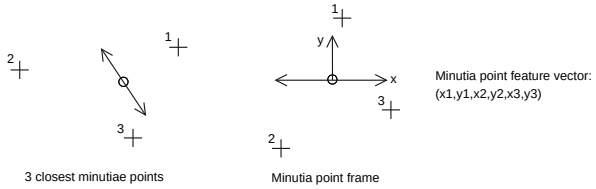

Figure 8. Feature vector computation

3.2.1 Feature vector

To solve this problem I need a non classic algorithm, somehow copying what humans do when trying to find correspondences. Let's consider a minutia point m_i in the template set and the 3 closest minutiae points from m_i in the template: we number them $m_{i,1}$, $m_{i,2}$ and $m_{i,3}$. We do the same for the points in the input set: m'_j, $m'_{j,1}$, $m'_{j,2}$ and $m'_{j,3}$. We cannot compare directly the euclidean distance between the points because of rotation of the print.
The first step is to rotate those points of an angle equal to the local orientation of m_i: it is the same as considering the points in m_i frame. The feature vector is given by the 3 positions of the 3 points ($m_{i,1}$, $m_{i,2}$ and $m_{i,3}$) in the minutia point frame. This way, the feature vector is independent of the rotation in the image. We can add the orientation in each point in the feature vector, but I did not try it. The figure 8 shows this process.

3.2.2 Feature distance

Since we have a rotation invariant feature vector in the template and input set, we can then compare them: I used the euclidean distance. But due to the numbering of the points, which may not be the same for the two feature vectors (and

	$m'_{j,1}$	$m'_{j,2}$	$m'_{j,3}$
$m_{i,1}$	5.3	3.1	10.2
$m_{i,2}$	1.7	6.4	8.2
$m_{i,3}$	7.9	2.1	0.2

Table 1. Feature vector distance matrix D: $D_{k,l} = \|m_{i,k} - m'_{j,l}\|$

also one or more false minutiae), we need to compute the distance between every permutation of the points. The feature vector has 6 dimensions $(x_1, y_1, x_2, y_2, x_3, y_3)$, when compared to $(x'_1, y'_1, x'_2, y'_2, x'_3, y'_3)$, we compute 9 euclidean distances between (x_i, y_i) and $(x'_{\sigma(i)}, y'_{\sigma(i)})$ as shown in the table 1.

Then, we need to combine those numbers: if we want that all the three points match perfectly, then the distance will be given by equation 10.

$$d_3(m_i, m'_j) = \min_{\sigma}(\sum_{k}(D_{k,\sigma(k)})) \tag{10}$$

An other extreme approach, is to say that only 1 point can match perfectly, the others can be wrong: then the distance is given by the equation 11.

$$d_1(m_i, m'_j) = \min_{k,l}(D_{k,l}) \tag{11}$$

I chose the intermediate solution, that allows one point to be wrong, and require 2 points to match: it is more robust to noise than (10), and more accurate than (11). The equation 12 gives the distance. In the example given table 1, it will be equal to 1.9.

$$d_2(m_i, m'_j) = \min_{k_1 \neq k_2, l_1 \neq l_2}(D_{k_1,l_1} + D_{k_2,l_2}) \tag{12}$$

Finally, for each minutia point m_i in the template, I choose the 5 points m'_j in the input set having the smallest d_2 distance, and assign them as being the correspondences. Ransac is then fed with this information, that will greatly help to find (or not) a solution.

This kind of approach, called geometric hashing, has been addressed in [5]. In order to have a feature vector that is independent of the permutation of the points, they are ordered according to the edge length. The edge length is, for the fingerprints, the number of ridges between the minutiae points. Furthermore, the orientation of the minutia point is added to the feature vector. This approach is interesting, because it removes the ambiguity of the feature vector, but [5] do not give more information, on how to compare two vectors: it might be the euclidean distance, or the more accurate Mahalanobis distance. It did not have time to compare my method and this one because I did not try to count the ridges (that would also allow to estimate the local ridge frequency).

3.3. Ransac

3.3.1 Affine transformation estimation

First, we need to define the kind of transformation we allow between the two sets of minutia points. I chose the simplest model, which is the affine transform. Due to skin elasticity, there can also be non linear transformation that will not be taken into account. An affine transformation in 2D is defined by 6 degrees of freedom as shown in equation 13.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = AX + B \tag{13}$$

A is responsible of the rotation, scaling and shearing, B is the translation between the centroids of the two sets. It exists only one affine transformation that map 3 points on 3 others points (if we are not in a degenerated case). So, if we take 3 points in the input set and template set, we can find the unique transformation that maps the first set on the other. I compute the determinant of A (which is the product of the eigenvalues), to see if the transformation spread the points. If it is close to zero, it means that A is degenerate. If it is close to 1 it means that the points are spread. As the prints are often about the same size (there is maybe some little scaling between them), I used a threshold of 0.7 to discriminate between the two cases. When the transformation is degenerate, I do not transform all the points to count the number of matches because it may be wrong. Computing the determinant has the advantage to be very fast compared to computing the ratio of the eigenvalues (which is more accurate).

A and B can be found using homogeneous coordinates and inverting the equation given in (14).

$$\begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \tag{14}$$

3.3.2 Ransac algorithm

Because of its combinatorial complexity, it is impossible to try every possible affine transform. To solve this problem I used Ransac (Random Sample Consensus). Its philosophy is to try at random some transformations and expect to find a good one, i.e. being close from the maximum defined by (9). First, I define some parameters of the algorithm:

n**:** minimum number of random points required according to the model chosen, $n = 3$ for an affine transformation

k**:** number of iterations required, I used $k = 5000$

d**:** number of matching points required to assert that the transformation fits well, I used $d = 10$

R_0**,** θ_0**:** thresholds used to identify that a point fits well, I used $R_0^2 = 30$ and $\theta_0 = 0.3 \approx 20^\circ$

The algorithm is based on this process:

- Take $n = 3$ random points both in the template and input set (only among the corresponding points of the points chosen in the template)
- Find the affine transformation mapping the template random points on the input points
- Count the number of points which match (using md function) after applying the transformation to the all the template points. That will give a score (using the distance between the points) of the match

This process is repeated k times, and stop if we count more than d matching points (then we can refine the solution using the matching points, solving for the least square solution). The result of two matching prints is shown on figure 9.

3.3.3 Error estimation

The algorithm may not find a solution, even if a solution exists. It is obvious, that when there is no solution, the algorithm will not find one. We can estimate the probability p_{err} to fail, when there is a solution (the solution presented is inspired from [3]). When a point in the template set is chosen, then we choose a point among the corresponding points. Let's define the probability p as being the probability of choosing the good corresponding point (it exists and is the good one). This probability depends a lot on the accuracy of the correspondences.

The probability to fail k times $p_{err}(k) = p_{err}(1)^k$. The probability to fail 1 time is 1 minus the probability to succeed, which means that we choose the 3 good corresponding points. Thus $p_{err}(1) = 1 - p^3$ and $p_{err}(k) = (1 - p^3)^k$. If we want to decrease p_{err}, we can increase k, or decrease p.

In my program, I use $k = 5000$ iterations. If $p = 1/20$ (quite bad correspondence algorithm) then $p_{err} \approx 0.53$. If $p = 1/5$ (better correspondence algorithm) then $p_{err} \approx 3.6 \times 10^{-18}$. This probability does not directly depends on the number of points, but p may decrease if the number of points increases. This shows that finding accurate correspondences is very important to make Ransac algorithm reliable.

4. Testing protocol and results

4.1. Training and testing sets

4.1.1 Fingerprint database

I downloaded some fingerprint samples from the Fingerprint Verification Contest [2] (years 2000, 2002 and 2004). I got a total of about 1200 prints. Each finger was acquired 8 times in different conditions, thus I had about 150 different fingers in the database. I kept 600 prints for the testing set, and the remaining were divided evenly into two training sets. I did not have the time to use the Synthetic Fingerprint Generator SFinGe from [2]. This allows, knowing the ground truth (minutiae point position and orientation), to improve the algorithm (mainly the minutia point detector).

4.1.2 Testing set

As the program was quite slow, I decided to run it on a subset of the entire testing database. I divided the testing set into 3 sets according to the quality (high, medium, low). The high quality set contained 21 different fingers, the medium and low quality contained 25 fingers each. I then randomly chose 4 fingers in each different quality set. Finally, the testing set contained 96 different fingerprints coming from 12 different fingers (presented in the figure 10).

I did $96 \times 95 = 9120$ comparisons to test my program. Among them, $8 \times 7 \times 12 = 672$ comparisons were between the same finger and $9210 - 672 = 8448$ comparisons were between different fingers (respectively 7.4% and 92.6% of the overall comparisons).

4.2. Overall results

I ran my program during about 20 hours and the results given in the table 2. Prints indexed from 1 to 4 are the high quality print, 5 to 8 the medium ones and 9 to 12 the bad quality ones. The score of 75 for the Finger 1 vs Finger 1 mean that 75% of the 56 comparisons declared the prints to match.

The results from the table 2 are averaged and given in the table 3.

4.3. Results explanations

The overall results are not so good, especially with low quality prints. The main reason is because the minutia point detector does not give accurate results. Added to that, low quality print contains a lot of noise and scratches that lead to false minutia points.

I had only one false accept in the 8448 comparisons. This number will increase with the size of the database. I

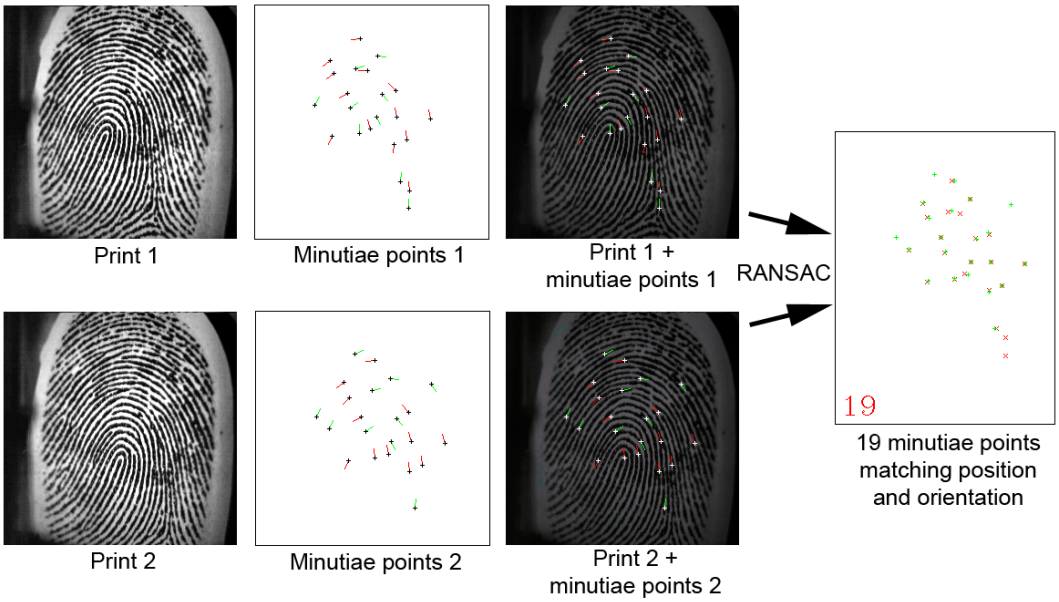

Figure 9. Ransac matching algorithm: the prints are considered to come from the same person

Quality	High				Medium				Low			
Finger	1	2	3	4	5	6	7	8	9	10	11	12
1	**75**	0	0	0	0	0	0	0	0	0	0	**2**
2	0	**57**	0	0	0	0	0	0	0	0	0	0
3	0	0	**11**	0	0	0	0	0	0	0	0	0
4	0	0	0	**4**	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	**50**	0	0	0	0	0	0
7	0	0	0	0	0	0	**7**	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	**7**	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	**5**

Table 2. Matching percentage

Quality	High	Medium	Low
High	**36.6**	0	**0.1**
Medium		**14.3**	0
Low			**3.1**

Table 3. Average matching percentage

also limit to 25 the number of detected minutiae points. This prevent partially that an impostor print generate too much minutiae points that will increase its chances to have a match.

Finally, my algorithm worked well on on the kind of prints I used for training and prints with a high contrast. I did not tuned it on a lot of prints, that is why the overall results are not so great. It is promising for a future work, since most of the time an algorithm is designed for a particular sensor.

5. Future work

This project was an introduction to fingerprint identification based on minutiae point matching. I can be improved in every approach I presented. The main improvements that can be done are given in the following list:

- Automatically compute the local ridge frequency, that is used in Gabor filters and minutiae templates
- Improve the minutiae templates, try different shapes (thin or thick minutia point). This part seems to be the most difficult: it requires a lot of training and tunning.
- Use an other algorithm for correspondence based on human strategy when doing this, add more information in the feature vector (orientation for example)
- Use fingerprint classification to speed up the algorithm
- Allow non linear transformation in the matching process

6. Acknowledgements

I would like to thank Serge Belongie for his helpful enlightments and experience. Also I enjoyed the CSE190a class environment: the projects were engaging and covered different areas of computer vision and learning. The comments and questions during the class were very valuable for all of us.

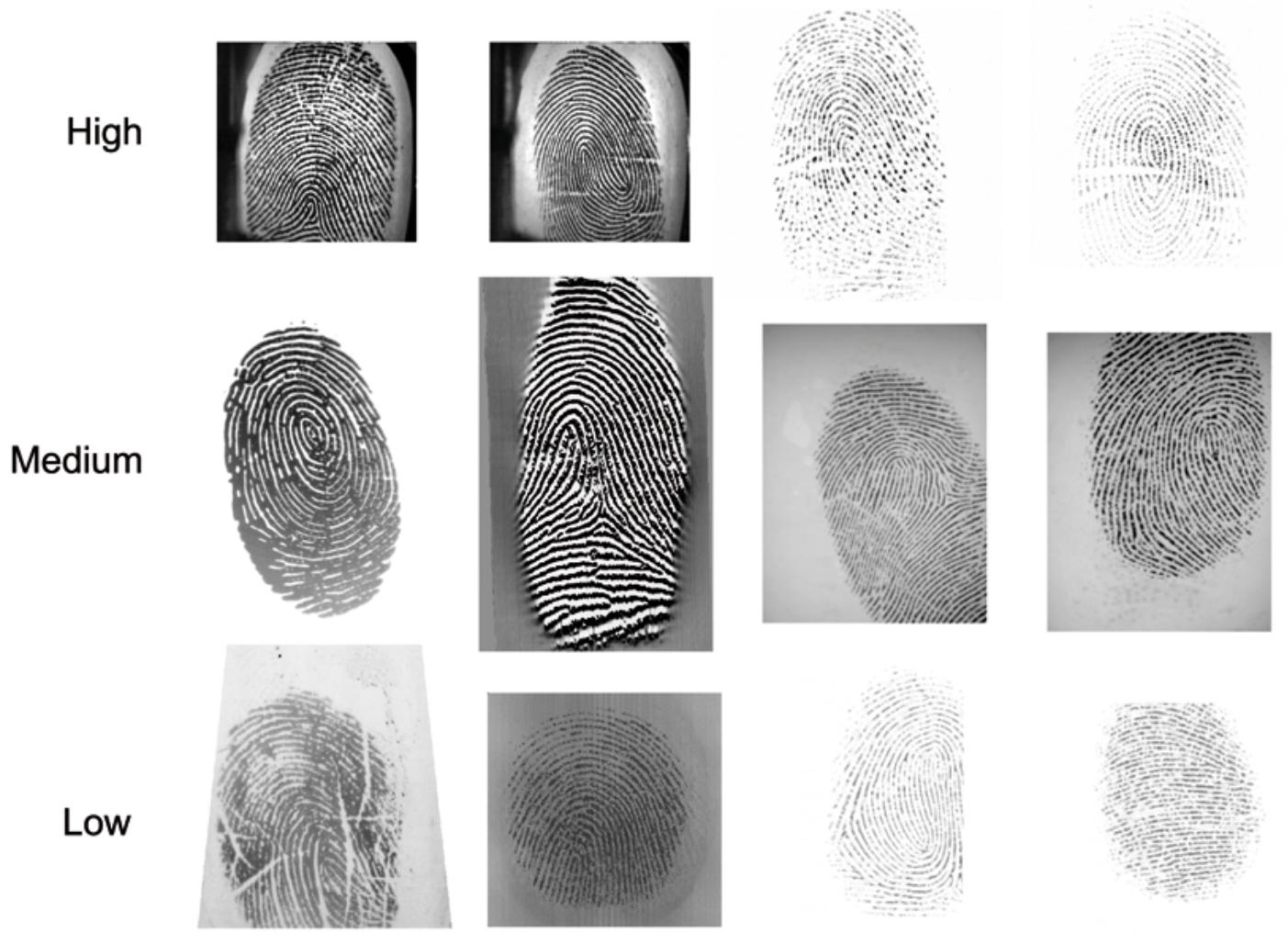

Figure 10. Testing set, one print per finger is shown

References

[1] Biometric research, Michigan State University `http://biometrics.cse.msu.edu/`,. 1

[2] Biometric System Laboratory, University of Bologna `http://biolab.csr.unibo.it/`,. 6

[3] R. B. Fisher. The ransac algorithm. `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FISHER/RANSAC/`,. 3, 6

[4] D. A. Forsyth and J. Ponce. *Computer Vision a modern approach*. Prentice-Hall, Upper Saddle River, NJ, 2003. 3

[5] R. S. Germain, A. Califano, and S. Colville. Fingerprint matching using transformation parameter clustering. IEEE COMPUTATIONAL SCIENCE and ENGINEERING. 5

[6] R. Gonzalez and R. Woods. *Digital image processing (2nd edition)*. Prentice-Hall, Englewood Cliffs, NJ, 2002. 1, 2

[7] A. K. Jain. Mitacs annual general meeting. `http://www.pims.math.ca/industrial/2002/mitacs-agm/jain/`,. 1

[8] D. Lowe. `http://www.cs.ubc.ca/~lowe/`,. 4

[9] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003. 2, 3