Biometrics (A6M33BIO) laboratory exercise
Task 1: Person identification using fingerprint

Jakub Schneider, contact email: schnejak@fel.cvut.cz

October 1, 2020

1 Introduction

Identification using a fingerprint image can be divided into two main parts: preprocessing and matching. Preprocessing involves segmentation of the fingerprint from its background, determining local ridge orientation, determining the local ridge frequency, filtration using Gabor filters for quality enhancement, fingerprint skeleton formation and minutiae search. Every part of the process can be done using number of methods. We will discuss only selected state of the art methods. Very exhaustive overview can be found in the book [**?**].

In the fingerprint matching two methonds will be compared. First, **minutiae matching**, which is based on classical dactylography. It compares minutiae topology on the fingerprint — only around 10 minutiae are needed to decidedly identify a person. Second method, using **Finger Code**, is based on classification of the features extraceted from fingerprint texture.

During the task you will get acquainted with preprocessing algorithms practically on several selected fingerprints. Some parts of preprocessing methods are planned to be implemented by you. In the second part — matching — you will have to implement both matching methods and than to test their potential in fingerprint identification. A toolbox for Matlab is provided within the lab carrying functions for data loading and processing. Functions used in all procedures are listed in the text and in the toolbox help.

2 Fingerprint Preprocessing

2.1 Local Ridge Orientation

First technique used in fingerprint processing is computation of local ridge orientation in a point $[x, y]$. The orientation is characterized by an angle Θ_{xy} and is computed at discrete positions rather than in every image pixel. The image is processed block by block which determines the spatial step of Θ_{xy} discretization. Computation is based on gradient methods — gradient $\nabla(x, y)$ of the input image is a two dimensional vector $[\nabla_x(x, y), \nabla_y(x, y)]$. This vector is directed to the direction of maximal intensity change. Θ_{xy} is then

perpendicular to the gradient. The final Θ_{xy} is obtained by averaging all directions in the block. Example of processed fingerprint is on Fig. 2.1.

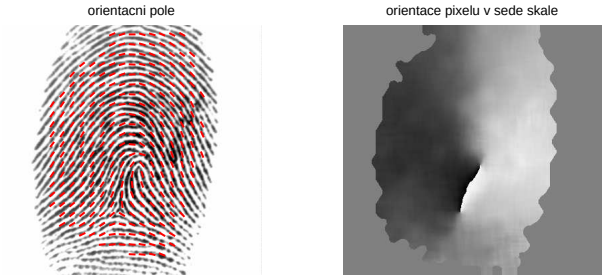

Figure 1: Local ridge orientation, function `computeorientationarray(im, imSegmented, 10)`

2.2 Local Ridge Frequency

Ridge frequency (or density) is a number of ridges per length unit in a direction orthogonal to ridge itself. As in the case of ridge orientation, ridge density is traditionally determined in discrete positions. Algorithm for local ridge frequency computation implemented in Fingerprint Toolbox is based on oriented window usage — the window (traditionally wider than longer) is oriented orthogonally to the ridge direction by its longer side. Columns of the widow are averaged providing one dimensional vector according to a ridge profile. From the profile we are able to assess number of spaces between peaks and than determine ridge frequency as $f = \frac{number\ of\ spaces}{\sum(total\ length\ of\ spaces)}$. Example of ridge frequency image is on Figure 2.2.

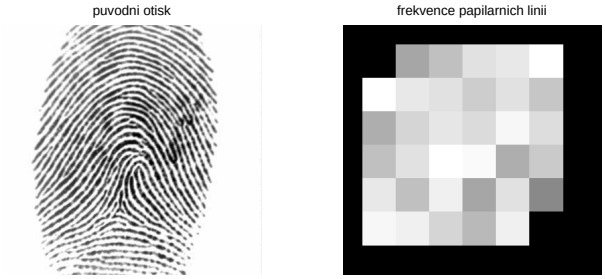

Figure 2: Ridge frequency, function `computelocalfrequency(im, imSegmented, orientationArray)`

2.3 Fingerprint Segmentation

Aim of the fingerprint segmentation is to process only relevant part of the fingerprint image. Number of algorithms have been developed for segmentation so far. Great review can be found in [**?**].

Segmentation is, same as previous algorithms, based on block processing. Image is divided into blocks and every of them is then classified as fingerprint or background. Alternatively, more classification groups can be defined, e.g. smudgy, poor. The easiest technique

for fingerprint segmentation would be thresholding based on pixel intensity. However, the intensity approach is not such powerful because the actual characteristic of the fingerprint is a ridge line structure.

Methods used for segmentation can be divided into following groups:

- Using **ridge orientation** when histogram of ridge orientations is assessed from every image block. If a significant peak appears in the histogram it means that the block has a ridge-like structure and is classified as fingerprint.
- Using **image pixel intensity variance** in **a direction orthogonal to ridges direction**. This approach is based on the assumption that intensity variance is independent on direction outside fingerprint.
- Using **pixel intensity gradient** averaged in every block. This approach assumes that gradient (i.e. contrast) is significantly higher in areas with ridges than outside fingerprint.
- Using **Gabor filters**. Each block is filtered using a set of 8 Gabor filters with different rotation. The output of this filter is used for both classification into fingerprint or background, and may be used to assess the quality of the block. This step may be associated with subsequent smoothing using fingerprint G. filtration.

Figure 3: Sample segmentation. Technique using gradient of pixel intensity was used.

2.4 Enhancement using Gabor filters

Filtering using Gabor filters (so called contextual filtration) is a directional filtering commonly used as an edge detector. We use is for smoothing discontinuities caused by poor image quality. The Fig. 2.4 shows a filter mask in the spatial domain. Its shape gives the directional properties.

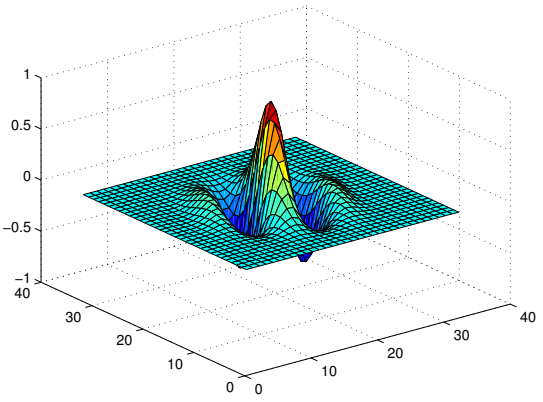

Figure 4: Mask used for Gabor filtration, which is convolved with filtered image. Isometric view.

Implementation of the Gabor mask will be a task. Its definition is as follows:

$x =< -16, 16 >, y =< -16, 16 >$
$x_p = sin(angle) \cdot x + cos(angle) \cdot y$
$y_p = sin(angle) \cdot y - cos(angle) \cdot x$
$gab(x, y) = exp\{-\frac{1}{2} \cdot [(\frac{x_p^2}{t_x^2}) + (\frac{y_p^2}{t_y^2})]\} \cdot cos(2\pi f \cdot x_p)$

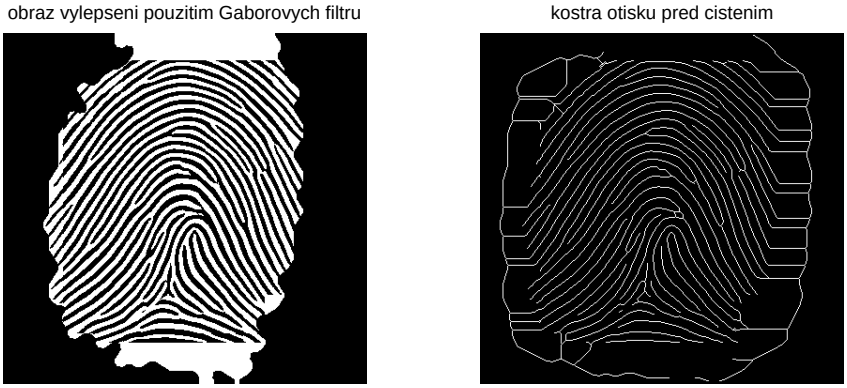

Figure 5: Gabor filtration, `enhance2ridgevalley(im, imSegmented, orientationArray, frequencyArray, 0)`

2.5 Fingerprint skeleton, minutiae

After Gabor filtration, ridge lines are reduced to a minimum width producing the **skeleton image**. Minutiae are then searched using simple rules.

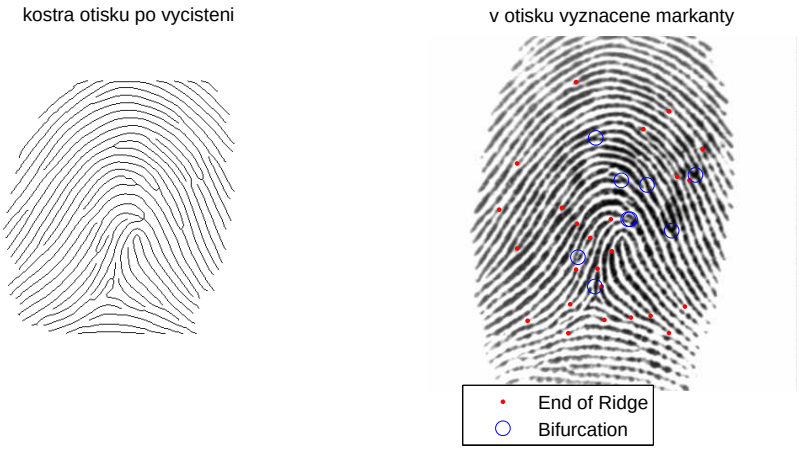

Figure 6: Final minutiae detection.

3 Person Identification

Identification of a person identity is well known method used in criminology. First, enough large database of fingerprints is needed. Then we get the finger impression, and task is to find matching fingerprint in the database. When no fingerprint in the database is suitable, we should declare unknown result.

Most important thing in the identification process is a method comparing (matching) the input fingerprint with those in the database. In previous chapters we have seen the fingerprint being preprocessed and prepared for extraction of possible features. Now we will introduce two methods used for fingerprints matching. The first is based on the classical fingerprint matching using minutiae. Minutiae topology determines uniquely the human fingerprint. The second method is based on the fingerprint texture.

3.1 Minutiae Matching

Minutiae matching implementation will by one of the tasks in this lab. Function `match.m` is prepared in the Fingerprint Toolbox for implementation.

Algorithm: two sets of minutiae points are on algorithm's input: `mAi1` and `mAi2` and threshold `d` determining maximal distance.

• For every point in `mAi2` find the nearest point in `mAi1`. When minutiae are nearer than `d` than mark them as a pair and remove them from both `mAi1`, `mAi2`. For every paired minutiae add `nbmatch = nbmatch + 1`

• After pairing of all minutiae from `mAi2` compute $matchingScore = \frac{2 \cdot nbmatch}{|mAi1|+|mAi2|}$ which ranks the fingerprint pair (score values are from interval $< 0, 1 >$).

Syntax of `match` function is as follows:

```
[matchingScore, nbmatch, inputmatch, dbmatch] = match(mAi1, mAi2);
```

On input we have matrices with marked minutiae for database fingerprint ($mAi1$) and

matched fingerprint (*mAi2*). Those matrices are output of function `findminutia` and *mAi2* must be aligned with *mAi1*.
Outputs:
matchingScore — score of fingerprint similarity.
nbmatch — number of matched pairs.
inputmatch — matrix with the same size as mAi2 containing only marks of minutiae which have been paired.
dbmatch - same as inputmatch but for mAi1.

3.2 Matching using Finger Code

Fingerprint image is filtered using a set of Gabor filters with different rotations. Each filtered image is processed block by block without overlap. Value for each block N is computed: $f(N) = |mean(N) - std(N)|$ where std denotes the standard deviation, mean denotes average value of the block. We obtain new image where every block is replaced with one pixel (Block image). In the Block image only annulus of defined radius is extracted and its values are used as the feature vector, as shown in Figure 7.

Figure 7: Finger Code matching.

Subsequent feature vectors f_1 (corresponding to the database fingerprint) and f_2 (corresponding to the input image) are created by putting the image values into the feature vector in defined order. Fingerprint matching is than computed as follows
$matchingScoreGabor = -mean(abs(f_1 - F_2))[1]$. A negative value is used as a measure to make it the biggest for the best matching fingerprint.

There is a function for Finger Code matching `fingercode_creation.m`.

`fingercode = fingercode_creation(imOriginal, Gfilt, core, maskSize, dia)`,

imOriginal is an input image, *Gfilt* is a set of k Gabor filters created using function `GaborFilter_creation`, *core* are coordinates of fingerprint core, *maskSize* is a size of the image block and *dia* is two-element vector containing diameters of the annulus.

Outputs:
fingercode - vector sized [1 x N], where N depends on annulus size.

4 Tasks

Each student will choose 5+ fingerprints of different quality (from individual subjects) and examines the preprocessing methods. The main task will be to create a system which finds the most likely matching fingerprint, to a given fingerprint, in a provided database or decides on the inability to find a suitable adept.

Tasks:

- First on provided fingerprint calculate frequency and orientation fields. Visually assess how has fingerprint quality affect the result of the calculation. [2 pts]
- Implement your own fingerprint segmentation algorithm. You can choose from the algorithms mentioned in Section 2.3 or choose another from the literature ([**?**]). You can come up with your own algorithm. [4 pts]
- To enable Gabor filtering you have to implement Gabor mask making able execution of the function `enhance2ridgevalley.m` (you implement sub-functions `filtergabor`) according to Chapter 2.4.
- Record your own set of fingerprints on our sensors and compare how preprocessing functions deal with outputs of different sensors. Are found minutiae always the same or different? Does your algorithm correctly segments the fingerprint? [2 pts]
- Implement function `match.m` for fingerprint minutiae matching. [4 pts]
- Implement function `fingercode_creation` for fingerprint matching using Finger Code. [4 pts]
- For a given fingerprint find the best matching identity (ID) using the provided database 'DataBase'. Examine the difference in obtained score/similarity using different sensor types. (The database includes fingerprints obtained from different sensors. You don't have to use all of them) [2 pts]

4.1 Bonusy:

- Provide optimisation for the automatic matching function. [up to 5 pts]

A Appendix: Fingerprint toolbox structure.

The toolbox is divided into 2 parts: *predzpracovani (preprocessing)* and *porovnani (matching)*, every part contain functions for particular parts of the task. Necessary functions are described in the text.