Biometrics laboratory exercise
Exercise 3: Iris recognition

Eduard Bakštein, eduard.bakstein@fel.cvut.cz

4. ledna 2013

Introduction

Iris recognition is a highly accurate biometric method with a wide range of applications, including airport automatic check-in, access systems or humanitarian aid missions and more. High robustness and lifelong stability of the iris pattern are among the main advantages of the system, as well as the possibility to read the pattern from a distance. Basic elements of an iris recognition system are image capture, segmentation (finding the iris region within the image), unwrapping to pseudo-polar coordinates and encoding into iris template.

The aim of this exercise is to try implement several parts of the recognition system and try them out on real data. The exercise is divided into following parts:

1. Implementation of iris region segmentation using circular Hough transform.
2. Match provided iris images to subjects in the database using iris code.
3. Bonus task: capture and processing of own iris images.

The solution will consist of implementation of requested functions within provided iris toolbox, available at the lab website. Additional to the source codes, you will write a brief report, including image outputs of individual processing steps and your comments. You should be able to easily reproduce what you did from the report even few years after! Outputs of some of the tasks are explicitly requested in this text.

You are expected to work individually. The lab uses source codes by Libor Masek from University of Western Australia.

Exercise 3, Iris - feature extraction and iris code matching (20 points)

Hand-in instructions

The task is to be handed-in personally to the TA at the last lab of the iris block. You will briefly comment your implementation (incl. source codes) and present function on provided images. For easier presentation, you can use our report. You should also be able to answer the questions, given throughout these instructions. It is necessary to also submit the source codes (just compressed ToDo folder) and report to the upload system by the end of the day when presentations take place.

1 Iris segmentation using circular Hough transform

Prior to encoding of iris pattern to iris code, it is necessary to segment (separate) iris region from background and noise (sclera, eyelids, eyelashes etc). The easiest way is to assume the iris region to be annular region, consisting of two non-concentric circles of different diameter, representing inner - pupillary - and outer iris boundary. To find this region, we can then use circular Hough transform (HT) (See second Iris lecture). In this method, edges in the image are first identified using edge detector. Then, each edge point is projected to parametric space: The parametric space is initialized with zeros. Then, all points in the parametric space at given distance (= known radius) from current edge point are incremented by a constant. Given the original image contains circle of the selected radius, a maximum value of the whole parametric space will emerge in its center. The method is easily extended to a circle of unknown radius: the projection is performed for a range of assumed radii separately and a maximum is searched over all the parametric spaces.

Parametric space

Let us assuma an image of size x times y pixels. Searching a circle of unknown radius in the range $r \in R, R = \{r_{min}, r_{min}+1, ..., r_{max}\}$ centered at arbitrary point in the image, the initial parametric space will be a zero matrix of dimensions $(y + 2 \cdot r_{max}) \times (x + 2 \cdot r_{max}) \times |R|$, where $|R|$ is cardinality of set R, here, the number of attempted radii [1].

The individual layers of parametric space are then filled with circles of radius corresponding to respective layer, centered iteratively at all edge points. To find circle coordinates, you can either start from the analytical equation for circle:

$$(x - a)^2 + (y - b)^2 = r^2, \tag{1}$$

from which you can easily derive one of the coordinates by solving the quadratic function, and iterating values of the remaining variable. The easiest way is to use function *circlecoords*, provided in the iris toolbox, which returns point coordinates, corresponding to requested circle parameters, directly (Here, function *sub2ind* may come handy [2]. Moreover,

[1] If you use provided function *circlecoords* for calculation of circle coorrdinates, which automatically limits points to given range, the extension of parametric space in the $x-y$ axes is unnecessary and the parametric space will be of dimensions $y \times x \times |R|$.

[2] If you try to index using two integer vectors to index a matrix in MATLAB, a carthesian product, rather than individual individual [row-column] pairs will be used! The *sub2ind*() function converts such pairs to linear indices, that can then be used

beware of the convention [$rowcolumn$] vs. [xy])). Another possibility is to use goniometric functions.

Brief description of Iris toolbox is provided as appendix at the end of this document. The skeleton functions, you are expected to fill your code in, are in the ToDo folder.

Task:

1. Implement the function

   ```
   [circlePupil, circleIris] = findIrisAnnulus(uint8 iris_image),
   ```

   based on Hough transform. *circlePupil* and *circleIris* are parameters of both circles you found in the form [x, y, radius] for pupil and iris. You should set the ranges for HT parameters such that no adjustments are necessary for different provided images.

2. Use your implemented function with the rest of Iris toolbox to encode provided iris images

3. Put the images from individual processing steps (edge detection, identified circles and parametric space) in the report.

Once you have implemeted the function *FindIrisAnnulus*, you can run the toolbox on arbitrary image - just exchange path to the sample image by path to your image in the *iris_demo.m* function. Iris function (the *createIrisTemplate* function in particular) will perform unwrapping of the annular region you datected to pseudo-polar coordinates and template encoding using Gabor wavelet phase quantization (see lecture slides for more details).

Implementation notes:

- For **edge detection** you can use the function $edge(I, 'canny', thres)$. The function first calculates gradient field of the image and then applies threshold (given by $thres$ parameter). Experiment with the threshold value to achieve satisfactory edge detection and excessive edge point removal.

- The high-frequency components in the image, such as eyelashes or other noise, lead to a high number of undesired edge points in the image. These produce noise in the parametric space and increase processing time substantially. To remove these points from the edge image, you can use pre-process the image prior to edge detection using vairous brightness/contrast adjustments or by low-pass filter. To apply *gaussian filtering* (or "smoothing"), you first have to create filter mask using $fspecial$, and then apply it to the image.

as a vector. Ex.: $a = magic(3); r = 1 : 3; c = r; a(r, c) = ?; a(sub2ind(size(a), r, c)) = ?$

```
G = fspecial('gaussian',[a a],sigma);
Ig = imfilter(image,G);
```

Variable a determines size of the mask produced, *sigma* is standard deviation of the 2D gaussian distribution.

- Try out, whether your code works better for inner or outer iris boundary and perform this detection first. The parameters you identify in the first step can then be used to restrict search space in the second step (e.g. center of the outer iris boundary should lie within the previously identified pupil etc.). It can be also useful to use different preprocessing settings for the detection of inner and outer boundary.
- Each reasonable restriction of the range of search parameters reduces processing time considerably!

2 Identification of provided iris images in the database

This task simulates the process of iris-based identification in a way, common in real-life application. First, iris scan of an unknown person is taken (here: provided iris images). This image is then encoded in a way compatible to the rest of the database. Then, a subject with highest similarity is searched in the database, which provides most probable match in the database. If this similarity falls below matching threshold, a match is declared.

To compare two iris codes, a measure called *normalized Hamming distance* (HD) is commonly used. The normalization constant is given by the number of bits in overlapping usable regions in both codes.

$$HD = \frac{\| (codeA \otimes codeB) \cap maskA \cap maskB \|}{\| maskA \cap maskB \|}, \qquad (2)$$

where $\otimes$ is logical *XOR* operator, $\cap$ logical *AND* and $\|$ *norm* operator[3] - the number of true-bits in a binary vector. CodeA and codeB are the iris codes to be compared, while maskA and maskB are corresponding noise mask (here, 1 means useful signal, 0 is noise. *Beware: in Iris toolboxu, the convention for mask is reversed! Use not/tilde operator to negate the mask.*)

To avoid undesired effects of possible iris rotation (produced by head tilt) on comparison output, a bit shift od the iris codes is used. One of the iris codes (both the template and mask) are shifted in selected range and HD is calculated for each shift. Then, minimum value is selected.

[3]Beware: *norm* operator in this formula operates on binary/logical values, contrary to the MATLAB function *norm()*, which provides euclidean distance from origin and is therefore inappropriate in this context! You can easily use the function *sum()* to calculate number of ones in a binary vector.

Task:

1. Implement the function

   ```
   HD = irisHammingDistance(codeA,codeB, maskA, maskB)
   ```

   for comparison of two iris codes, based on the formula (2). The function should implement also the bit shift in the range ± 12 bits[4]. Here, the $circshift()$ function may come handy.

2. Walk through the database of stored iris codes (*database.mat*) and identify the person with best match for each of the images provided. For each of the images, put id of best matching person (and eye - L or R) together with minimum HD in the report. You can use the provided script *walkDatabase.m* for easier database iteration.

3. Iterate through the database one more time and perform all-against-all comparison. Store the resulting HD values to two arrays, based on whether the same (iris codes from the same eye of the same person) or different eyes were compared. Plot both sets in a common histogram. What are the HD distributions for same and different eyes? Where would you put the recognition threshold (select one)? What would be the false accept (FAR) and false reject (FRR) rate for your threshold? You can use ROC curve for appropriate threshold setting. What does the resulting histogram say about our method. Is it accurate? Is it robust? Put the histogram together with your comment in the report.

3 Bonus tasks (up to 5 extra points)

In these bonus tasks, you can make use of all your implemeted functions. The tasks are not too time consuming but will provide you with deeper understanding of some involved problems, as well as the possibility to process your own biometric data.

Bonus task: processing your own iris images

- Use the iris camera VISTA FA2 to capture iris images of both your eyes.

- Adapt parameters of the iris toolbox to suit images from the camera used. It will most probably be necessary to adjust *irisConfig.loNoiseThreshold* in *iris_demo.m*, which defines threshold for eyelashes (regions darker than given threshold are marked as noise in the noise mask). Current threshold value is set for the default database.

[4]Realize how many bits are produced in one step of phase quantization (see lectures for more details). Adjust the shift step accordingly and avoid substantial number of meaningless comparisons!

- Convert the images to iris codes and store them in your own database. What is the hamming distance between images of your left and right eye? What is the relation between this value and the threshold you determined in the previous task? Put the results in the report together with your comment.

Bonus task: Attacking the iris system using printed eye images

We will use eye photographs, captured using conventional camera, printed using laser printer and classical photographic technique.

- Capture the provided eye image prints using the iris camera.
- Segment the images, convert to iris codes and compare them to the provided iris codes, calculated from direct iris images. What is the hamming distance? Would such attack be successful, considering your decision threshold?
- Attempt to detect such attack using 2D Fourier transform: calculate 2D power spectra of direct image and both printed images. What are the differences between the spectra? Is there some difference between the laser-printout and the photograph? Which method do you suggest to detect such attack and how robust do you think it would be? Put images and your comments in the report.

Congratulations!

You have just finished implementation of a remarkable part of iris recognition system. You have tried out, what preprocessing and segmentation means and explored various properties of iris code comparison using normalized Hamming distance. You have shown histograms for comparison of same and distinct eyes and assessed qualities of your system from statistical point of view. Your system works in a very similar way as commercial systems all around the world, used in security or airport systems. Moreover, we hope that when you come to think ”it was great, but how did I actually do that?”few years later, you will have your protocol still handy.

Acknowledgement

Preparation of this task was supported by the Ministry of Education Youth and Sports by the grant FRVŠ 2529/2012.

Appendix: Brief overview of the Iris Toolboxu

- You can get basic idea of the iris toolbox by running *iris_demo.m*. The demo will show values, precalculated in the cache.
- The main function, responsible for feature extraction, is *createiris-template.m*
- The function *iris_init.m* sets toolbox parameters and adds paths to MATLAB PATH. Among others, you can set here the noise threshold when processing your own iris images.
- Empty skeleton functions, you are supposed to implement, can be found in the *ToDo* directory. This is the place to store all codes and functions you will produce during working on the assignment.
- You can find the provided iris images in the directory *Images*.

Data structure

Iris templates and masks, produced by encoding iris images, can be found in *database.mat*, containing array of the same name. There are 30 fields for 30 persons, stored in the database. For each person, there are structure arrays *R* and *L*, containing various number of iris codes. Each iris code is stored in field called *template*, corresponding noise mask is in the field *mask*.

Example: The iris code, corresponding to the third image of the left eye of person no. 12 can be accessed through: `database(12).R(3).template`, corresponding noise mask then through `database(12).R(3).mask`. However, iterating through the database is implemented for you in the script*walkDatabase.m*.