# Lab 5 & 6 Iris Recognition

## 1. Classical iris recognition

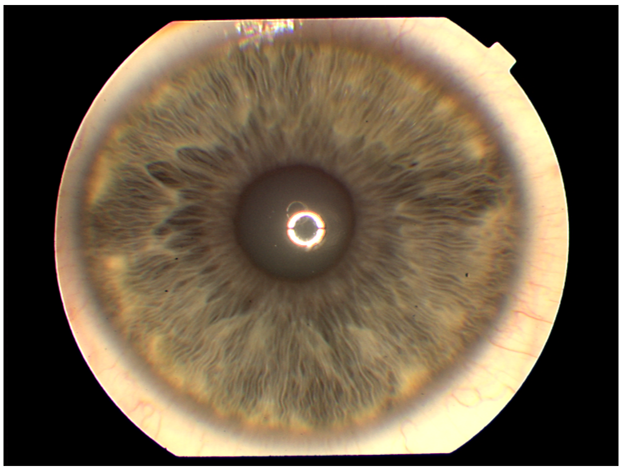
### A. <u>Introduction</u>

In this exercise, the approach proposed by John Daugman - one of the most widely known “classical” solutions - will be presented. It is worth noting that the presented approach is (was) patented.

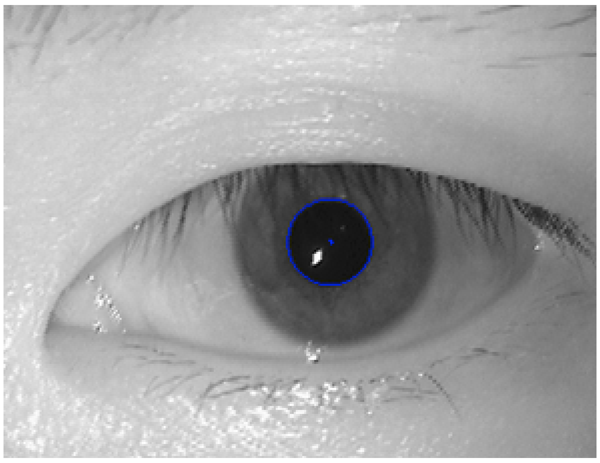
Daugman, J. (2004). "How Iris Recognition Works". IEEE Transactions on Circuits and Systems for Video Technology. 14: 21–30. doi:10.1109/TCSVT.2003.818350

### B. <u>Overall concept</u>

We will work with images like this (to be blunt - their quality is very good and not obtainable with typical iris image acquisition devices):



A typical iris biometric image look like this (here with pupil segmentation):



The issue can be divided into several stages:

1) Iris segmentation - it is necessary to find and isolate the iris on the image of the eye - in practice, this means the need to determine the circle (ellipse) in which the pupil is inscribed and the circle (ellipse) in which the iris is inscribed.
2) Feature extraction - filtering the image with Gabor filters - extracting important information about the anatomical features of the iris (texture)
3) Creation of an iris "mask" and ultimately a so-called iris code (in case of this method a binary one)
4) Creation of a mechanism to compare two iris codes and determine a similarity measure.

## C. Iris segmentation

- Get the train dataset: https://drive.google.com/file/d/1vuYI2haT3AI7Xs-hWDksYdGwXoVUZpWv/
- Read the image *irisA_1.png.* Convert it to grayscale. Display it. Why grayscale ? (1) Daugman's algorithm was designed for grayscale images, (2) acquisition of iris images often takes place using active near-infrared illumination and so the registration of colour images is very difficult. (3) we are interested more in texture than in colour.
- At this point we will take a moment to think about what we want to achieve and with what methods (stay tuned).
    - Goal: iris segmentation - in the simplest case we need the parameters of two circles (one describing the pupil and the other describing the iris). It is worth noting that the two circles do not have to have a common centre and, additionally, the circle describing the iris can be partially obscured by the eyelids (not in this database, but often - look in the mirror). Note, in fact using circles is a simplification and we should describe both the pupil and iris as **ellipses**.
    - Method: Daugman proposed the method of "exploding circles". Mathematically it is described by the formula:

$$max_{(r,x_0,y_0)} = \left\| \frac{\delta}{\delta r} \Phi_{(r,x_0,y_0)} I(x,y)\, ds \right\| \quad (1)$$

      where: I - image, $\phi$ mask of a circle with parameters $(r,x_0,y_0)$
    - The idea is as follows. We analyse circles with a given centre and an expanding radius. We sum the pixels lying on the circle (for accuracy we assume that the circle has a "thickness" of 1 pixel). As there is a big contrast between the pupil and the iris (in general, the pupil is dark and the iris is brighter), we should detect a big change in brightness at the pupil/iris border (the calculation of this change in formula (1) is illustrated by the derivative). Additionally, this change should be greater the better the circle describes the pupil.
    - The idea of searching the iris/retina border is similar, but due to the potential obscuration of the upper and lower parts of the iris by the eyelids and the presence of glare, full circles are not analysed, only arcs.

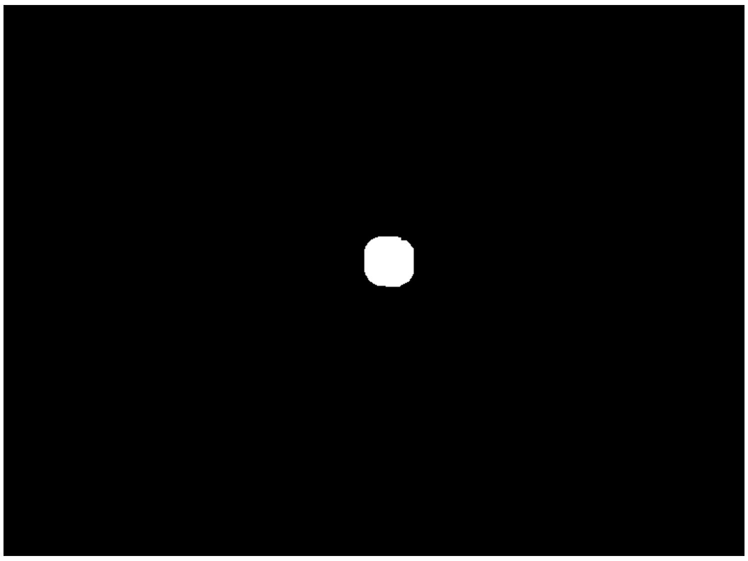
The following steps **are already done** (but you can read the description to get better knowledge about it):

- Image preprocessing. The image we are analysing is generally of very good quality (clear pattern on the iris, good sharpness, nothing is obscured), the only problem is the glare occurring in the centre of the pupil (from the lighting). It does not allow the exploding circle method to work properly - therefore it must be eliminated. Note: this stage of image processing is strongly dependent on the method/device with which the test images were collected. Additionally, in this exercise we will not try to fully automate the method - the goal is to present the problem in general, not to create an application that works in most (all) conditions.

- The glare is the brightest element in the image. Plot a histogram of the image (`H = cv2.calcHist([I],[0],None, [histSize], histRange)` - note the syntax) and then, based on the analysis of the histogram, set the binarization threshold to segment the glare. Binarization - `cv2.threshold`. Display the result of the binarization. If necessary, adjust the binarization threshold so that the glare is quite clear. You will still see a thin border around the eye, but don't worry about it.

- The binarization output can (and should) be improved using morphological operations - e.g. opening, closing, erosion, dilaton:
  ```
  morphologyEx(img, cv2.MORPH_OPEN, kernel)
  morphologyEx(img, cv2.MORPH_CLOSE, kernel)
  erode
  dilate
  ```
  with a structural element, e.g. 3x3 (`np.ones((3,3),dtype='uint8')`) square and with hole filling (this is a little bit tricky in Python - https://learnopencv.com/filling-holes-in-an-image-using-opencv-python-c/).
  Basically we want to get a clear mask of the glare and its centre (hence hole filling).
  The proposed (but not the only right) sequence:
  - opening with a 3x3 structural element - allows to remove unwanted pixels from the edge of the image (there is such a bright border there)
  - dilation with a 7x7 structural element - allows to "recreate" the reflection area
  - hole filling
  - dilation with 7x7 structural element - improves the effect of hole filling

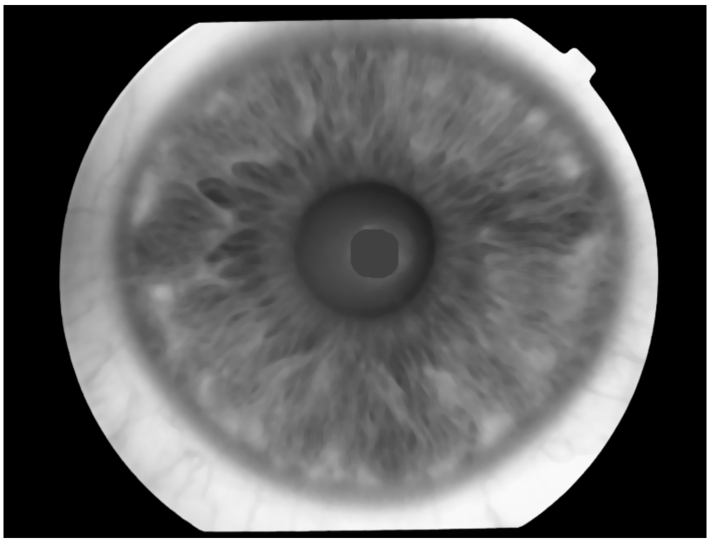
  Note: if the glare fails to fill, the binarization threshold should be adjusted.



- Next we will have to find the centroid of the glare - as our initial estimation of the pupil centre. Use `connectedComponentsWithStats`. You should end up with two objects - background and our glare. Draw the obtained centroid (`cv2.circle`).

- Now we should darken the reflection area on the original image (in grayscale). This is realised e.g. with the syntax (choose which one you want):
    - Version 1
      ```
      I = (I*(1-mask/255.0) + 65*(mask/255.0))
      I = I.astype('uint8')
      ```
    - Version 2
      ```
      I = np.where(mask,64,I)
      ```

  (we leave the pixels for which the mask == 0, and assign the remaining ones the value 65 - not 0, because the difference would be too big).
    - I - is the image in grayscale
    - mask - is a binary image with reflection
    - the fill value (65) should be chosen so that the pupil area is as uniform as possible (for this image 65 is an acceptable value)

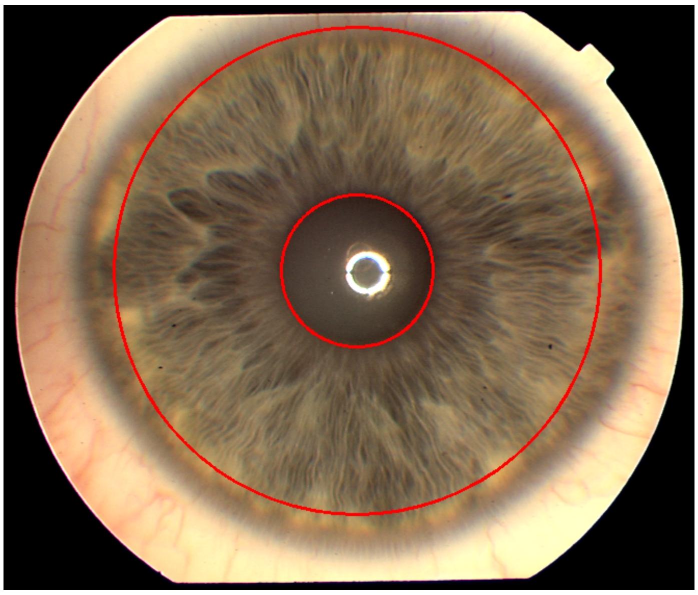


- In addition, we perform median filtering of the image (with darkened glare) (cv2.medianBlur(I,5)) with a 5x5 mask.
- Check the results of glare removal and filtering.
- If the code works correctly, please convert it into a **function**.

From now on, you should **write your own code** and extend the given script:

- Now we can proceed with the detection of the circle describing the pupil using the exploding circle algorithm.
- Concept:
    - Using the initial estimation of the pupil centre, we search on a 3x3 grid (seed, starting points) with predefined distance between points.
    - In each iteration we start with an initial radius, compute the **mean brightness** on the circle and store this value. Then we increase this value (step) and again compute the mean brightness.
    - At the end we analyse the obtained results and find the radius, for which the brightens change is the biggest.
    - Then we find the biggest brightness change for all seed/starting points. This allows us to get the center and radius.
    - Finally, we find the highest value for all seeds - this is our estimated **pupil centre and radius**.
- Implementation:
    - General remark: although this algorithm seems complicated, it is not. You only need to understand what you are trying to accomplish and also use visualization to check if your code is correct.
    - The whole code works in two loops over the seed points. We assume that we used 9 seed points (3x3 grid) with a configurable step. As the center we use the estimate from glare detection.
    - Inside the double loop we have another two loops: (1) radius and (2) angle.
    - Radius: we select an initial radius (eg. 50), step (eg. 5), an upper limit (200). The initial value should be configurable (when we convert code to a function).
    - Angle: for the pupil we use the full 360 with a step eg. 5.
    - Inside we should obtain the radius pixel locations (using radius, angle, sin, cos, and the seed point). We sum the number of points and their brightness.
    - We store the mean brightness for each radius.
    - Hint. Here a good idea is to draw the points - just to be sure that they are OK and form “exploding circles”. Use: `cv2.circle.`
    - After gathering values for a particular seed , we need to compute the absolute differences between consecutive values and find the biggest one (with the corresponding index).
    - Finally, we should find the biggest step (the absolute differences between consecutive values described above) for all seeds - the location and radius are our result.
    - Hint. Display the circle to check if the method works as expected.
    - Convert the code to a function.
    - To further improve the pupil centre we should run the function a second time - with the initial location from the previous set, decreased radius (by about 20) and smaller step between seed points (fine grain search).

- The next step is to detect the circle describing the iris using the exploding half-circle algorithm. We do not use circles due to possible occlusions.
- The approach is very similar to exploding circles. We have to assign an additional parameter (left or right part) and modify the angle range. We should use the pupil radius (slightly increased) as the start radius. Again - visualize how the method works.
- Finally you should have parameters of the pupil centre and radius, as well as  the iris center and radius (seperate left and right). Visualise them:



## D. Feature extraction - Gabor filters

- The second stage of biometric iris identification is feature extraction from iris images. The essential differentiating element of irises is the arrangement of connective tissue patterns clearly visible in the analysed images.
- Basic idea: We use Gabor filters to extract iris features. The idea is as follows (proposed by Daugman): we cut a fragment (of size larger than the filter) from the iris image and perform a convolution operation with the Gabor filter. From the obtained coefficients (composite) we calculate the sum. As a result, we obtain one complex number. We analyze the sign of the real and imaginary part. If the part is positive then we encode this information in binary as '1', otherwise we write '0' in binary. Example : 2 -3i corresponds to the code '10'.
- Daugman proposed a code of 2048 bits. If we assume that we use filters with 8 orientations (0º, 22.5º, 45º, 67.5º, 90º, 112.5º, 135º, 157.5º), and each of them allows us to obtain 2 bits of information it follows that we need: 2048/ (2*8) = 128 filter locations.
- Of course, Gabor filters with different parameters can be used (smaller, larger, sinusoids with different frequencies, etc.) The solution proposed here is one possibility (rather simpler).
- The remaining problem is how to select 128 locations in the image. To do this, we use the previously determined: coordinates of the pupil centre and the pupil and iris radii (left/right). The idea is to create a coordinate system that is independent of the current stretch of the iris (it is assumed that the pattern on the iris behaves like an

image on a rubber band - it stretches and contracts respectively). The coordinate system is polar. The radius is given as:

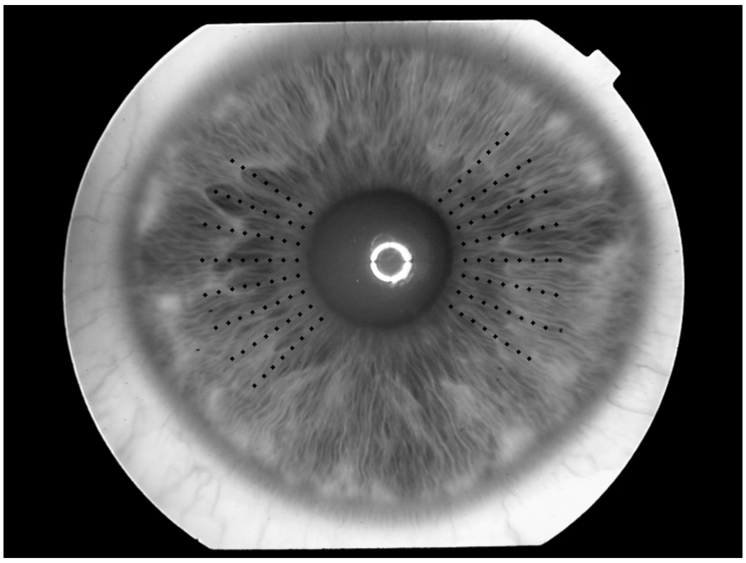
(7)

where: $r_z$ - pupil radius, $r_t$ - iris radius, - alpha parameter from the range [0;1].

- The range of variation of the angle is assumed from -45° to 45° counting from the horizontal line passing through the centre of the pupil (due to potential interference in the upper and lower parts of the iris). The interval of variation of the radius is divided into 8 parts (so in total there will be 16 because of the left and right side), the interval of variation of the angle is also divided into 8 parts. In total, this gives us 128 locations.
- Note: this is only one possible implementation (rather the simplest).
- Implementation:
    - again we work in two loop radius and angle,
    - radius: we set the step and range (here check if you do not have more than 8 items
    - angle: we use two loops - left and right.
    - in each of them we obtain the location coordinates (similar like in the exploding circles) and append them to a list
    - Gabor: we prepare a bank of filters:

```
from skimage.filters import gabor_kernel
kernels = []
sigma = 2
for theta in range(8):
   t = theta / 8. * np.pi
   kernel = (gabor_kernel(0.15, theta=t, sigma_x=sigma, sigma_y=sigma))
   kernels.append(kernel)
```

    - Loop over the sample points:
        - get a sample location (from the one obtained above) - the points, for a given image are presented in the Figure below.
        - get a patch around the sample location (eg 21x21),
        - for each filter in the bank:
            - perform filtration (separately with real and imag part) and encode the result (like described). Save the code.
    - Finally, you should end-up with a 2048 binary code.
    - It would be nice to have the whole processing integrated in one function - input image output code. However, you should also display the segmentation result, as in some case the “glare-based” approach could fail



Points on the iris where we “measure” the pattern i.e. apply the Gabor filter and extract the information.

**E. Comparing two iris codes**

- At this point we have a mechanism to convert the iris image into a 2048 bit vector (iris code). For a complete (but simplified) system we still need a mechanism to compare the two iris codes. Daugman proposed a very simple way using the XOR operation and so called Hamming distance.
- It is defined as:

$$H = \frac{\sum XOR(A, B)}{2048}$$

- Basically we get the percentage of differing bits.
- Implement a function that will allow to compare two codes.

**F. Biometric identification system**

Now we have all the modules of need to implement an iris-based biometric identification system. At this point we will try to create to integrate them
Assumptions:

- we have 5 people in the system, for each person we have 2 iris images in the database and one test image
- we want to load all images and generate an iris code for each of them
- we want to check if, based on such data, it is possible to create a biometric identification system, i.e. if by presenting any image of an eye we will obtain correct identification in the case of a person from the database or information that a given image presents an iris from outside the database.

Implementation:

- First we should compute the iris codes for all images in the train database (using the created code).

- Next compute ale Hamming distances and analyse the results - does the system work correctly ?
- Test of the biometric system (image in the test set):
    - get the test dataset: https://drive.google.com/file/d/1sitO3V6OLRS4otdfTYpCjPlq4i8lP5I4/view?usp=sharing
    - load image 'irisA_3.png',
    - determine the iris code for it
    - compare the code with any from the train database - find an item from the database with the minimum Hamming distance with this code
      - check if this distance is smaller than the selected threshold
      - you may be tempted to display a feedback of the type: "Identification correct - person 1" or "Person not present in database".
- Do the same for images 'irisB_3.png', 'irisF.png'.
- Did the system work? Were persons A and B correctly identified and person F considered unknown?

# 2. AI based iris recognition

Prepared database:
MMU Iris Database

**Task 1.** Task is based on the following paper:
S. Minaee, A. Abdolrashidiy and Y. Wang, "An experimental study of deep convolutional features for iris recognition," *2016 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*, Philadelphia, PA, USA, 2016, pp. 1-6, doi: 10.1109/SPMB.2016.7846859.

The solution we will implement in this task is quite unusual. The concept is as follows. Normally to perform Iris recognition we would preprocess the images, segmentate the iris and then solve the recognition tasks. You can find a lot of different AI based solutions for both iris recognition and segmentation. So, let's try something less intuitive.

Firstly, let's skip the Iris segmentation phase. During this exercise we will use full images of the human eye. We will try using the VGG-16 neural network pretrained on the Imagenet dataset to extract features. We are not going to train this network - it is already more than suitable for image classification. Then, we will use these extracted features to train the SVM classifier. Moreover, we will use PCA (Principal Component Analysis) to reduce the feature dimensionality. Can we use a DCNN without training to perform different tasks than intended (so called transfer learning) and still get good results? Let's find out.

1. Download the MMU Iris Database and unzip the archive. You will see that there are 90 folders in total, gathered from 45 different people (number 4 is missing :) ). In each folder we have 5 different images of one eye (right & left for each person).
2. Download and open the Python script from UPeL platform. You will see that image reading with proper transformations are already prepared.
3. Load VGG-16 network with weights pretrained on ImageNet dataset: https://pytorch.org/vision/stable/models.html
4. Change the entire VGG-16 classifier to nn.Flatten layer. In this way you will get the 1-D feature vector at the output.
5. In loops extract features from train and test images using the network and save them in a list. In the same manner save labels (you have them already from loaders).
   **Note!** Due to the structure of torches used by pyTorch, features and labels need to be properly "converted" to the NumPy ndarrays. We can achieve this in the following way:
   ```
   feat_numpy = feat.squeeze().cpu().detach().numpy()
   ```
6. Before PCA decomposition we need to change the arrangement of labels. We can do it with ravel method, e.g.:
   ```
   train_labels = np.ravel(train_labels)
   ```
7. Perform PCA decomposition. Firstly create the PCA object with the given number of *n_components* (choose yourself) and then use fit_transform on train features and transform on test features.

8. Use obtained data to train and validate the SVM classifier. Firstly create the SVM (SVC) object. Then use fit on train data and predict on test data.
9. Check the obtained results (from test data) - print metrics.
10. Now you can experiment with the number of PCA components. Save each obtained result in a comment at the bottom of the script. Try to find the best trade-off between accuracy and number of PCA components.

**Task 2 (EXTENSION EXERCISE).** Use iris segmentation like described in the “classical” task and then use the masked images to extract features with the VGG-16 network (as in the “AI based” task). Compare the SVM results for different numbers of PCA components (like in the “AI based” task). Compare obtained results with Gabor features (“classical”) and no-segmentation VGG features (“AI based”).