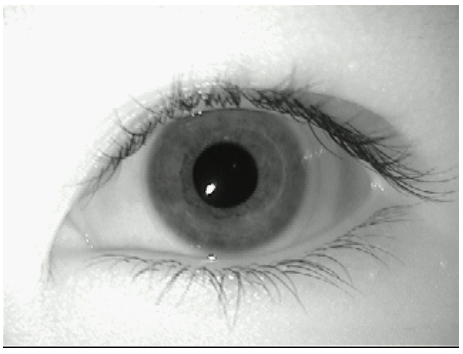
Iris Recognition Report

Digital Image Processing
Professor Shah
Zeinab Kazemi, zkazemi90@gmail.com
Akshay Soni, akshay.soni@live.com
Rimmi Patel, Rimmi.d.patel@gmail.com
Ravi Teja, raviteja94@gmail.com

This is the report of what have been done for Iris Recognition project, this report includes:

- explanation of each function and reasons those functions have been used
- figures of each step
- issues faced
- responsibility of each team member and contribution to each project

For better explanation this sample image from the given data base will be used in this report.



All the functions explained in this report can be found in the submitted directory. For better understanding, we start from the script which is used to run the whole project called loopdir.m. We will go through all the images used in this script in order and explain why and how they are being used.

LoopDir.m:

The script to run is called loopdir.m, in this script, we are going through all the images in each sub directory and compute the signature of each image. There are two for loops for going to the image directory and an extra two for loops for left and right images.

A matrix X is used in this script to store all the gabor codes of each image. For filling this matrix, the function iriscode() is called.

Iriscode.m:

This function is the main part of all the algorithms and functions we used on our images, for each image this function should be called, and then different algorithms should be applied on each image which are the following:

Teta and Rosize:

These two variables are the size of our polar image, Teta is the size of the angles and Rosize is the radius size, which have been set to 100, and 40 respectively.

Localization.m:

This function takes care of finding the iris and the pupil of the image, it takes an image as an argument and return the center of pupil, radius of iris, radius of pupil, the difference of these two centers as ox and oy, and pass it to the next function.

Since the image is not two dimensional, function rgb2gray has been used. After that, a median filter is used to remove the noise from the image. This way the segmentation of image will be easier since there are fewer edges in the image.

In order to find the two circles of iris and pupil, we need to find the edges of the image, there are different functions for edge detection, and we used 'Canny'. by default edge() function uses the 'Sobel' method, so we had to change it as 'Canny'. Canny finds edges by finding the local maxima of the gradient of I, this method uses two thresholds, strong and weak, if the weak edges are connected to the strong edges, it will display it on the output. In our case, Canny worked better than other methods since weak edges are detected, and also it does not get fooled by noise.

Figure 1 shows the edges detected in the image by Canny method.

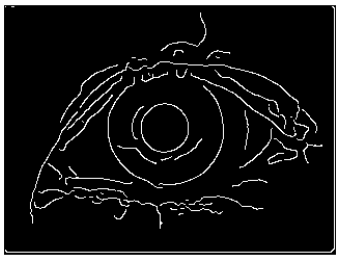


Figure1. Edges detected used canny method.

After finding the edges of the eye, the next step is to find the two circles. Function imfindcircles() has been used for this purpose, it is matlab's built in function and uses Hough transform to find circles. Arguments passed to this function are the edge-detected image along with the range of radius, and a sensitivity variable. This function finds every circle that its radius is in the range. It returns centers and radiuses of circles found. Sensitivity factor is between 0 and 1, as this factor increases, more circular edges will be detected including weak and partial circles.

In the end, after using imfindcircles, twice, our desirable circles are detected. Values ox and oy will be calculated by comparing their centers. Figure 2 shows the detected circles.

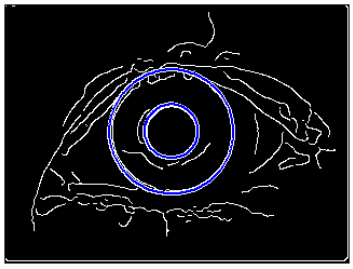


Figure2. Circles detected in the image using Hough transform.

mynormalize.m:

After finding the iris and pupil circles, the image along with the centers and the radiuses are sent to mynormalize.m function. This function is responsible for unwrapping the iris and converting it into its polar form. For this purpose, we used the Cartesian to polar and polar to Cartesian functions implemented in homework 2. The new radius is calculated using the given formula in project description. Since the centers of circles inside the images can be at different locations, we need to calculate the radius at each angle. So the radius size depends on its angle. A for loop for iterating over angles is implemented, and inside the loop new radius is calculated using the given formula.Mynormalize() function received its arguments from previous function (localization.m), and returns the polar image as an argument.

Figure 3 shows the original image in Cartesian system and the image in Polar system. The second image has the size of Teta and rosize, which was covered in the beginning of the report.

Figure 3. Cartesian and Polar image

equalizer.m:

The next step is to equalize the polar image, so the features of the iris become more clear. This function was implemented in homework 1 of the course; it uses probability to expand the features of the iris. Figure 4 shows how the equalized image displays the differences of specific features of the iris.

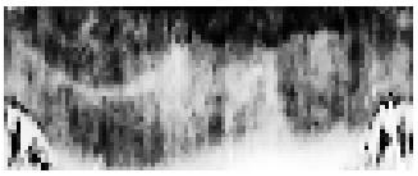


Figure 4. The equalized form of polar image. Differences are clear in this image.

mygabor2D.m:

Finally, after creating a clear polar image, it is time to create a signature of each image, this signature is a matrix of 0s and 1s, that is specific for each image. Images belonging to the same person should have a close iris code and others should be different. The code length in our project is called sizeofvec=4*Teta. This value is the optimal value since at the end, both the false positives and false negative will be the smallest. Different models of gabor wavlet were implemented, at the end we picked the one giving the best result, which is mode 1. This gabor is the same as the formula described in the project. In our case, since the useful information is more around the top of the image, we chose the factors as following:

r0 = 0, teta0 is equal to half of the angle size of image, alpha is equal to the rosize/3, and beta is equal to anglesize/3. Alpha and beta are the Gaussian factors.

In the final image of the iris, there are still some unnecessary information like the eyelashes and the eyelids, and since in our polar image they are all in the angle=0 and angle=2*pi, we picked up the Gaussian factors as described above. These factors are calculated in such a way that more than %99 of the filter only covers the useful parts of the image. That is to say, the eyelashes and eyelids are mostly ignored and have a small weight in the final result.

We defined a vector of sizeofvec*1 size to save all the gabor outputs for different images, there are three for loops in this function and the inner is for iterating over the w factor in the formula. Since the formula results in both imaginary and real parts, there are two vectors for each image, HI and HR respectively. HI and HR have both negative and positive values, and as instructed in the project description, these values are converted to 1s and 0s base on their quadrants. mygabor2D.m returns them as its output and iris code of each image.

Since other models of the gabor wavelet have not been used, we will avoid explaining it in this report.

Figure 5 shows a small part of iris code of a random image generated using gabor wavelets.

Figure 5. Iris code of a random image, including 1s and 0s.

Comparing the iris codes using hamming distance:

Going back to loopdir.m script file, eventually we need to compare the results of iris codes in order to find out which images belong to the same person and which belong to others. For this purpose we

computed the hamming distance of the codes. Hamming distance uses exclusive or to differentiate each pixel of the iris code and at the end will calculate the percentage of different pixels by total. For comparing the images to others, we used matrix X, mentioned before, and calculated hamming distances of every two images. Two factors are defined, a mean and a variance. If the hamming distance of two images are smaller than mean+2*variance, the image becomes accepted as the same person, otherwise we will mark it as a different person. Two thresholds were defined. False positive and false negative, if while comparing images together, one image is considered as similar, and in real the two images are not for the same person, it will be considered false positive, and if we ignore the image which belongs to the same person, we will increase the false negative value.

Figure 6 shows the hamming distance of all the images. At the end the false positive rate is %1.043951 and the false negative rate is %26.888889. In this project, we assumed false negative rate cannot cause as many problems as false positive, so we tried to sacrifice it for better false positive results. Assuming an application of iris recognition like identifying a bank user, if we identify a different person to be the same as another, it will result in lots of damage, but if ignore certain images of the same person, because of image blurriness, noise, etc., we can ask the user to provide us with a better image. By changing the thresholds discussed above, the false positive rate will decrease to 0.0 but the false negative will increase.

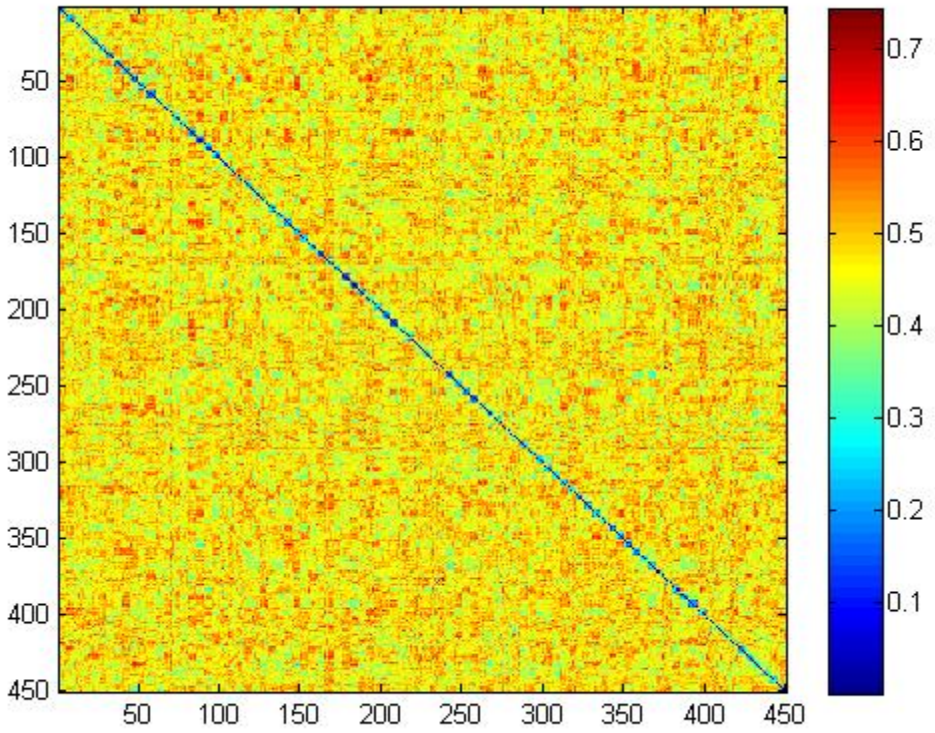


Figure 6. The hamming distance of all the images. Along the diameter of the matrix, the hamming distance of each image with itself is shown in blue (zero).

Figure 7 shows a closer look at the hamming distance matrix. A long the diameter, five images should be selected as similar (five images from the same eye), four blue squares can be seen in this figure, which means the hamming distances between these images are less than 0.2, in the last one hamming distance is a bit higher (around 0.3). Images from the same eyes should have a small hamming distance.

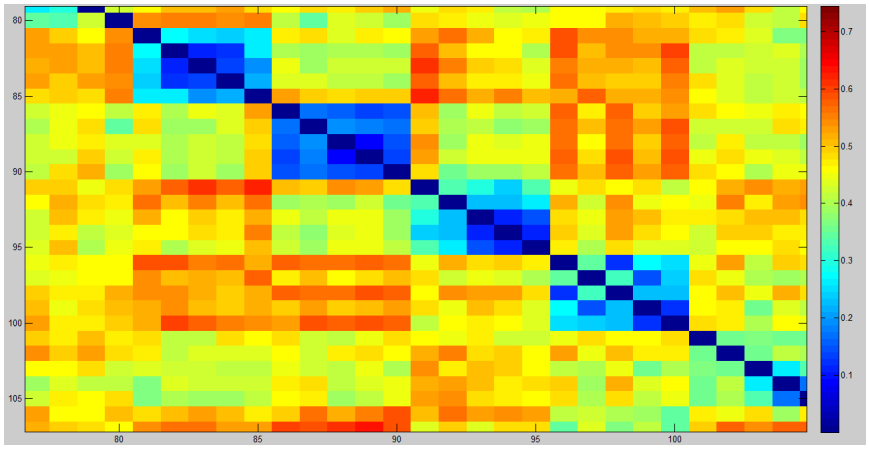


Figure 7. Closer look at hamming distance matrix