Biometrics Project 02 Report

Igor Rudolf 327310

Summer Semester 2025

1 Introduction

During the Biometrics summer semester 2025, I was tasked with designing and developing software that was designed to recognize humans based on their iris images. This report describes the application with theory needed to understand the implementation process.

First, we will describe the project requirements specified by the instructor. Then, we will describe the main code that constitutes the obtained results, including what the program can do and what its limitations are. Then, I will go through the process leading to obtaining the results. That is, I will first focus on the main code, what is our final effect, and only then I will focus on what led to obtaining such conclusions.

2 Brief description of the requirements and implemented part

2.1 What program has to do

The general assumption is that the program is intended to recognize a person based on an iris image. This involves iris segmentation, where the goal is to develop an algorithm for extracting the iris from an eye image. To do this, it was necessary to transform the image to grayscale and perform binarization, then find the boundary of the pupil, iris and then expand the iris to a rectangle.

Then, on the expanded iris, it was necessary to apply the Daugman algorithm to get rid of the partial influence of the visible eyelid in the expansion and eyelashes, then it was necessary to apply the Gabor wavelet transform and the iris code based on it. After that, it was possible to start comparing iris codes based on the Hamming distance.

2.2 Implemented part

Of the given requirements, the only part that has not been implemented is the comparison of iris codes, everything up to the point of determining the iris code has been implemented.

3 Solution description

My solution uses all the fragments provided by the instructor. However, some additional features had to be added to improve the results. I will write more about this in the analysis section. For now, I will simply write my solution, and only later justify my choice.

3.1 Preprocessing

The main goal of preprocessing was to obtain more "distinct" edges. The idea was that in this way the pupil and iris, which take the shape of circles, would be more visible and in this way it would be easier to extract them from the photo. In addition, in addition to increased contrast, I wanted to add local noise smoothing and smooth it (so that there are no sudden jumps in pixel intensity).

For this purpose, first we create a CLAHE object. What it does is divide the image into small tiles ($8x8$) and a brightness histogram is built. Then we add a bilateral filter, which aims to smooth out the noise, the edges are still sharp, it works in such a way that we smooth this noise by replacing the weighted average of pixels from the neighborhood.

Then a Gaussian filter is performed. The assumption is that after a bilateral filter there may be such a problem that on even surfaces (small differences in pixel intensities), there is still noise, in the high-frequency sense (I mean such "teeth" in the brightness levels that make such jumps "rough"), Gauss smoothes them.

3.2 Pupil detection

First, we perform the binarization operation. As for the binarization threshold (calculate_treshold), I set it globally, in the sense that the goal is that mean is the average brightness after preprocessing, the goal is to be low enough that the pupil becomes a white area and the iris becomes black.

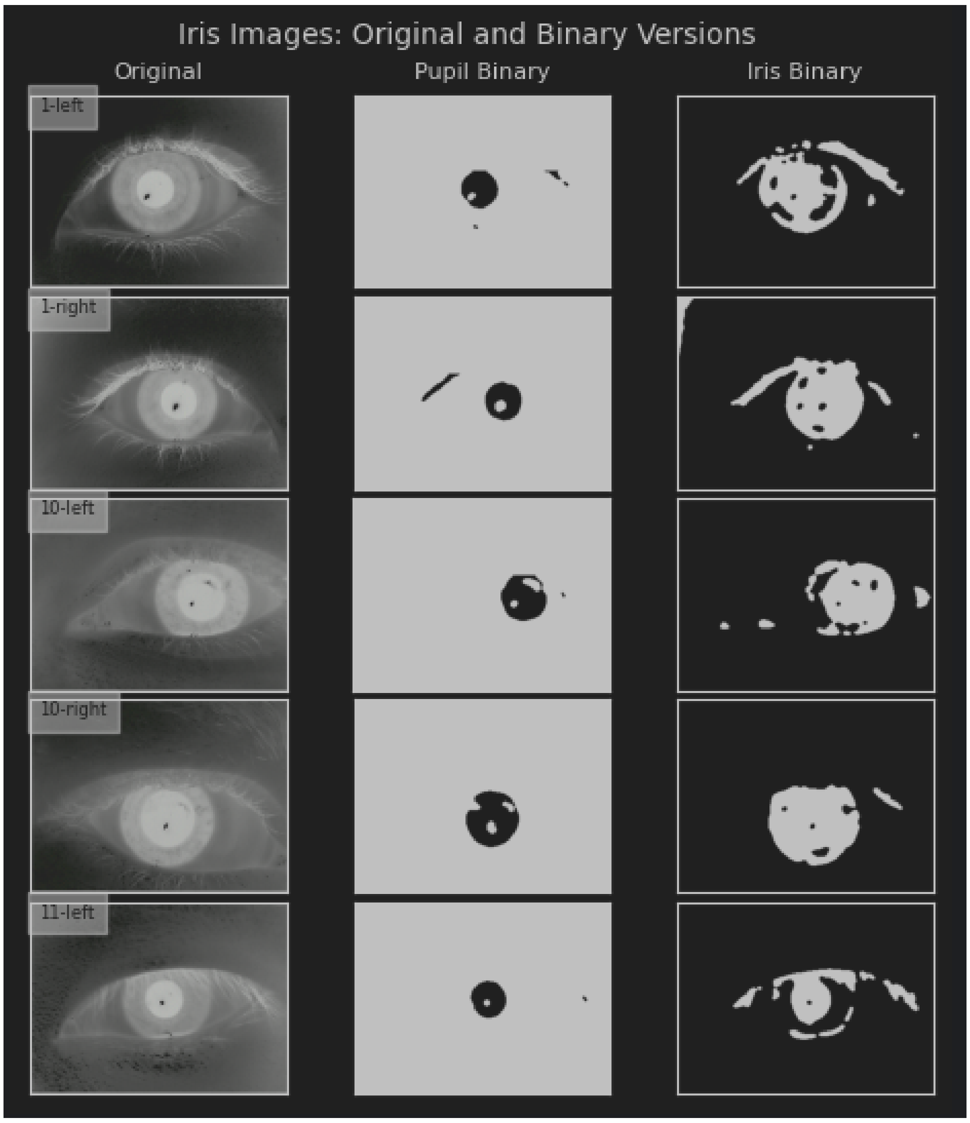

Figure 1: Sample photos, before and after binarization, a few examples.

Then what we do is use the calculate_projections method, which gives two one-dimensional arrays, exactly the sum of white pixels in each column for the vertical and horizontal projection. The goal is that on this basis we can determine the initial position and size of the pupil.

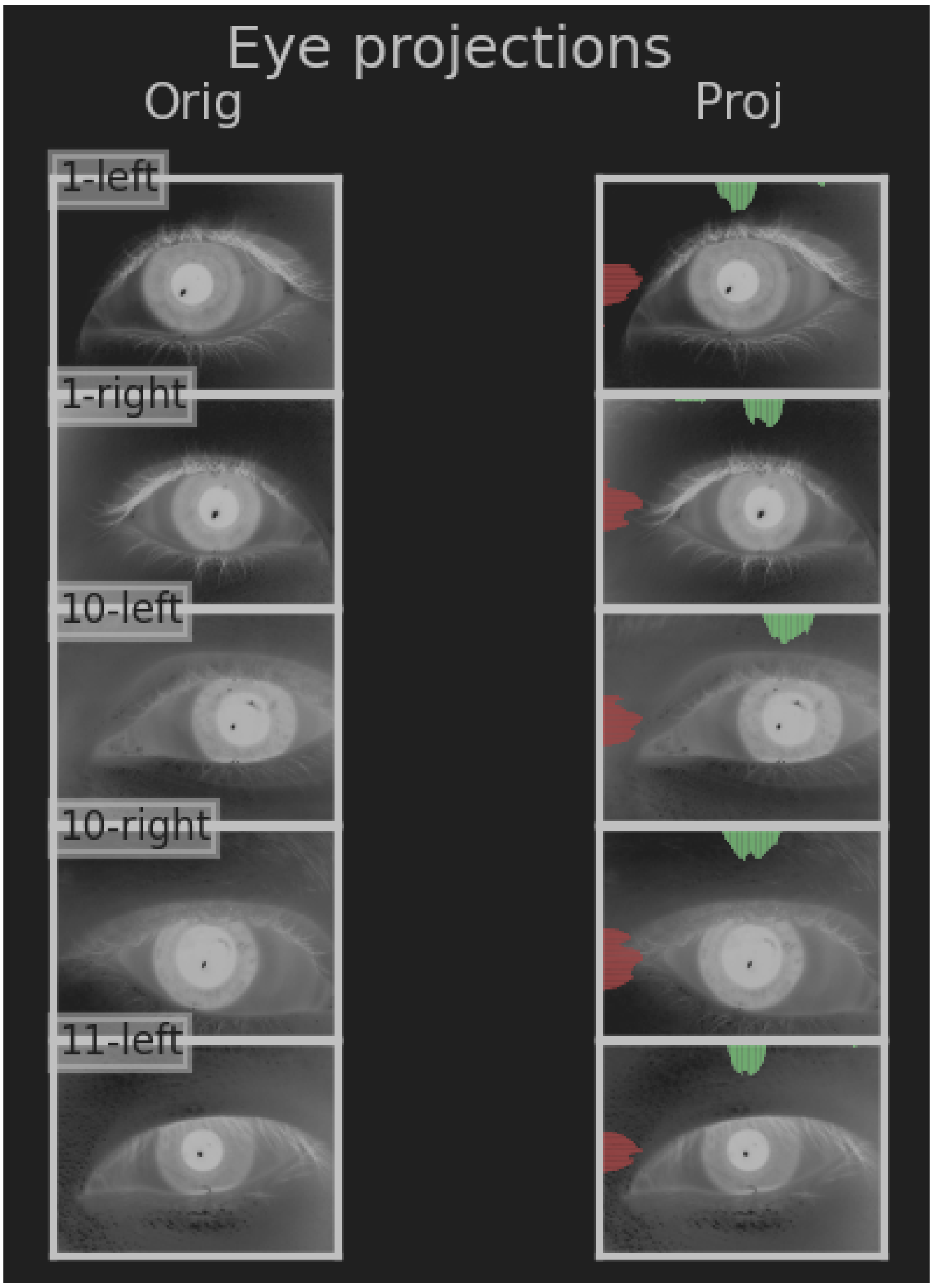

Figure 2: Photo of the vertical and horizontal projection graphs plotted on the eye, something like in iris-to-rectangle, but without a visible intersection point.

Then we use the find_center_by_projections function, this is a method whose main goal is to determine (cx, cy) in the sense of the coordinates of the center of the pupil based on the projection.

Now what we do next is to determine the centers of gravity of our projection. We do not simply take max or min, it is useful in the method: detect_pupil_boundary. What we do exactly is to treat white pixels as a value equal to 1. Next, we sum the intensity of the pixels that interest us. We calculate the center of gravity in the x axis as C_x from the formula:

$$c_x = \frac{\sum_{x=0}^{w-1} x\, H(x)}{\sum_{x=0}^{w-1} H(x)}$$

Now if we are talking about y, we use the weighted average again and have the formula for c_y as

$$c_y = \frac{\sum_{y=0}^{h-1} y\, V(y)}{\sum_{y=0}^{h-1} V(y)}$$

where here x and y refer to the appropriate column and row indexes.

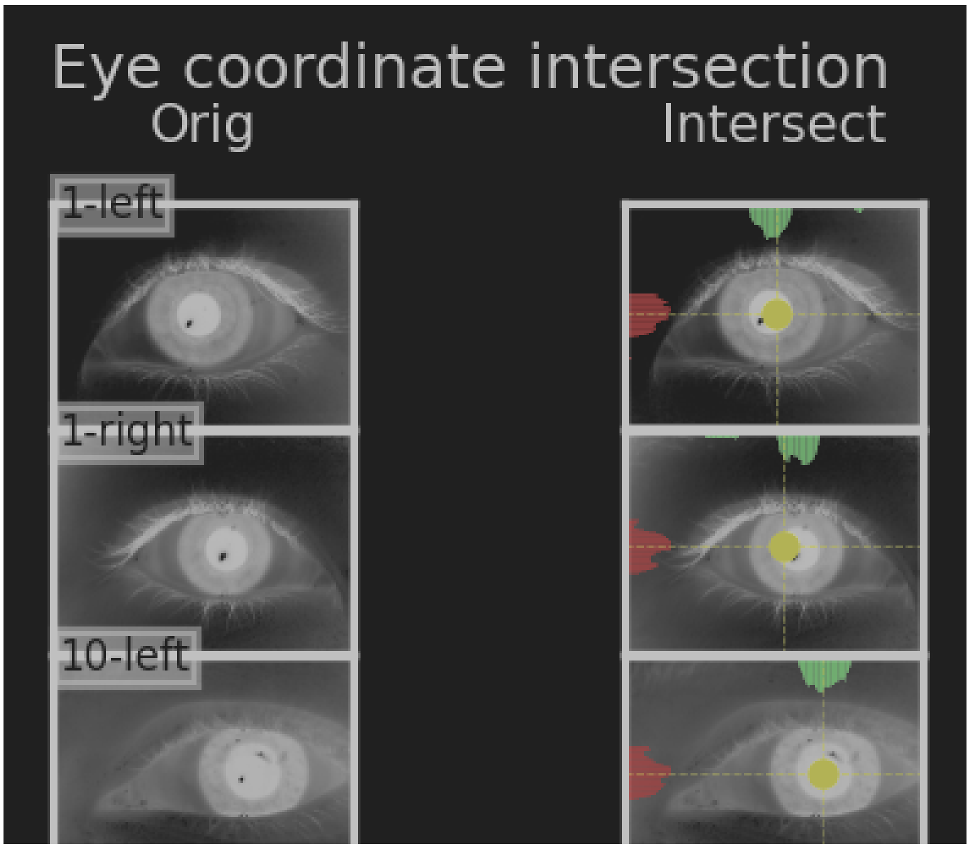

Figure 3: Example of photos with determined pupil center by projections.

Then we use the `estimate_radius_by_projections` method. Having the center of the pupil (cx, cy), we estimate the circular radius of the object. It works like this: we take the entire row and column corresponding to cx and cy, we go from this center to the left as long as there are white pixels in the binary mask, we do the same going to the right, down and up relative to the center (cx, cy).

We go as long as there are white pixels, when we come across another one (information that we come across the iris), then we stop and count the average number of pixels that we have determined from going in each of these directions. This average is information about the pixel radius.
Important note. Looking at a lot of photos from the MMU Iris dataset, we can determine in which regions the pupil is located more or less, so what I do before starting the operation is to throw out 40% of the top area of the image that could contain eyebrows. This way we analyze the code on a much smaller region. The logic of this approach is relatively simple. First of all, the MMU Iris Dataset contains photos where eyebrows are visible. Without adding this fragment, the algorithm made errors relatively more often.

Now we use the `detect_pupil_boundary` method, which uses the fragments mentioned above and a few others that I will write about in a moment.
First, I cut out our area of interest, which we will call `ROI` (I mean leaving 60% of the area), we perform the mentioned binarization. Then we perform morphological operations `OPEN` type 3×3 and `CLOSE` type 7×7. The assumption of the first is to get rid of small spots from the pupil areas, which may be e.g. from the reflector. The second one is responsible for filling in the gaps in the diameter.

After that, it is the initial determination of the center of the pupil (we use the projections described earlier here). Now the important thing, I wrote that we cut off the upper part of the image because there is no point in considering it, now that we have the coordinates of the center of the pupil, it is for a small area, so now I take into account the offset to have these coordinates of the center of the pupil but relative to the entire image.

Now what we do is on the binary mask we use the `CCL` algorithm to extract coherent structures, now what we do is we calculate the smallest distance of each component to the previously determined center, the component from which the distance is the smallest is our pupil center.

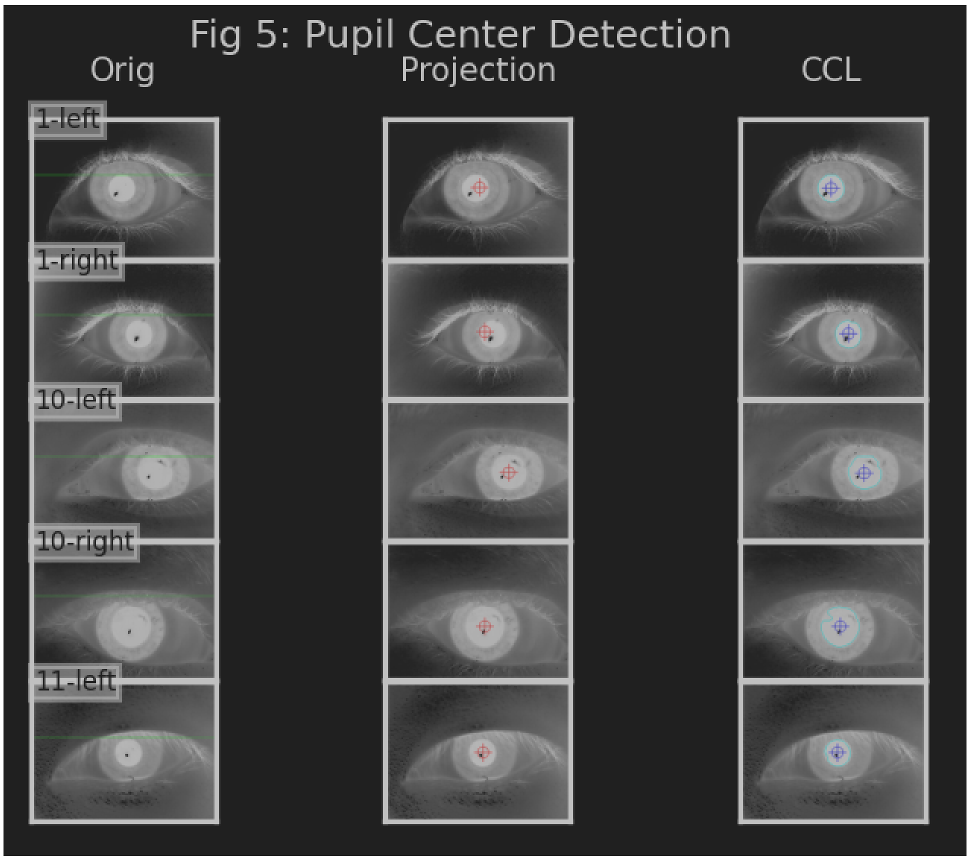

Figure 4: Examples of the centers determined by the initial projection and the CCL algorithm.

The method of determining the radius has been described above. We go in each direction relative to the radius (left, right, bottom, top).

3.3 Detecting the iris

Now, when it comes to the center of the iris, it is always in the same place as the center of the pupil. I described the method of determining the center of the pupil in the previous subsection.

Now, when it comes to the radius of the retina, it is determined by first cutting out the area of interest, which I defined as a square with side $2R$, but the maximum radius we consider is $4.5R$.

Then we use the edge map, which shows the gradients of brightness changes in the neighborhood of a given pixel. What is done is to calculate the derivatives of brightness G_x and G_y, here we use the Sobel operator with size 3×3. Then we calculate the gradient, which is defined by `edge_map`.

After that we use Gaussian Blur, keeping in mind that it also aims to reduce noise after applying Sobel.

Next we use normalization to scale the gradient to the range from 0 to 255. Later we only consider ranges from about 1.5× radius to 4× pupil radius.

After that, we can move on to the most important part of the algorithm. What is done now is that having this search area and the gradients calculated, we look at it as a circle and go every 5° and choose the distance that corresponds to the largest gradient. From all these rays, we calculate the `MAD` statistic, and we discard the outliers, which here can be understood as all rays greater than 2 × MAD.

Then we calculate the median again and treat it as the average retinal problem. However, if none of these rays has been detected, we try to determine the radius by summing these gradients. Here, a note that there is always a limitation, that the iris radius must always belong to the interval $[2r, 4r]$, where r here is the pupil radius.

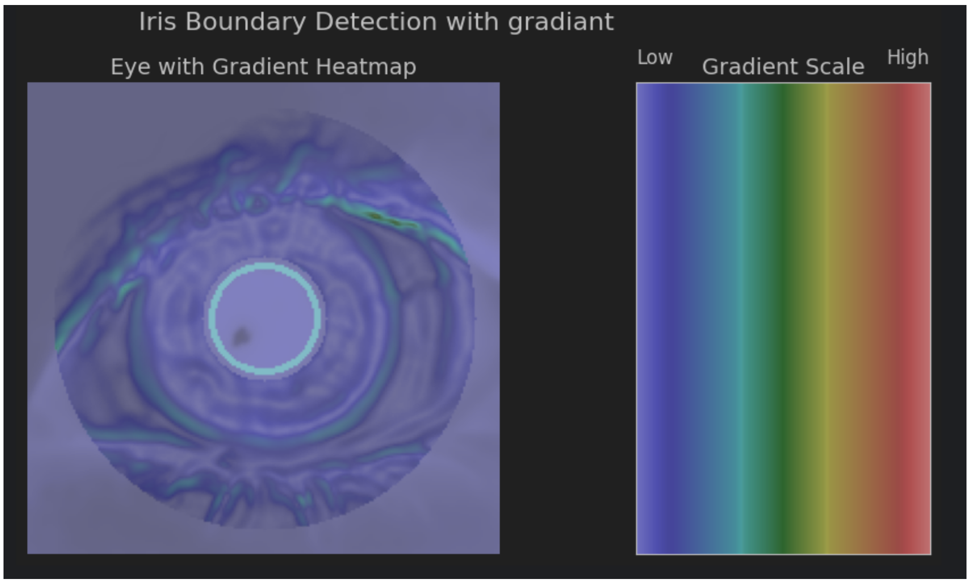
Iris Boundary Detection with gradiant
Eye with Gradient Heatmap
Low
Gradient Scale
High

Figure 5: Photo example of the changing gradient.

3.4 Iris Unwrapping

Within this fragment we have several methods that are key to the operation of the entire program. First of all we have the `unwrap_iris` method, which changes the area from the pupil boundary to the iris boundary, from our system, defined by (r, θ), to a system in which we have columns and rows.

You can look at it, that the row is minimal, i.e. 0 at the pupil and the maximum value (the last row), at the last fragment of the iris. Exactly what is done is create a matrix with zeros, dimensions 360× the distance between the pupil and the end of the iris. We then copy these pixel intensities to this matrix. In this way we can visualize a typical expansion from polar to rectangular.

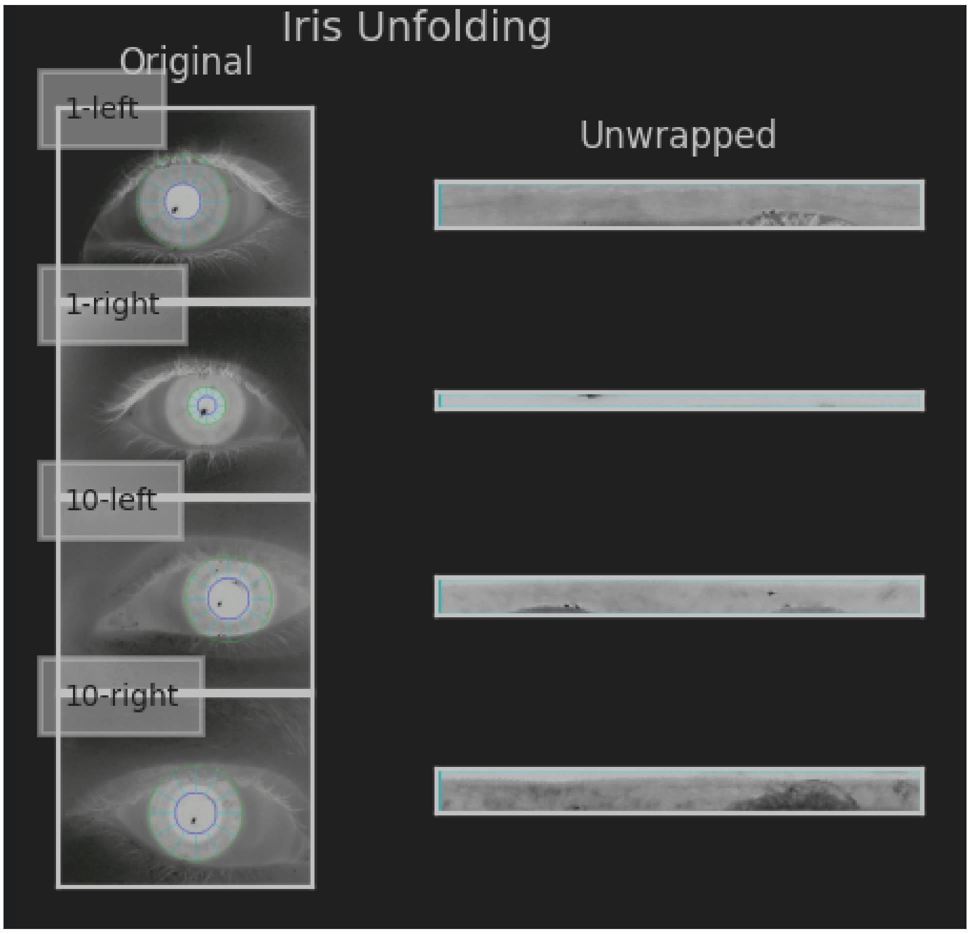

Figure 6: Example of the photos unfolded to a rectangle.

Another more important method used is `unwrap_iris_daugman`, here we use to unfold the iris according to the Daugman model. This means that we divide the radial area into 8 rings, which we will call rings. For each ring, we are only interested in its part, that is, we are only interested in the angular part, which in the code is called `ang`.

The point is that if we used whole angles, we would include the eyelids along with the eyelashes. In this way, we create a mask for the allowed fragments and ignore the rest. Now, additionally, after unfolding the iris, we can remove the fragments that do not enter this iris and after applying the Daugman algorithm are still visible. This concerns various eyelashes and reflections from light.

Above is a photo with 8 rings marked, each of them (except the last eighth) occupying
As for reflections, the way of detecting them is relatively simple. By reflections we mean all those areas whose brightness is $> \mu + 2.5\sigma$.

After that we perform dilation on a 25×3 kernel, after 2 iterations. The goal is that in this way thin and broken areas of eyelashes become uniform areas.
Finally, we perform the morphological operation `CLOSE`, which merges close regions into one. We return the iris brightness mask and the binary pixel mask, which represents the iris.

The important thing here is that in the above area the marked black areas are ultimately those that are to get rid of the eyelid and at least partially the eyelashes. In reality, however, the matter is not so simple and the visible black stripes have been appropriately shifted to the left/right so that they uncover one area and cover the other.

What we do next is cut out, for each ring we cut out the areas marked in black, connect them with the remaining areas and then, with the notes that each of the rings after partial cutting and connecting may be of a different size, we resize to the dimension 128×8.

Below is an example photo of how it looks in practice.

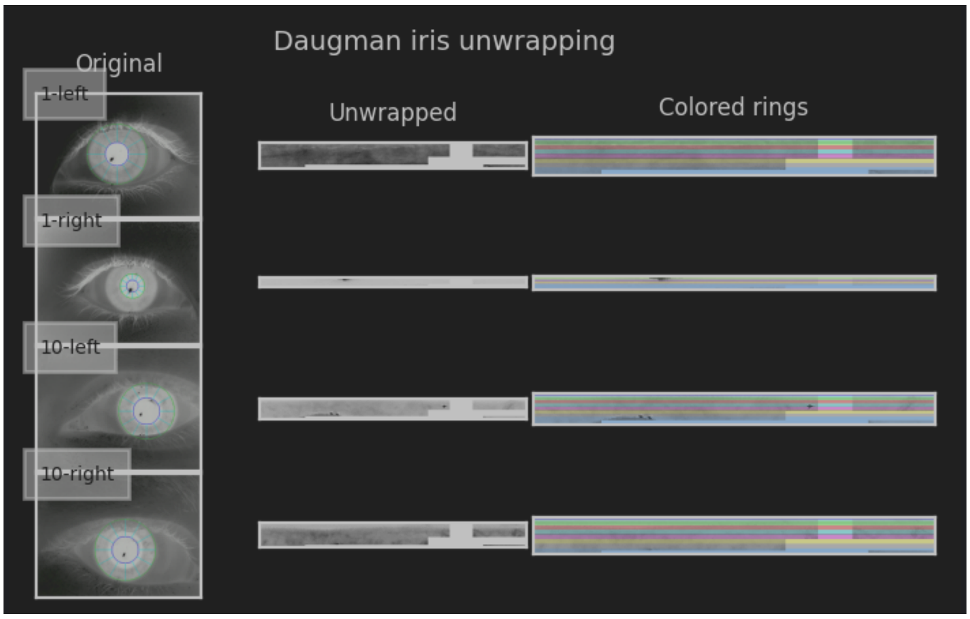

Figure 7: Examples of photos with marked Daugmann rings.

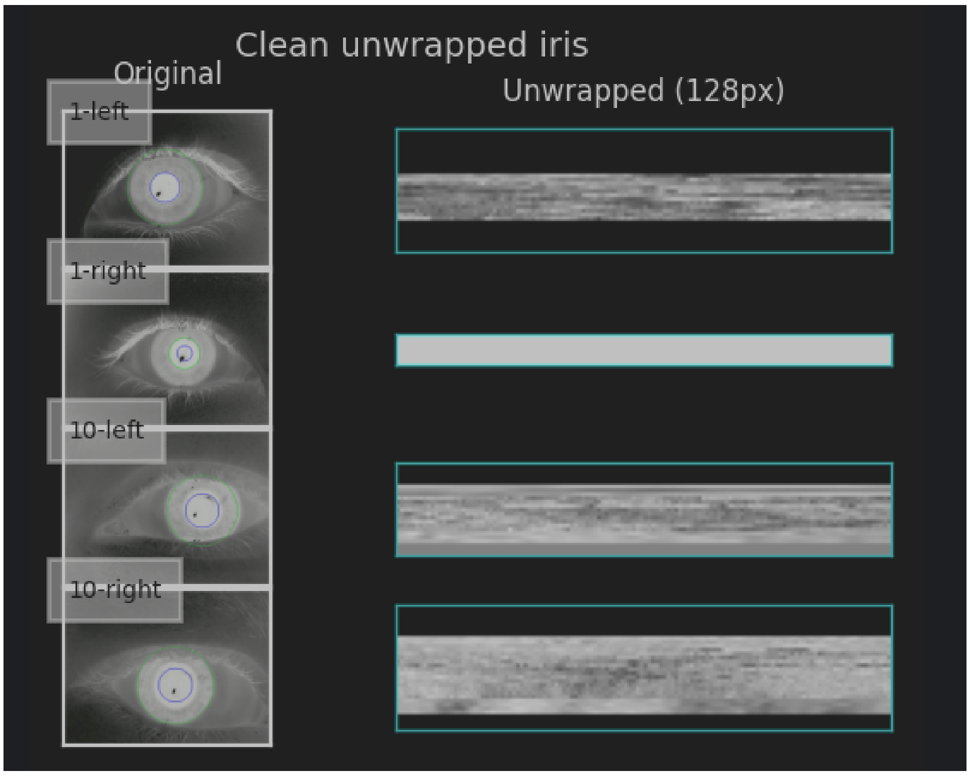

Figure 8: Example of images with cleaned iris.

3.5 Iris Code

Here we use three functions `extract_ring_signals`, `gabor_kernel_1d` and `compute_iris_code`. `extract_ring_signals` for each using a Gaussian kernel, we take into account the rapid changes of the 8 rings, we perform weighted radial averaging of the p brightness, but we will have 128 pixels representing the brightness of each of the angles in a given ring, we smooth with a Gaussian kernel so that these changes in brightness are evened out and the changes are not so rapid.

Then what happens is we use a Gabor filter so that each of those eight rings is filtered with a Gabor kernel, using a frequency that's labeled `gabor_f` in the code and can be manually controlled by the user, and then we filter based on those three frequencies as `gabor_f` $\times$ 0.7, `gabor_f`, and `gabor_f` $\times$ 1.3, and those low and high frequencies are used to capture different shape changes in the image.

After using these three frequencies for filtering we will get a result for each pixel that contains a real and complex part. What we do is not remember the entire answer but only the sign of the answer, if the part $\mathrm{Re}(x) \geq 0$ we give bit 1, otherwise bit 0, the same for the imaginary part.

In this way for each ring and for each pixel in it we have information about bits, on these three different frequencies we build such bit masks. We build such a tensor map of size $(8, 128, 6)$, 8 rings, 128 pixels in width and 6 layers, because 3 for different frequencies, then we flatten the obtained result and we can visualize it.

Below is an example photo of the obtained iris code.

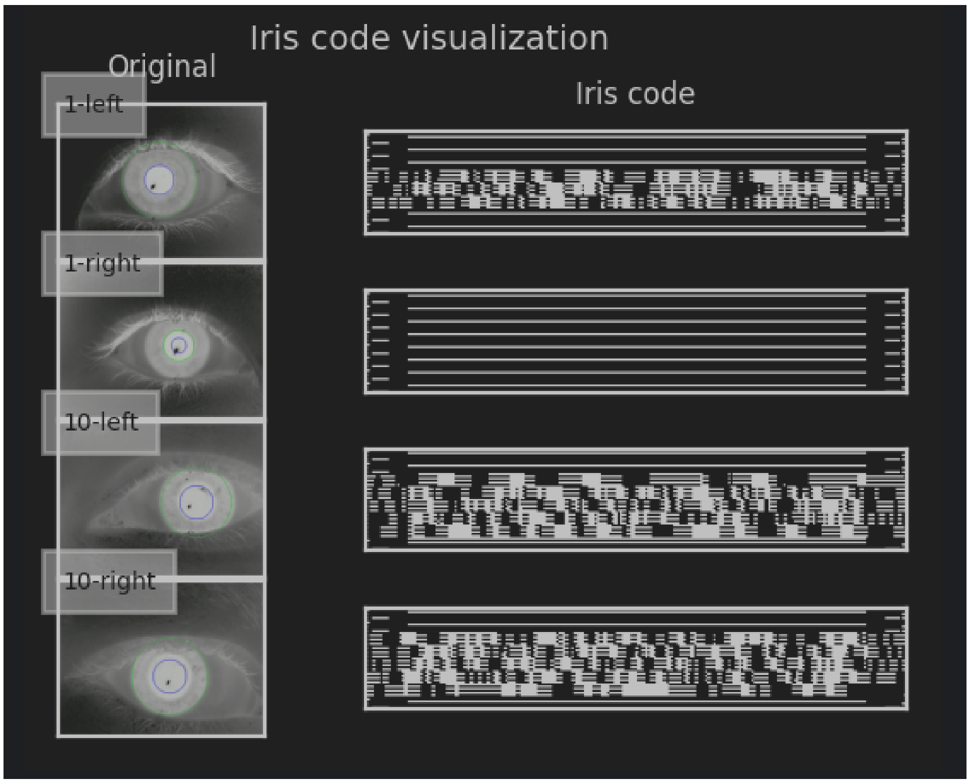

Figure 9: Example of the photos with obtained iris code.

3.6 further work

Although the presented method does quite well in detecting the pupil and iris (although there are cases where it is wrong, which will be discussed below) and we have obtained the iris code for each eye, at this stage what remains to be done is to test these results, however due to other projects this part was not done.

4 How I chose the algorithm and analyzed preliminary results

In this section I will describe how I came to the conclusion that this algorithm gives the best results, dividing the analysis into three parts. In the first I will present how I came to the choice of the pupil detection algorithm, I will show the results of previous algorithms and the results of the current algorithm, then I will do the same for the iris.

4.1 How to detect the pupil

Basically I tried to detect the pupil in four different ways.
In the first approach I focused on global thresholding with the addition of morphological operations.

I loaded the image, applied conversion to grayscale, then applied Otsu's method with the binarization threshold, then from the morphological operations I performed the `Open` operation on the 3×3 kernel and `CLOSE` on the 7×7 kernel. Only after that I started determining the center of the pupil. In this approach I used horizontal and vertical projection to find the center of mass, in the case of the radius, I went from the center (center of gravity of the projection) to the left, right, up and down and counted white pixels to determine the radius and then I calculated the average from these four directions (I used this part of the solution in the final solution).

The problem with this approach is that here the binarization threshold is global and results from the image not yet processed, i.e. from Otsu from the raw brightness histogram. For this reason, it was not completely visible in some photos, which could be the reason for the threshold being set too low.

Hence, I decided to move on to the second solution, in which I used adaptive thresholding, i.e. each image would have its own threshold set, not a global one, and in this way I thought it would be possible to better adapt to differences in the brightness of image fragments.

What was done in this technique, apart from the standard conversion to gray, was also this approach with an adaptive threshold, which consists of:

Later, as I used the `OPEN` and `CLOSE` operations early on, we know that some larger areas can still appear on the mask, which do not necessarily have to be the pupil. For this reason, this time I decided to decide to use the `CCL` algorithm and I choose the area with the largest surface area, which potentially has the greatest chance of being the pupil. All pixels that were not classified as belonging to the pupil (`CCL` tells which elements belong to a given component) I simply zero them. After that, I determine the radius in the same way as in the first approach.

However, I also decided to abandon this solution because this adaptive thresholding, although on paper it seemed like a good solution, in practice, while adaptive thresholding used different 11×11 size windows, when a fragment of the window covered the iris and other fragments neighboring the eyelid, then the transitions, in the first case the binarization threshold will be very high, in the second very low, and then we have such sudden jumps. From what I managed to determine, this was a problem with this solution. However, I will use the fragment with the `CCL` algorithm in the final solution.

Approach 3 was a bit different, instead of doing such artificial thresholding, I wanted to use the `k-means` algorithm to determine this division window myself. Instead of using the 11×11 division myself, I preferred to group into such clusters, into dark and light areas. After grouping the areas into these clusters, we choose the area that is the darkest as the pupil, the rest of the solution is consistent with the third one.

The transition from the second to the third solution solved some of the problems, but did not solve them all. After switching to clustering, it turned out that there were also problems. First of all, in `k-means`, you have to choose the number of clusters, which was not always so simple, and even so, the areas under the eyelid often appeared in clusters grouping darker areas. There were no generally best parameters that would work well.

Moving on to the fourth solution (final), I wanted to take what seemed to me to be the best of these three solutions and simply instead of making the binarization too complicated I decided to do one global binarization, using `CLAHE` and `Gaussian Blur` beforehand, and instead of taking the threshold manually or using `Otsu`, I decided to do a global average of all pixels. I described the rest of the approach above. It turned out that the fourth approach was the best.

4.2 Method of detecting the iris

Here I will also describe four approaches to the subject. In the first method I used the same threshold as for the pupil. Next I used the morphological operations `Morph OPEN` and `Morph Close`, after that I used the `CCL` algorithm similarly and projection, similarly as for the pupil.

The problem with this approach was that often the circle that decided the iris area was too small or too large, basically the same problem was in other methods. For this reason I moved to the second approach to the problem, which consisted, as in the case of the pupil, of adaptive thresholding, but this also did not give satisfactory results.

The third solution assumed the use of `k-means` taking into account the area of interest `ROI` with the application of morphological operations. This approach somewhat combines this fourth final approach because we consider the area of interest after we know where the center of the iris is (similarly as in the fourth method we determine it, i.e. as the center of the pupil), and we take a search from the range of $2r$ to $4.5r$, where r here defines the radius of the pupil.

On this `ROI` area we use the `k-means` algorithm, and I took the cluster that best corresponds to the average intensity value, in the sense that intuitively it is the best, because the pupil is black and the fragments outside the iris, unless they are eyebrows or eyelashes, should also be lighter than the iris. On the cluster where the iris is located, the `CCL` algorithm is used, which detects the iris component, in short we skip small fragments that could be mistakenly recognized as the iris.

5 Things to consider for the future

First of all, there is still the comparison of iris code results, which has not been implemented yet and is a step to consider in the future. It would certainly be worth using these Hamming distances, checking for different images of the same eye whether it detects the same person correctly, and how it compares to other people, and confirming the assumption that each person has a different iris.

There are certainly also parts of the algorithm that could be improved, such as: sometimes detecting a pupil that is too small and therefore also a much too small iris, and then most often everything fits within the pupil. This is one of those parts that could be improved.