Embedded Iris Recognition for Secure Biometric Identification using a 2D Gabor Wavelet

Jannes Thomas Koopmans, *s2855437*

***Abstract*—Iris recognition is a proven form of identification, providing more accurate results than other forms of biometrics like facial recognition or fingerprints. Yet its commercial impact has been limited so far due to system costs and size. This report presents the development and evaluation of an iris recognition system that makes use of a readily available and affordable Raspberry Pi 4. The hardware also includes near infrared image capturing using an Arducam OV9281 NoIR camera module and near-infrared illumination to visualize iris features that are invisible using visible light. Iris localization is done using Python and OpenCV and feature extraction is performed by convolving the normalized iris image with a 2D Gabor wavelet. The resulting iris code is matched using Hamming Distance. Results show the system can achieve reasonable accuracy at a 0.8% false positive rate when the iris is successfully detected but still faces challenges with consistent iris edge detection. This holds for both images from the CASIA v1 iris dataset and live images made by the device. Overall, the algorithm proved to be secure and usable as a verification system, yet more research needs to be done to be able to use it as a means of identification.**

I. Introduction

Automatic identification and one-to-one verification of human individuals have been of interest for researchers and institutes for a long time [1], [2]. For this, biometrics are often used. In current forensics, fingerprints and DNA are essential [2] and today almost everyone has a fingerprint scanner on their phone, which makes verification of the user and unlocking the device easier than ever. Similarly, face or voice recognition can be used. A lesser-known alternative is iris recognition. It turns out that iris recognition can be far more accurate (less false matches or false rejections) than using fingerprint scanners or face recognition [1], [2], while maintaining roughly the same identification speed of less than a second. According to John Daugman in [1], depending on the decision requirements, one could set the threshold in such a way that there is a 1 in 13 billion chance that if there is a match, it belongs to another person. Accordingly, iris recognition has the potential to not only become a more accurate one-to-one verification method but also as passport-like identification. This could be an added security measure to identification systems currently in use. Furthermore, it might also add to the forensic possibilities that researchers and detectives currently use, as it is in some cases possible to ID persons in online images, when of high enough resolution, or even to some extend deceased people, should a database be constructed [2]. The forensic use without such images is limited however, as the iris does not leave residue behind on a scene. Commercial products, such as the Panasonic BM-ET200 and the LG IrisAccess 4000 series all use the algorithm made by Daugman [3], [4], which is described in [1]. However, most of these Machines are large, heavy and not really portable. The aim of this research is to design a functioning embedded iris recognition system. For this, the following research questions are formulated:

1) What physical components are required for a cost-effective and compact embedded iris recognition system?
2) What are the key steps for developing an efficient iris recognition algorithm?
3) How can the iris features be extracted the most accurately?

II. Related work

A. How iris recognition works

The following section is based on the algorithm by Daugman in [1], as well as on the adaptations made by [2]–[4].

1) Image capture: The first step is to create an image of the iris and localizing the iris within that image. As described in [1], [2], for this a minimum resolution of 10 pixels per millimeter is required, though more is advised. The final rating is mostly influenced by the picture with the least resolution [2]. This image commonly, captured on Monochrome CCD cameras in the 700 to 900nm wavelength range [1], since Near Infrared (NIR) light is the most optimal to capture the iris features. The specific wavelength can vary amongst research. [4] describes that 850nm wavelength yields optimal results, yet a more recent paper [2] describes that among tests in the range, 910nm gives the most accurate result. The choice of NIR lighting is due to melanin pigments in darker eyes. These absorb visible wavelength, which makes it impractical for use [2].

In order to check if the image is in focus, Daugman explains that the energy of the higher frequency components in the image can be measured. The image with the most energy in that region will be the most in-focus image. This can be done using a calculation heavy 2D Fourier transform, but also using an equivalent algorithm as described in [1], which convolutes the image in the image domain.

The sharpness of the image might also be determined using Purkinje spots [4]. Purkinje spots are reflections in the pupil area of the used NIR spotlights to illuminate the iris. They are known to have their focus point in the pupil itself, which is at the same distance to the camera as the iris. Hence the iris was the most in focus when these spots would be the smallest. [4] For this to work, one would need a dedicated array of NIR spotlights.

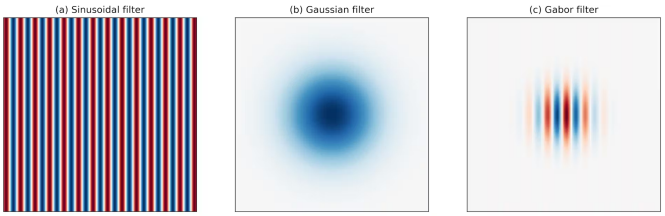

Fig. 1: construction of a Gabor wavelet, from [5].

2) Locating the iris: To know the location of the doughnut shaped iris, the location of the limbus (outer boundary of this iris) and the pupil have to be found. For this, the operator given in 1 is used [1].

$$\max_{(r,x_0,y_0)} \left| G_\sigma(r) * \frac{\partial}{\partial r} \oint_{r,x_0,y_0} \frac{I(x,y)}{2\pi r} ds \right| \tag{1}$$

As explained in [1], this can be used to determine the center coordinates and radii of both the pupil and limbus by looking at the change of intensity over radial lines, which will be the largest for the change between pupil and iris and between iris and sclera, the white part of the eye. A similar approach is also used to find the eyelid boundaries, the details are discussed in [1]. These boundaries are used to determine which sections of the image should be used when matching and to guide the feature extraction.

3) Feature extraction: To differentiate between irides, phase information is used (contrasts relative to other parts of the local iris, such as crypts, freckles and furrows) since merely amplitude or intensity information depends on other factors and is therefore not discriminating enough to identify people. For this, the image is first normalized using the rubber sheet model, stretching the iris into a rectangular shape, which is normalized from 0 to 1 (pupil to sclera) on the y-axis and from 0 to 2π, or around the iris, on the x-axis. [1]–[3]. This action counters dilation of the pupils inherently, as the iris always gets stretched to the same size.

[2] notes that large dilation can give some folding of the iris tissue, which the model does not account for. The presence of ambient light is therefore advised.

Since phase information is the desired feature, extraction is done by convolution of the image with a 2D Gabor wavelet. A 2-D Gabor wavelet is best explained as the combination of a sinusoide and a Gaussian filter, as can be seen in Fig. 1.

From [6]: The Cartesian form of 2D Gabor-wavelet as defined:

$$H(x_0, y_0) = I(x_0, y_0) * G(x_0, y_0)$$

$$= \iint_{xy} I(x,y) e^{-iox'} e^{-0.5\left((x_0-x)^2/\alpha^2 - (y_0-y)^2/\beta^2\right)} dxdy$$

Where:

$$x' = (x_0 - x)\cos\theta + (y_0 - y)\sin\theta$$

Here α, β and ω are constants, depending on the wanted wavelet. The output of the Gabor filtered image has both a real and an imaginary part. By looking at the sign of these outputs, the phase quadrant of each region can be determined by 2 bits, as demonstrated in Fig. 2. This process is repeated over the entire iris, which in the case of [1] yields 2024 bits, or 256 bytes.

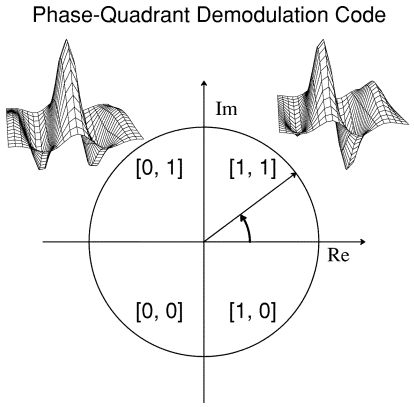

Fig. 2: Phase quadrant demodulation, copied from [1].

4) Matching: These bits, the iris code, hold the information of the resulting calculation per phase partition. To find the (dis)similarity between two irides, Hamming Distance (HD) is used. HD is a measure of dissimilarity between two bit vectors. In the case of Daugman, the HD is calculated as in **??** [1].

$$\mathrm{HD} = \frac{\|(\mathrm{code}\, A \bigotimes \mathrm{code}\, B) \bigcap \mathrm{mask}\, A \bigcap \mathrm{mask}\, B\|}{\|\,\mathrm{mask}\, A \bigcap \mathrm{mask}\, B\|} \tag{2}$$

In the denominator an XOR operation is performed on both iris codes, meaning that when they are not equal, this results in a binary 1. The AND operations with both masks make sure that only the relevant areas of the iris, those which are not covered by eyelids or obstructed in other manners, are measured. The denominator then sums the total relevant area, resulting in a normalized HD. Here 0 is an exact match and 1 is exactly unequal. Hence, the closer one is to 0, the more likely it is that the irides match.

5) Statistical independence: In general, the phase information of any iris random and unique for each individual and thus for each bit of information there is a 50% chance that the bit in question will be a 1. This can also be modeled as a coin flip or a Bernoulli trial with p = 0.5, N=249 [1], which is used to prove the statistical likelihood of a certain HD score by binomial distribution. To prove that an iris matches with another scan from a database, the score does not have to match exactly the HD should just be less than a certain threshold, as it is statistically extremely unlikely that two uncorrelated irides result in a HD of less than 0.3: 1 in 1.5 Billion [1], which can hence be used as a decision threshold This fact gives the HD analysis of the iris code a certain robustness, as small deviations due to lighting differences and sharpness issues will have less effect on the rejection of an image. In fact, [1] shows that when using a lab setup, in 90% of the measurements taken the HD is 0.0 when two scans of the same iris are taken. Using different setups resulted in an average HD of 0.1 (SD = 0.065), which is within the proposed threshold of 0.3. Even if the database somehow becomes too large, the threshold could simply be adjusted downward to reduce false positives.

6) Effects of rotation: in [1], Daugman also mentions that the iris is not always perfectly in the same phase. This might be due to a change in the angle of the test setup, but most often, the angle of the iris changed, which might be to the person taking a different stance, tilting their head slightly, or even just having their eyes in a different position. When this happens, the iris code also changes, as the code is relative to 0 angle. But since the iris code is radial, a simple bit shift is enough to compensate for this, in general about 3 shifts in each direction. Calculating the hamming distance for each of these shifts and using the lowest is all that is needed to compensate for bit shifting. This rotation does influence the statistics however, since in the code, each bit is now ever so slightly more likely to be a 0, Shifting the expected value towards 0.

III. Method

As explained by Daugman, iris recognition can be divided into 4 different steps. First an image has to be captured, which should be assessed on quality. After this, the limbic and pupillary boundary should be located to determine the location of the iris. When the iris is located, the features should be extracted, these are saved in a binary code. Lastly, this code can be compared with other codes to determine similarity between irides. In the following sections, we will discuss the algorithm step by step.

Iris detection algorithm

A. Localizing the iris

1) Iris: Before processing the image, histogram equalization is applied, using the OpenCV CLAHE method. This makes it easier to set the thresholds needed to find the iris. In order to detect the limbic and pupillary boundary, Hough Circle Transform is used, as described in [7]. This option is chosen, mainly due to time constraints as it is easy and quick to implement. For this, some pre-processing is required in the form of edge detection. Hough Circle Transform relies on a Canny-edge detected image, which is a binary edge map of an image. Edges that fall above a threshold are set to 1, everything else is 0. To perform Canny edge detection, the image should first be blurred to remove high-frequency components, as the interest lies in the larger transitions, such as the limbus and pupil. For this, a 9x9 Gaussian blur is used.

The algorithm will try to fit circles on depending on a few constraints such as size, position and a score threshold. It does this by drawing circles on the image and counting how many binary ones coincide with the circle. Using two different instances of the Hough Circle Transform method, both the pupil and limbus can be found. However, the algorithm might find multiple candidates for these boundaries as multiple scores lay above the final threshold. To combat this, multiple checks are performed. First, we make use of the pupil which reflects no light. Therefore the grayscale value is close to 0 (black). When pixel value of the center of the circle is close to 0, it might be a good candidate for the limbic or pupillary boundary, so they are saved. Then, the distances between all the candidate circles for the limbus and pupil are cross-referenced. Those which are the closest together are the most likely candidates for the limbic and pupillary boundary, which are used to determine the location of the iris. The algorithm might not always directly find good candidates, so the function might be called multiple times using different parameters.

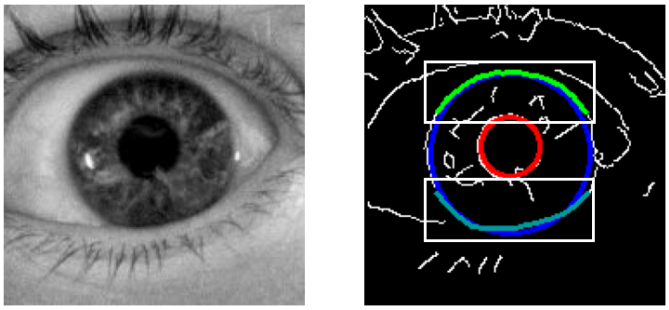

Fig. 3: intermediate results, left: histogram normalized input image, right: Canny edge detection and detected boundaries. Note: this picture is taken without infrared lighting. The white boxes signify the ROI used for detecting eyelids.

2) Eyelids: In order to find the eyelids, a similar strategy is used. Using a similar strategy as [8] A region of interest is defined based on the detected circle, where the eyelid is to be expected. This ROI is applied to the previous Canny image, in which a contour check is then performed. By checking which contour is the longest the eyelid can be found. This process is then repeated for the other eyelid. As can be seen in Fig. 3 the eyelid detection failed for the top eyelid, since the eyelid did not interrupt the curve of the limbic boundary, which therefore was longer.

3) Mask generation: From the detected circles, a binary mask is then generated, which can be used to determine which parts of the Hamming code are valid. This is done by generating a doughnut shape of binary ones between both detected circles. Another mask is used for the eyelids, setting all pixels between both detected eyelids to 1 and outside to 0. A bitwise and operation between both of these masks results in the final mask.

B. Feature extraction

1) Coordinate transform: To normalize the iris to a rectangular shape the polarTransform library was used [9], which cuts the iris shape and transforms it into polar coordinates. The same transform is applied to the mask to make it usable when calculating the similarity score later on. To further normalize all irides to the same size, the output of the transform is resized using OpenCV to a 192 by 48 rectangle. This size is chosen to make it easier to extract exactly 2048 bits. This, together with the polar transformation of the iris ensures that each iris will have the same dimensions when extracting features.

2) Gabor Wabelets: In [1], Daugman describes that many different Gabor wavelets are used to create the iris code. To improve the speed of the algorithm for an embedded device, this research aims to only use a single wavelet. In order to use a Gabor Wabelet as a filter, a kernel should be constructed which

can then be used in python. Previous research has already described code that can be used for this [5], [6]. This code can be found in Section A. It is not yet known which Gabor wavelet is the most accurate, however, this can be found by implementing the algorithm on a few images by iterating over the number of rows and columns needed, center frequencies and orientation and scaling factor and analysing the output. The best filter should have low Hamming distances for same eye pictures and Hamming distances of around 0.5 for all other pictures.

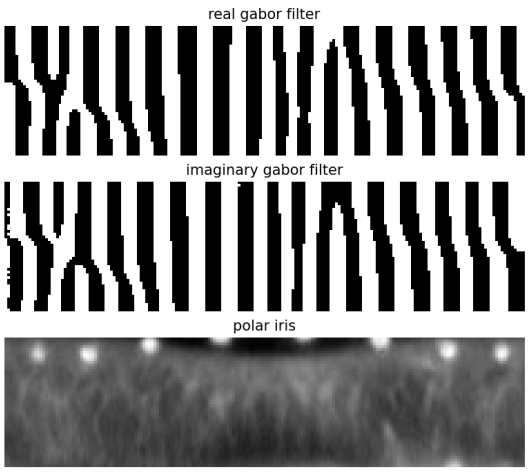

Fig. 4: Example of a filtered iris with arbitrary Gabor wave values.

3) Matching: Like Daugman, 2048 bits are extracted from this output by analysing 1024 regions on their phase information. This is done by reading the bit in each third bit of each row and column, starting from (1,1), which leads to $\frac{192 \cdot 48}{3} = 1024$ regions. The resulting iris code is then 2048 bits long. by varying the starting position or performing a bit shift of 32 bits (1 column), a slight difference in rotation can be countered. As explained earlier, the Hamming Distance is then calculated accordingly. By checking the database for the lowest match, a similar eye can be found. Naturally, this has to fall below a certain threshold. Although it is not the aim of this research, this could then be used as an identification system by linking irides to persons.

PHYSICAL SYSTEM

One of the goals of this paper is to make the system embedded and standalone (no connection to other machines needed). For this reason, it is chosen to use a Raspberry Pi 4 as the platform for the algorithm, since it easy to interface with camera modules, monitors, mouse and keyboard. Making it perfect to demonstrate embedded possibilities.

C. Image capture

1) Camera: The first step is to create an image of the iris. Incorporating the information from the previous section into the prototype design, the choice falls on the Arducam OV9281 NoIR camera module. This camera is affordable, compatible with the Raspberry Pi 4 and more importantly, does not have an IR filter. It is not a Monochrome CCD camera and there is not a visible light filter, but when using NIR LEDs to light up the eye, the camera should still mostly be saturated by the wanted wavelengths, the picture can then be converted to grayscale to get the desired result. The camera has a resolution of 1MP (1280x800) which is enough to be able to ensure a minimum of 10 pixels per millimeter.

2) Focus assessment: To check the focus of the image the image is convoluted with the Laplacian kernel, of which the variance is then calculated. The Laplacian is a kernel that is often used for edge detection. It calculates the second derivative of the pixel it is convoluted by, signifying rapid change. Since a picture that is in-focus should, by definition, include a lot of different sharp transitions in pixel values, taking the variance of this calculation results in a higher number for more in focus images. Inversely, a blurred image is very monotone, does not have sharp transitions and will therefore not have a high variance, as most transitions will be roughly the same [10], [11]. By setting a threshold it can be ensured that a picture will only be used if the calculated variance is high enough.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

3: The Laplacian kernel.

3) Lighting: The LED's used to light up the eye are 850nm NIR LED's, 8 in total, in line with [4]. They are equally spaced with a radius of 25mm to irradiate the eye equally from all sides. Since it is not certain how bright they have to be, they are driven by a TLC5940PWP LED driver. This is a PWM driver which is 12 bit addressable, to make it possible to set the brightness on a later stage. A simple PCB has been designed to fit all the components, as shown in Fig. 5. The LED's and chip can be powered by the 5V output of the Raspberry Pi and connectors are added to which the communication channels can be connected from the Pi. A hole is left in the middle of the PCB for the camera. As denoted on the PCB, a single 390Ω current reference resistor and a 0.1μF decoupling capacitor are added to ensure proper operation. To communicate with the LED driver, an open-source code by Aiden Holmes was used [12].

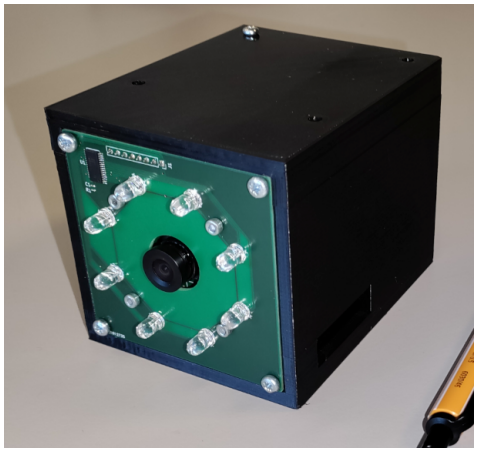

Fig. 6: Embedded iris scanner.

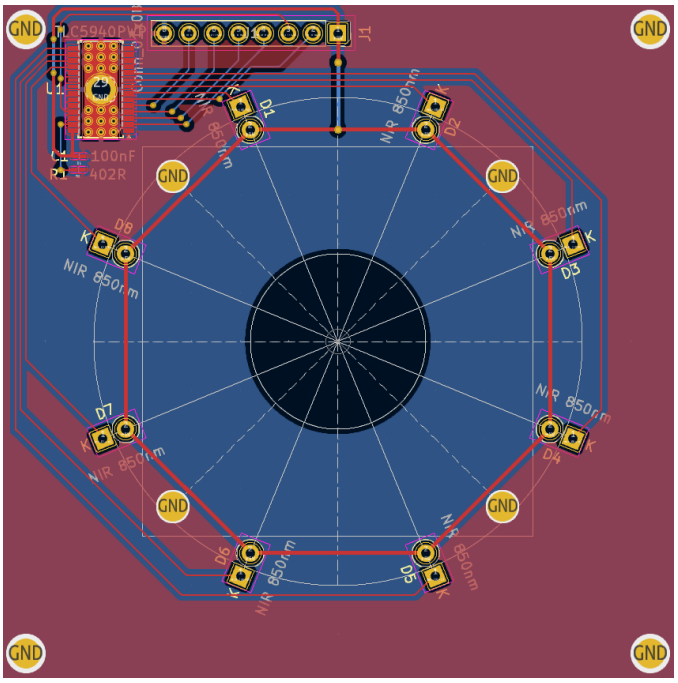

Fig. 5: PCB design for the the brightness control of the LED's.

D. Embedded design

The final prototype is supported by an enclosure for the PCB and Raspberry Pi and can be seen in Fig. 6

IV. Results and Discussion

To analyse the performance of the algorithm, pictures of the CASIA V1 iris dataset are used. Eliminating any possible errors in the pictures that the physical model might have. These pictures have all been manually checked and are in focus.

A. Iris & eyelid detection

The results of the boundary detection algorithm can be seen in Figs. 3 and 7.
Here it becomes clear that the algorithm can properly detect the limbic and pupillary boundary using the Hough Circle

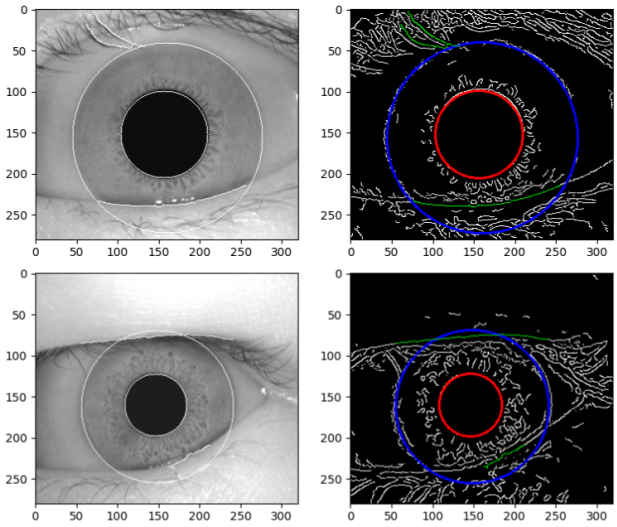

Fig. 7: Boundary detection algorithm on database images.

Transform. Some misalignments can still be noticed, since the iris is not actually a perfect circle, but can have a slight oval structure. However, the algorithm struggles a bit more with the boundaries of the eye-lids. As can be noticed from Fig. 7, the contour of the eyelid is not always properly visible, since it is obstructed by eyelashes, causing it to break. The canny edge detection struggles to notice this, causing the contour to be improperly detected. A way around this could be to implement a form of shape fitting. Since shape of the eyelid is always parabolic, trying to fit parabolas with specific values on the ROI should improve this, as the rough shape of the parabola is known beforehand.
The algorithm did however not always function properly, as shown in figure Fig. 8. sometimes the Canny edge detection fails to highlight the right edges, due to high-intensity changes of other parts of the image, this causes the algorithm to misinterpret the image and draw circles on the wrong parts. In the case of the right image in Fig. 8, the iris has a circular ring of muscle which is clearly visible. Since the algorithm is trying to locate circles from large to smaller, it detected this circle before detecting the iris. In general, the iris is detected correctly in 60% of the images. A possible solution for these issues is to increase the Gaussian blur and tuning the canny edge detection differently in order to put more emphasis on the detection of larger edges over local edges, while smoothing out the muscle edges in the picture. This might also improve the eyelid detection, since the eyelashes will have less effect on the edge detection. Since iris detection requires the cooperation of a person in order to take a proper picture of the eye, it could also be asked to open their eyes further, limiting the obstruction caused by eyelids and improving the circle detection.

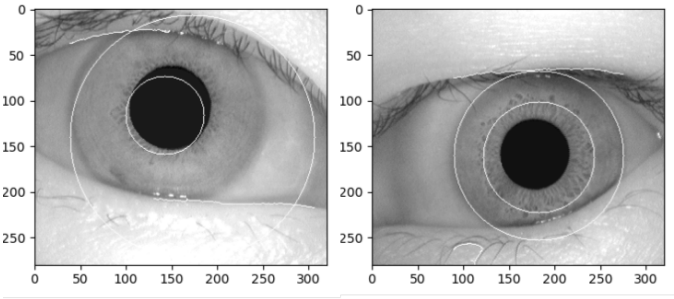

Fig. 8: Incorrect boundary detection on database images. Left: incorrect edge detection causing different circles to have higher scores. Right: A circular pattern in the features of the iris causing the Hough circle detection to classify the wrong circle.

Fig. 9 shows the normalized image and mask that are created from an extruded iris. The mask is white for bits that count and black for bits that do not count towards the final calculated hamming distance. It shows that the lower eyelid is correctly masked, the black regions in the mask corresponding to the warped eyelid in the polar transformed image. Part of the iris which is still visible is also masked out.

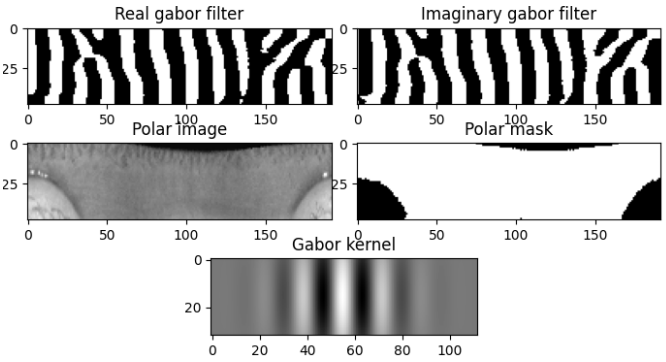

Fig. 9: Normalized iris and iris mask of the upper iris in Fig. 7, the filtered output and the used Gabor kernel.

Due to the edge detection not being fully reliable, for all other steps in the algorithm, each picture was manually checked to ensure proper results. Pictures in which the iris was not properly detected are rejected and not used for determining Gabor wavelet parameters.

B. Matching

To find the best Gabor wavelet, 10 images were used, 4 of the same iris, 2 pairs of different irides and 2 single irides. The hamming distances between these irides were calculated for 780 different filters by looping over the following parameters of the filter. A part of the results can be seen in Fig. 10.

- number of rows
- number of columns
- center frequency
- orientation
- wavelet scale

In Fig. 10, a clear difference can be seen between same iris pictures (iris 7, 8 and 9) and different irides when using the 162th filter, corresponding to R = 32. C = 112, f = $3\sqrt{(2)}$, u = 4 and v=1 as parameters for the filter as described in Section A. This Gabor wavelet is visualised in Fig. 9 This gap can be used to set a decision threshold.

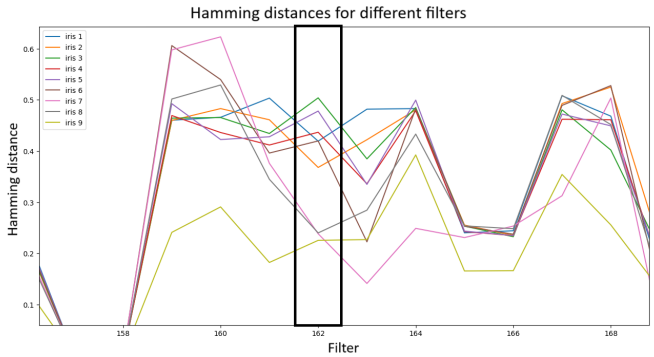

Fig. 10: Hamming distances versus different Gabor wavelets. Note that each Gabor wavelet has different parameters and that they are uncorrelated. The black box encircles the chosen Gabor wavelet ”162”.

This was repeated for both other pairs of eyes, yielding similar results. For all 3 groups of pictures, this filter appeared to be the most suited. This filter was then applied to the algorithm. The filter was then verified by using 55 irides for the CASIA v1 iris database, creating a decision environment as in Fig. 11.

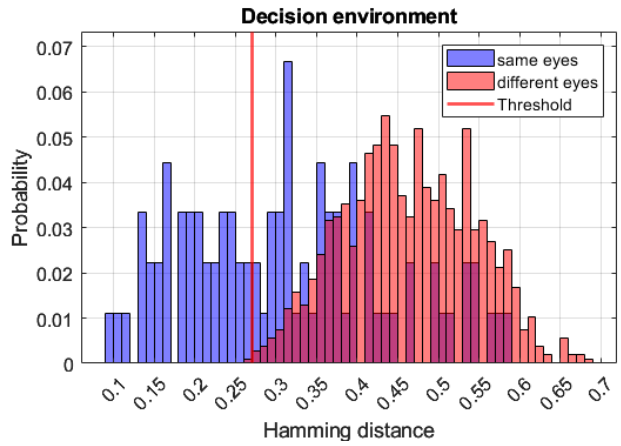

Fig. 11: Hamming distance probabilities.

In Fig. 11 there is a clear bell curve visible for intra-iris comparisons, which can be described with a mean of 0.463 and a variance of 0.0064. The mean being in line with the expectation due to the rotations performed on the iris. However, the curve for inter-iris comparisons is less visible, it’s mean and variance being 0.31 and 0.124 respectively. This spread is most likely due to differences in the pre-processing of the image. If the image is not normalized properly, or an eyelash or eyelid is not masked properly, this will hurt the Hamming distance between the irides. By defining the decision threshold at 0.27 there is only a 0.8% chance for a false positive assuming the mean and standard deviation of the intra-iris matching. While 41% of the same eye pictures is still classified correctly. The 59% false negative rate is large, but

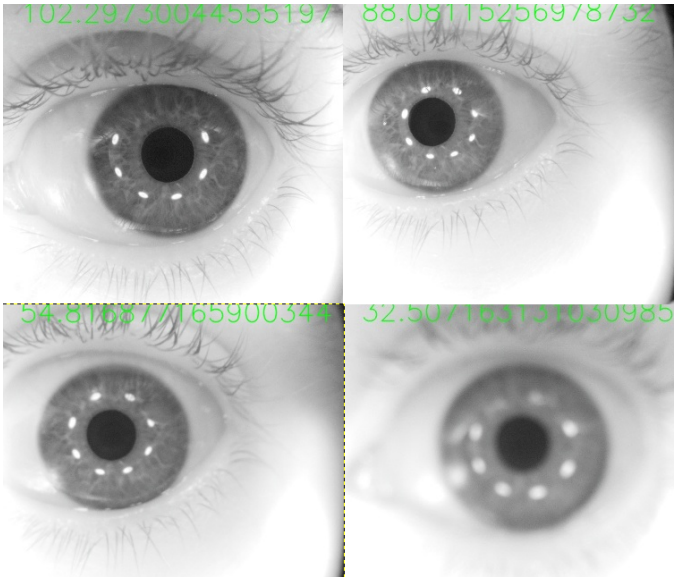

Fig. 12: 4 different states of focus, with calculated variances: 102, 88, 55 and 32.

not problematic, since it simply tells that the algorithm should try again using a different picture. From this, it is possible to conclude that the Filter and Hamming distance calculations are functioning properly and allow for proper iris classification using a threshold of 0.27 or lower, depending on the security requirements. The large false negative rate can be explained due to the inaccuracies that exist in the normalization of the iris.

C. focus detection

Focus detection using the Laplacian variance algorithm proved to be quite effective, as shown in Fig. 12. It shows that pictures with a variance of more than 90 are in focus enough to show the details of the iris. It did seem to be influenced a lot by the light level of the LED's, but after fixing the PWM output of the driver, this was not an issue anymore.

D. lighting

As said before, the reason for using LED lights is to avoid melanin in the iris, to make phase information of the iris more visible. This is effect is illustrated in Fig. 13, which shows the iris of an eye which is dark brown in the visible light spectrum. However, a clear issue with the current setup is the location of the LED's. As visible in multiple pictures throughout the report,for example figs. 12 and 13, the Purkinje spots are not inside the pupil but are visible within the iris. This is the result of the radius of the LED placements being too large for the distance that the eye is from the camera. Moving the eye further away and using a lens for focusing or reducing the radius of the LED ring would resolve this issue, sadly due to time constraints, this could not be realized anymore. Another issue with the lighting was that the direct light from the LED's would create distortion in the picture. This was solved by adding 2 layers of light diffusing plastic over the

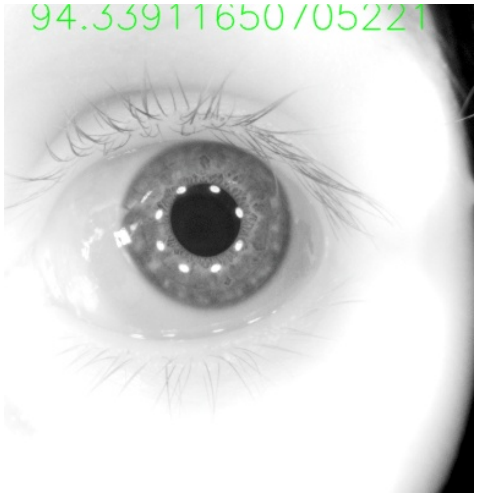

Fig. 13: image of a dark brown eye with visible features.

LED's. After this, the optimal PWM signal of the LED's was found to be at 14/4086. At this value, the iris features are properly visible. However, adding more light would saturate the image, causing a loss of focus.

V. Combined results

The final product was tested by creating 20 images, using 3 different irides. Due to differences in the lighting conditions with respect to the database images, the thresholds for the canny edge detection were altered slightly. The results can be seen in Fig. 14 and in Fig. 15. In these pictures, 1 was unusable, 4 had improper limbic boundary detection and 10 pictures had (slightly) wrong pupil detections.

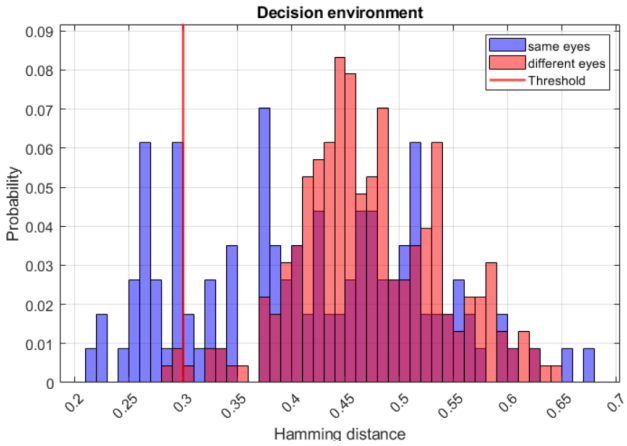

Fig. 14: Hamming distances using the embedded system, without filtering for improper edge detection.

In these figures it becomes clear that when the edge detection is working properly, classification using a threshold is perfectly possible. However, due to image imperfections, there still is a 60% false negative rate, which is relatively high. Part of this is due to the iris being rotated further than the algorithm checks for, this is due to the embedded system being small and not gyro stabilized. When taking a picture, the angle relative to the eye can differ quickly.

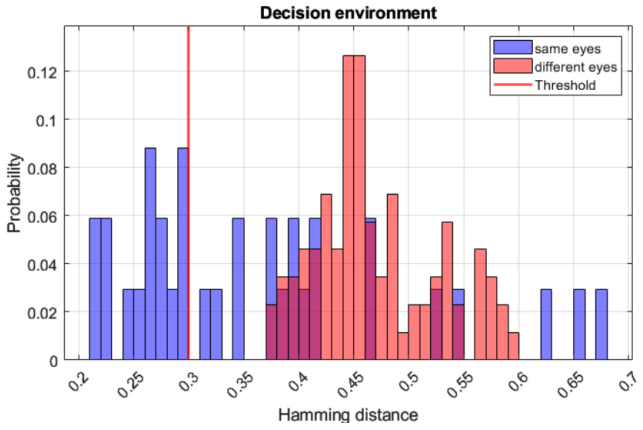

Fig. 15: Hamming distances using the embedded system, only using properly edge detected images.

Fig. 16 shows the influence that the Purkinje spots have on the boundary detection algorithm, In the bottom eye, they are detected as a circle and hence classified as the pupillary boundary. Fig. 17 shows the impact this has after convolution with the Gabor wavelet. The structures of the lines is completely warped, which explains the false negatives. Interestingly, the Purkinje spots do not seem to affect the shape of the output.

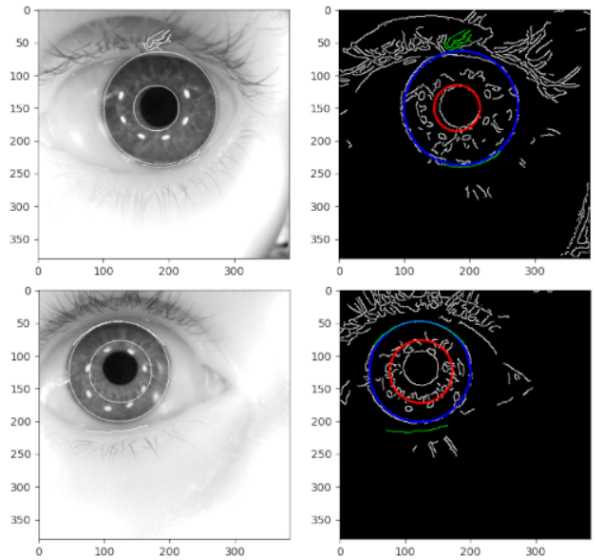

Fig. 16: captured images using embedded system. Top: correct detection, bottom: wrong detection.

VI. Future work

As discussed throughout this report, the main challenge of this algorithm lies not in the feature extraction, but in the pre-processing. To improve this is to improve the entire system. There are multiple strategies which can improve the edge detection. For one, implementing Daugmans' integro-differential operator [1], which can be used for both eyelid detection and circle detection. Furthermore, machine learning could be applied to recognise the structure of the eye. The current system also expects the iris to be a perfect doughnut

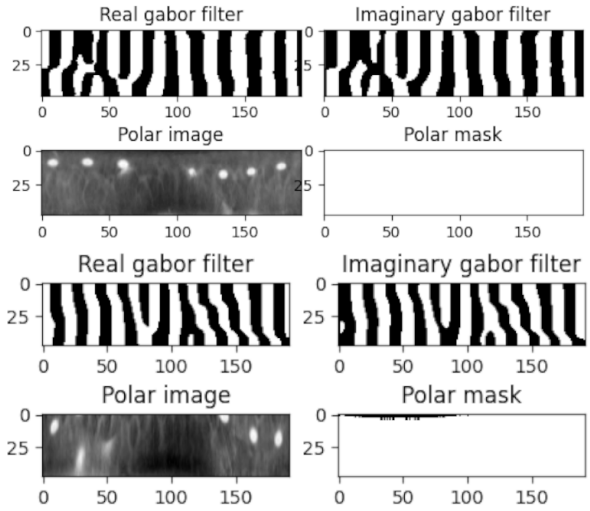

Fig. 17: Filter outputs corresponding to the irides in Fig. 16.

shape, while in reality the iris does not necessarily have to lie exactly in the center of the iris. To compensate for this would further improve the algorithm. The effects of using multiple Gabor wavelets could also be studied, improving the accuracy of the algorithm even more. Creating some form of holder for the system in order to keep it horizontal would also improve the results. Additionally, adding a headrest would ensure the distance from the camera to the eye does not differ greatly per picture.

VII. Conclusion

Iris recognition is proven to be one of the most reliable forms of biometric identification. The aim of this research was to create an embedded iris recognition system. This was done by creating an algorithm on a Raspberry Pi 4 that follows the key steps of iris recognition: image creation, focus assessment, iris detection, normalization, feature extraction and matching. The final result was a functioning algorithm, with which irides could successfully be classified. However, inconsistencies in boundary detection and normalization of the images led to a high false negative rate, which was partially caused by the reflections of the near infrared LED's in the final product. To improve the algorithm, more focus should be put on the pre-processing of the image, as a lot of issues are originating from it. In the current algorithm, the edge detection of the iris is only correct in about 60% of the cases which already hinders the algorithm. Furthermore, the iris is normalized assuming a doughnut shape with perfectly centered inner circle, which is not always possible since the pupil is not necessarily in the middle of the iris, which causes the iris to be incorrectly transformed. Lastly, eyelid detection still struggles with eyelashes, causing incorrect mask generation. Overall the algorithm proved to be secure and usable as a verification system, yet more research needs to be done in order to be able to be able to use it as a means of identification.

REFERENCES

[1] J. Daugman, "How iris recognition works," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 21–30, 2004.

[2] J. Matey, G. W. Quinn, and P. J. Grother, "Forensic iris: A review, 2022," 2022-07-18 04:07:00 2022.

[3] H. Gu, Y. Zhuang, Y. Pan, and B. Chen, "A new iris recognition approach for embedded system," in *Embedded Software and Systems* (Z. Wu, C. Chen, M. Guo, and J. Bu, eds.), (Berlin, Heidelberg), pp. 103–109, Springer Berlin Heidelberg, 2005.

[4] Y. Si, J. Mei, H. R. Karimi, C. Wang, and H. Gao, "Design and implementation of a low-cost embedded iris recognition system on a dual-core processor platform," *IFAC Proceedings Volumes*, vol. 45, no. 4, pp. 278–282, 2012. 1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control.

[5] Y. Chen, T. Jeanneau, and C. Brendel, "Iris feature extraction with 2d gabor wavelets," Apr 2023.

[6] H. Ghodrati, M. J. Dehghani, and H. Danyali, "Iris feature extraction using optimized gabor wavelet based on multi objective genetic algorithm," in *2011 International Symposium on Innovations in Intelligent Systems and Applications*, pp. 159–163, 2011.

[7] C. Houston *Iris Segmentation and Recognition Using Circular Hough Transform and Wavelet Features*, 2010.

[8] M. Adam, F. Rossant, F. Amiel, B. Mikovicova, and T. Ea, "Reliable eyelid localization for iris recognition," in *Advanced Concepts for Intelligent Vision Systems* (J. Blanc-Talon, S. Bourennane, W. Philips, D. Popescu, and P. Scheunders, eds.), (Berlin, Heidelberg), pp. 1062–1070, Springer Berlin Heidelberg, 2008.

[9] A. Elliott, "Polartransform," 2018.

[10] J. Pech-Pacheco, G. Cristobal, J. Chamorro-Martinez, and J. Fernandez-Valdivia, "Diatom autofocusing in brightfield microscopy: a comparative study," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, vol. 3, pp. 314–317 vol.3, 2000.

[11] A. Rosebrock, "Blur detection with opencv," Sep 2015.

[12] A. Holmes, "Tlc5940: Python driver for tcl5940 16 channel pwm," Jul 2017.

[13] Z. Chai, "2d gabor wavelets," Jul 2008.

[14] S. B. Hathwar, "How to "apply" 2d gabor wavelet to an image," Feb 2018.

APPENDIX A
GABOR WAVELET IN MATLAB AND PYTHON

The function used to describe Gabor wavelets is shown in both Matlab and Python code. The inputs are:

- R: the number of rows the Gabor wavelet kernel will have.
- C: the number of columns the Gabor wavelet kernel will have.
- Kmax: the maximum frequency of the Gabor wavelet.
- f: a frequency scaling factor, used to control the center frequency of the wavelet.
- u: the orientation of the Wave vector. Usually a number between 0 and 7 allowing for 8 different rotations of the wavelet.
- v: a scaling vector for the Wave vector.
- Delt2: The variance of the Gaussian envelope. Influences the bandwidth of the filter and how quickly the influence of a pixel falls off at a distance.

The function returns a kernel which can be used as a filter by convolution. In Python, OpenCV can be used as: cv2.filter2D(image, output type, filter kernel)
The following Matlab function is copied from [13]

```
function GW = GaborWavelet (R, C, Kmax, f, u, v, Delt2);
% Create the Gabor Wavelet Filter
% Author : Chai Zhi
% e-mail : zh_chai@yahoo.cn
k = ( Kmax / ( f ^ v ) ) * exp( i * u * pi / 8 );% Wave Vector
kn2 = ( abs( k ) ) ^ 2;
GW = zeros ( R , C );
for m = -R/2 + 1 : R/2

    for n = -C/2 + 1 : C/2

        GW(m+R/2,n+C/2) = ( kn2 / Delt2 ) * exp( -0.5 * kn2 * ( m ^ 2 + n ^ 2 ) / Delt2)
            * ( exp( i * ( real( k ) * m + imag ( k ) * n ) ) - exp ( -0.5 * Delt2 ) );

    end
end
```

and in python, copied from [14]:

```
def gabor_wavelet(rows, cols, kmax, f, orientation, scale,
                                                    delt2):

    k = (kmax / (f ** scale)) * np.exp(1j * orientation * np.pi / 8)
    kn2 = np.abs(k) ** 2

    gw = np.zeros((rows, cols), np.complex128)

    for m in range(int(-rows/2) + 1, int(rows / 2) + 1):
        for n in range(int(-cols/2) + 1, int(cols / 2) + 1):
            t1 = np.exp(-0.5 * kn2 * (m**2 + n**2) / delt2)
            t2 = np.exp(1j * (np.real(k) * m + np.imag(k) * n))
            t3 = np.exp(-0.5 * delt2)
            gw[int(m + rows/2 - 1),int(n + cols/2 - 1)] = (kn2 / delt2) * t1 * (t2 - t3)

    return gw
```

APPENDIX B
AI STATEMENT

During the preparation of this work the author used Perplexity and Grammarly in order to review and check the document on errors. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work. During the preparation of this work the author used ChatGPT 3.5 in order to review and check the python code for debugging purposes. After using this tool/service, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.