Optimisation of pupil detection algorithm

Ian Franson Mendonca(1653742)

Supervised by : Prof.Dr.Kristof Van Laerhoven

Abstract

This research presents a fresh approach for detecting pupils using an embedded system to boost processing speed. The approach is compared against an established model, both operating within the same system. The new method is evaluated using an open-source pupil dataset and results are found to be satisfactory and can be used in the real-time detection of pupils. The limitations concerning the new method and future scope for the same are discussed in the paper.

1 Introduction

The collection and analysis of biological data to obtain specific information have diverse applications in the real world [4]. Studying the human eye can provide information about the health and emotional state of the user [3]. Real-time eye tracking is an essential topic in the healthcare sector, as well as in artificial intelligence, augmented reality (AR), and virtual reality (VR) [1,3]. There are several devices available in the market, ranging from head-mounted setups to mobile phones, that utilize open-source software to track the eye [2]. To study these behaviors, it is important to capture the movements of the eye in real time with high resolution and speed (frames/second). This can be achieved by tracking the darkest region in the eye, known as the "pupil" [2]. This paper is based on existing research [4] which takes advantage of edge detection to track the movement of the eye. In this paper, an image binarization [2] technique is used to differentiate the darkest region from the rest of the image, enabling precise analysis.

The setup used for pupil detection is an open-source miniature and low-cost hardware design that consists of an embedded system with widely used camera modules and the Rasberry Pi platform [4]. The miniature and low-cost design of the setup helps researchers to make progress with reduced efforts.

The rest of the paper talks about the method (2.1) used to track the pupil location and the reason for using binarization over canny edge detection (2.2). The section (2) gives more insight into the methodology and explains the code in brief along with the prerequisites required to execute the code. In section (3), the model is evaluated on the existing state-of-the-art publicly available dataset for detection and performance of the model for different conditions. The section 4 explains the limitations associated with the model explained in the paper. The paper concluded with a conclusion and future outlook.

2 Methodology

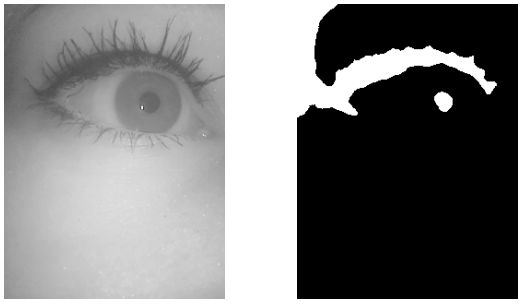
2.1 Image binarization

Near-eye infrared cameras produce images that separate the pupil from the rest of the eye. This will speed up the process of finding the eye position, pupil dilation, or pupil diameter [3]. Image binarisation divides the eye image into two parts, black(pupil) and white(rest of the eye) and this is possible by finding the threshold value of the pixel which divides the two regions. Widely used techniques such as the Otsu method, and adaptive thresholding can not be used for pupil segmentation [2] since the surrounding environment plays an important role in deciding the thresholding pixel value. Therefore binarisation is performed based on the pixel value obtained from the histogram of the image.

"_,thresholded=cv2.threshold (grey image, (1), (2), cv2.THRESH_BINARY_INV) "

The provided code demonstrates how to create a binary image from a grayscale image. By using the **cv2.threshold** function with the **cv2.THRESH_BINARY_INV**, the pixels in the grayscale image are separated into two categories. Pixels with values below a specified threshold are set to white (intensity 255), while pixels with values above the threshold are set to black (intensity 0). This process effectively creates a binary image where the regions of interest are represented by white pixels, and the background is represented by black pixels. The threshold value is inserted where '1' is specified, and the maximum value of the pixel intensities in the image is added where '2'

is mentioned in the code.



(a) (b)

Figure 1: **shows the binarised image 1b of the grey scaled image 1a. As can be seen in figure 1b dark regions in 1a are converted to white and the rest to black color.**

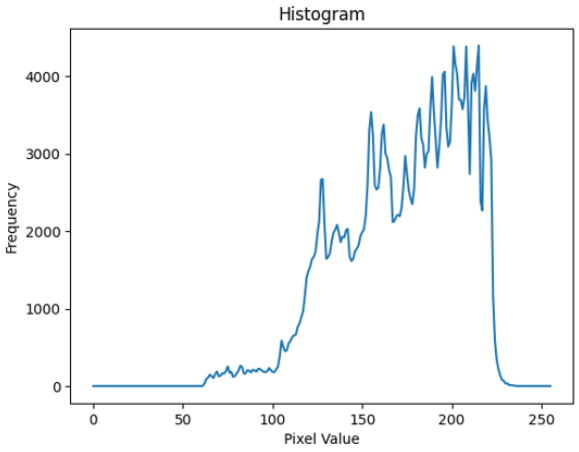


Figure 2: **Pixel density in a 640x480 grey image**

To address the challenge of determining the appropriate threshold pixel value by visually inspecting the image, a helpful approach is to plot the histogram of the grayscale image. This histogram provides the distribution of pixel intensities and by analyzing the histogram, pixel intensity that closely corresponds to the pupil can be located. This helps in estimating an optimal threshold value, which can then be used in conjunction with the cv2.threshold function to perform thresholding and obtain a binary representation of the image.

Figure 2, shows the pixel density in an image of an eye under a grey scale. Typically, the initial peak in the depicted graph corresponds to a potential threshold value. However, it's important to note that this assumption might not always hold true. Hence, determining the threshold value manually becomes necessary, involving an iterative trial and error approach. The idea here draws inspiration from how humans identify the pupil within a provided image of an eye.

Dealing with images that have sharp edges can be tricky when trying to convert them into binary images due to the sudden shifts in pixel intensity. This abrupt transition can complicate the process of choosing appropriate threshold values for binarization, leading to undesirable outcomes in the binary result. In order to make the model less susceptible to noise and sudden changes in the pixel intensities, a Gaussian Filter is used. This filter serves the purpose of smoothening the image and reducing noise in the image.

The subsequent code explains the steps required to apply the Gaussian Filter to remove the noise and sharp edges from the image.

"blurred = cv2.GaussianBlur(image, (kernel size,kernel size), X_sigma, Y_sigma)) "

The OpenCV library is utilized to access the Gaussian Filter function. This function requires three inputs: the image for denoising, the kernel size determining the blur level (larger values mean stronger blur), and the standard deviations along the X and Y axes.

To discover the best Gaussian Kernel value, parameter adjustment was performed on the LPW/4 dataset. For this purpose, odd numerical values were selected as Gaussian Kernel sizes, ranging from 1 to 9. The accuracy for each kernel size was computed using the L2 norm and the same was plotted as below (Figure 3) for different pixel errors(1px, 5px, 10px, 15px, 20px). It is important to note that larger kernel sizes demand more time for filtering the image.

2.2 Thresholding Vs Canny Edge Detection Time

The canny edge detection algorithm is used in the paper [3] takes more time to process a single frame when compared to thresholding. This is shown in figure 4.

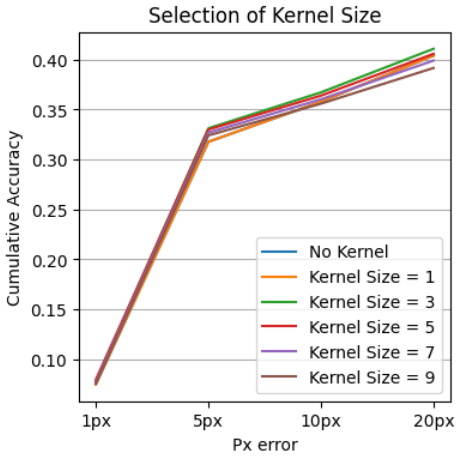


Figure 3: **Parameter selection for Gaussian filter Kernel size based on LPW/4 dataset.**

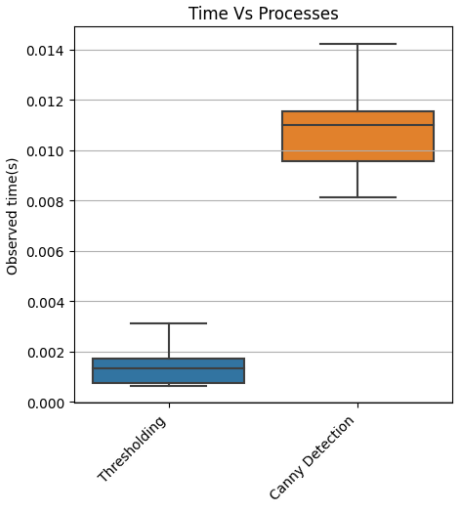


Figure 4: **Comparision of Binarisation and Canny edge detection time**

Figure 4 shows the processing time of both algorithms per frame. It shows that the time taken by thresholding is about 10 times less than that of canny edge detection. (The data required for the comparison is collected from the setup mentioned in [5])

3 Evaluation

3.1 Accuracy Evaluation

In order the evaluate the effectiveness of the new algorithm in real-time, the model is tested using the LPW dataset [5]. The dataset contains a wide variety of images, with different conditions such as outdoor/indoor/artificial lighting, participants with prescribed glasses or contact lenses, and different eye colors. As an example, participant LPW/3 is wearing prescribed glasses in outdoor conditions and there are a total of 22 participants in the dataset, and it contains about 130,856 images with 640x480 resolution [5].

To evaluate the performance of the model, Euclidean distance or L2 norm is used [3]. Using the L2 norm, the error between the center of the pupil detected using the model and the ground truth provided in the dataset for all the participants.

As shown in the above table 1, the proposed model performed fairly well when compared to Pupil Labs, ExCuse, and Swirski algorithms which stood at about 30% detection rate with less than 5px error on LPW dataset [5]. For better results, it is always better to optimize the parameters such as pixel threshold value and Gaussian Kernel size according to the dataset/condition.

Table 1: **Accuracy of the model with Kernel Size = 3 on LPW Dataset**

Sl.No	Px Error	Accuracy%
1	1Px	21.64
2	5Px	51.44
3	10Px	57.26
4	15Px	59.16
6	20Px	60.16

The model's performance was poor under certain conditions, such as when participants wore glasses, likely due to light reflection, as well as in cases where the camera position was bad or when there was excessive outdoor lighting.

Figure 7 illustrates scenarios where the model failed to accurately predict the pupil location. For instance, Figure 7b depicts a case of poor camera positioning, while Figures(7a,7c7d) demonstrate examples of challenges due to external light reflection and difficulties posed by eyeglasses. [5].

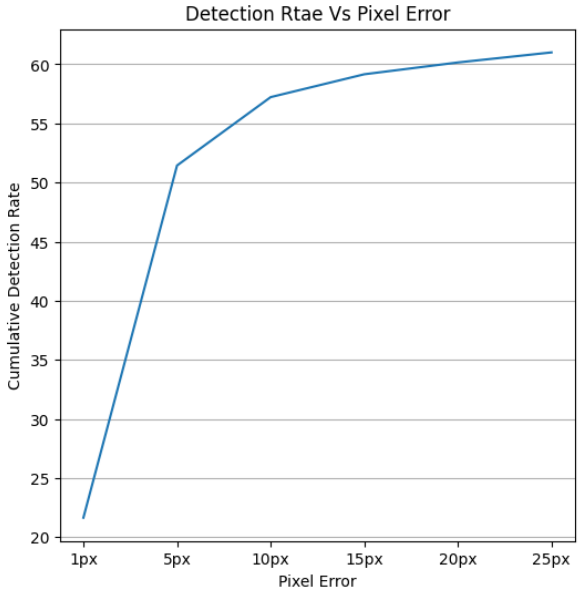
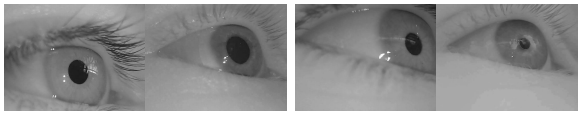


Figure 5: **Detection rate of the model on LPW dataset [5] for different pixel error**



(a) (b) (c) (d)

Figure 6: **Images from LPW dataset where the model was able to successfully detect the pupil**

3.2 Model Runtime Evaluation

The main goal of this paper is to reduce the overall processing time of the processor per frame and this section sheds some light into the model's runtime. To evaluate the processing time 500 frames were captured using the setup with a resolution of 480x640px & 240x320px and the time required for every process is calculated.

The runtime model consists of pre-processing, finding contours, filtering contours, post-processing, and Total time for a single frame. Pre-processing includes the time required to convert RGB images to greyscale along with blurring and binarization. The below figures (8,9) show the comparison and time needed to perform different operations in the model. Initially, each frame is

(a) (b) (c) (d)

Figure 7: **Scenarios where the model was not able to successfully detect the pupil due to external factors**

cropped for the region of interest(ROI) before the processing of the same. The time taken to process a single frame, as indicated in [3] exceeds the duration achieved in this study by 40%. This reduction benefits selecting a higher resolution camera with faster fps, subsequently enhancing the tracking speed. The provided figure illustrates that the majority of time during the entire operation is consumed by preprocessing (blurring).

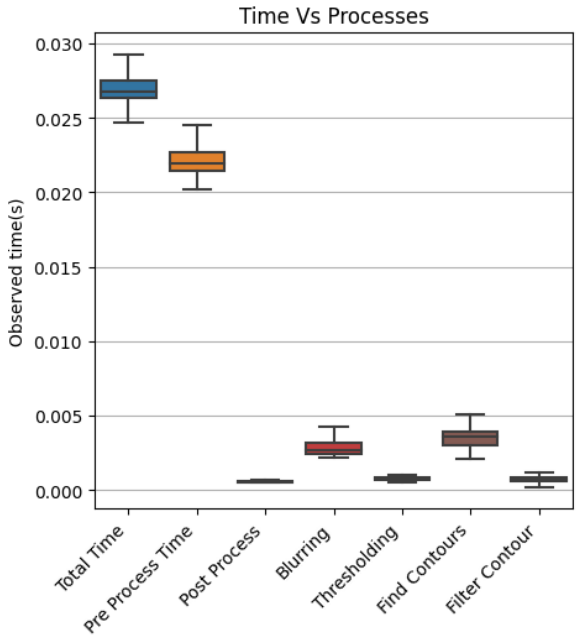


Figure 8: **Time required for pupil detection for a resolution of 480x640px along with time for each process present in the pipeline.**

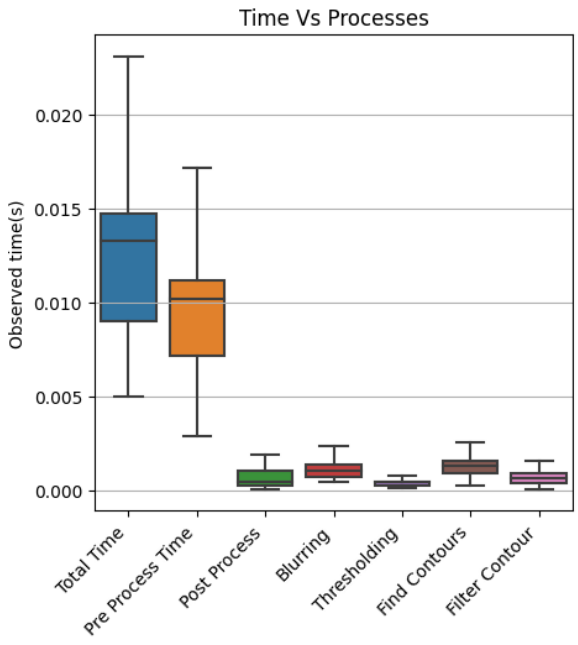


Figure 9: **Time required for pupil detection for a resolution of 240x320px along with time for each process present in the pipeline.**

4 Limitations

The discussed algorithm in this paper is highly dependent on the surrounding environment since the pixel value required for thresholding is inserted manually. The accuracy of the algorithm drastically changes when the surrounding environment changes as discussed in the section 3.

5 Conclusion

This paper introduces a novel technique that holds promise for advancing pupil detection in the future and the following points were concluded.

- Reduced processing time of the model.
- Reason for choosing binarization(thresholding) over canny edge detection.
- The effectiveness of the model using the LPW dataset.
- Limitations affecting the accuracy of the model.

Addressing the limitations could lead to a more robust and improved system for eye tracking.

References

[1] Ke-ke Gu. Real-time pupil detection based on contour tracking: Selected papers from csma2016. 2017.

[2] Moritz Kassner, William Patera, and Andreas Bulling. Pupil: An open source platform for pervasive eye tracking and mobile gaze-based interaction. 2014.

[3] Ankur Raj, Diwas Bhattarai, and Kristof Van Laerhoven. An embedded and real-time pupil detection pipeline. 2023.

[4] Filip Rynkiewicz, Marcin Daszuta, and Piotr Napieralski. Pupil detection methods for eye tracking. 2018.

[5] Zhang X Sugano Y Bulling A Tonsen, M. Labeled pupils in the wild a dataset for studying pupil detection in unconstrained environments. 2016.