Eye Tracking on Unmodified Mobile VR Headsets Using the Selfie Camera

Panagiotis Drakopoulos

A thesis presented for the degree of

Master of Engineering

Dept. Of Electrical and Computer Engineering

Technical University of Crete

Greece

May 2021

Declaration of Authorship

I, Panagiotis Drakopoulos, declare that this thesis titled, "Front Camera Eye Tracking for Mobile VR" and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

Jury

3-member Committee

Assoc. Prof. Katerina Mania	Technical University Of Crete
Assoc. Prof. Antonios Deligiannakis	Technical University Of Crete
Prof. Konstantinos Mpalas	Technical University Of Crete

Abstract

Input methods for interaction in smartphone-based virtual and mixed reality (VR/MR) are currently limited on uncomfortable head orientation tracking controlling a pointer on the screen. User fixations are a fast and natural input method for VR/MR interaction. Previously, eye tracking in mobile VR suffered from low accuracy, long processing time and the need for hardware add-ons such as anti-reflective lens coating and infrared emitters. We present an innovative mobile VR eye tracking methodology utilizing only the eye images from the front-facing (selfie) camera through the headset's lens, without any modifications. Our system first enhances the low-contrast, poorly lit eye images by applying a pipeline of customised low level image enhancements suppressing obtrusive lens reflections. We then propose an iris region-of-interest detection algorithm that is run only once. This increases the iris tracking speed by significantly reducing the iris search space in mobile devices. We iteratively fit a customised geometric model to the iris to refine its coordinates. We display a thin bezel of light at the top edge of the screen for constant illumination. A confidence metric calculates the probability of successful iris detection, based on knowledge from previous work and experimentally validated heuristics. Calibration and linear gaze mapping between the estimated iris centroid and physical pixels on the screen results in low latency, real-time iris tracking. A formal study confirmed that our system's accuracy is similar to eye trackers in commercial VR headsets in the central part of the headset's field-of-view in optimal illumination conditions. In a VR game, gaze-driven user completion time was as fast as with head tracked interaction, without the need for consecutive head motions. In a VR panorama viewer, users could successfully switch between panoramas using gaze.

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Katerina Mania for her continuous motivation, patience and friendly encouragement throughout the timespan of this work, that played a fundamental role in overcoming this project's inherent challenges and difficulties. She shaped the way I approach research while helping me develop valuable life traits such as patience, mental endurance, positive mindset and collaboration skills. Furthermore, her kind-hearted human side greatly contributed to the success of this project at times of personal issues and struggles, which will be remembered. I feel lucky that we worked together. I will remember her not only as a professor and advisor, but also as a friend and life coach.

Secondly, I would like to thank Dr. George Alex Koulieris for the large amount of time, effort, brainstorming and energy he invested. His expertise and previous experience on the rapidly evolving field of Virtual Reality and Eye-Tracking provided vital assistance to decision making and problem solving. Without his guidance, this work may not have been successful. His contribution will not be forgotten.

Thirdly, I would like to deeply thank my family for their limitless and selfless support through all these years. Without them I would have not reached my goals. I would also like to thank my friends and lab colleagues. Each and everyone of them contributed in their own way to the fulfilment of my goals.

Finally, I would like to express my sincere gratitude to my professors and all university staff in general for their hard work and sharing valuable knowledge and expertise. I am proud to have been a part of Technical University Of Crete.

Publications

- **Eye Tracking Interaction on Unmodified Mobile VR Headsets Using the Selfie Camera**.

  ACM Transactions on Applied Perception, Volume 18, Issue 3, May 2021.

  Authors: Panagiotis Drakopoulos, George-Alex Koulieris, Katerina Mania

  URL: https://dl.acm.org/doi/10.1145/3456875.

- **Front Camera Eye Tracking For Mobile VR**.

  2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW).

  Authors: Panagiotis Drakopoulos, George-Alex Koulieris, Katerina Mania

  URL: https://ieeexplore.ieee.org/abstract/document/9090461.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Context

Virtual, Augmented & Mixed Reality. *VR* is the most widely known of these technologies, and is often used as an umbrella term to describe other technologies similar to, but different from, an actual Virtual Reality experience. In VR, users are fully immersed in a computer-generated world of imagery and sounds, using a VR Head Mounted Display (HMD), which completely hides the user's surroundings. *AR* overlays digital information on real-world elements, supplementing a user's reality or environment with digital details, while keeping the real world visible and in the center of the experience. *MR* aims to shrink the gap between between real and digital elements. In MR, users interact with and manipulate both physical and virtual items and environments, using next-generation sensing and imaging technologies. The aforementioned technologies have a wide gamut of applications, including, but not limited to healthcare, research, gaming and entertainment. Virtual Reality technologies are enabled by wearing purpose-specific headsets, each coming with its own set of abilities and limitations, depending on the use case they are designed for.

(a) (b) (c)

Figure 1: (a) Oculus Rift VR headset (b) MagicLeap AR headset (c) Microsoft Hololens 2 MR headset.

Mobile VR refers to the subset of VR technologies that utilize a smartphone as a VR headset to display immersive computed-generated imagery, while being placed in a compatible HMD. Smartphone hardware capabilities have improved dramatically the past decade, enabling applications that were previously available only on high-end desktop computers. Current-generation smartphones come with high performance CPUs and GPUs, as well as embedded **IMUs** (inertial measurement unit), such as accelerometers and gyroscopes. Consequently, smartphone are nowadays able to ren-

der high resolution virtual environments, while also responding to a user's movement, using the input from the smartphone's sensors. To achieve the perception of depth, the smartphone's screen renders stereoscopic images by splitting the screen in half-dedicating each half to the corresponding eye.

Key VR Terminology. The level of immersion and thus the overall user experience is highly dependent on a specific set of hardware characteristics, common to all VR systems. The most noteworthy terms that are also frequently referred to in this thesis are:

- *Frames per second (FPS)*: Frequency at which a system can display consecutive images, called frames. A high frame rate (greater than 60fps) is critical for a pleasant VR experience. Without a high and constant frame rate, motion will look jittery and users could up feeling sick.
- *Field of view*: The angle of the observable virtual world. If the window of view is too narrow, immersion will be hindered and users could make unnatural head rotations.
- *Degrees of Freedom (DoF)*: The number of directions that an object can move or rotate. For example, six degrees of freedom are pitch, roll, yaw, left and right, forward and backward, up and down. More DoFs allow more natural movement in VR.
- *Latency*: The amount of time it takes a system to react/respond to movements or commands. Latency is critical when it comes to the presence inside a virtual world—if the system doesn't respond instantly, realism and immersion is significantly degraded.

User Interaction in VR/MR. User interaction is a key element to the immersion and realism of a VR/MR experience. To this end, a variety of input methods and purpose-specific hardware have been devised academically and commercially, each with their own strengths and weaknesses. Several of these methods are referred to as *Tracking* methods in bibliography, due to the fact that in order to achieve real-time interaction, a system has to keep track of a user's eyes, hands, head movement and/or rotations and react to changes. For example, head-tracking tracks the user's head position/rotation, by utilizing sensors embedded within HMDs. A detailed presentation of all tracking modalities has been recently published by Koulieris. et. al. [32], though we present the most widely adopted interaction methods below:

(a) (b)

Figure 2: Examples of two common interaction modalities in VR. (a) Interacting with a virtual aircraft cockpit in real-time, using using LeapMotion's hand-tracking controller, mounted onto an Oculus Rift VR HMD. (b) Fighting with a virtual enemy, using handheld wireless controllers.

- *Head Tracking* is the most widely adopted interaction method. As already mentioned, head rotation tracking is accomplished by utilizing the feed from IMUs (accelerometer, gyroscope) embedded in most today's HMDs. However, knowing the head position in 3D space is also necessary to allow for more DoFs. Thus, many HMDs combine IMU data with optical tracking to infer the headset's position in 3D space. This is the least convenient method for user interaction, as users are required to perform head movements that can become cumbersome, in order to interact with digital elements.

- *Hand Tracking* aims to reconstruct the pose of hands and fingers, which are crucial for our interaction with the real and/or virtual world. Hand tracking is a challenging problem because fingers look very similar and tend to occlude each other. Most hand tracking approaches use optical tracking by employing a pair of infrared cameras to capture depth. Subsequently, the hand tracking system's software uses sophisticated image processing and machine vision algorithms to convert the scanned 2D images to a 3D representation of hands, that can be used as input to a VR/MR system. Hand tracking provides unique advantages, such as being able to perform precise finger gestures and not having to constantly hold a controller (figure 2a).

- *Controller-based* interaction is also one of the most widely adopted interaction methods, and

are included with most consumer VR/MR headsets. Such controllers are wireless, battery-powered and come with a familiar gamepad-like ergonomic design, so that the average consumer almost instantly gets used to. Like head-tracking, IMUs embedded in the controllers allow for multiple DoFs. Moreover, most consumer-grade controllers are equipped with vibration motors, enabling **haptic feedback** and thus increasing immersion. On the flipside, handheld controllers have the drawback of not allowing precise interaction using finger gestures, as opposed to hand tracking. Furthermore, a user has to constantly be holding a controller to interact with the virtual environment (figure 2b).

- *Eye Tracking* aims to infer a user's gaze position, so that it can be utilized as a means of input. In this thesis, we thoroughly discuss the history and state-of-the-art approaches of eye tracking methodologies. After providing all the related background, we present our mobile VR eye tracking system and discuss its potential benefits, why and how it should be applied, as well as its weaknesses and limitations. In general, eye-tracking has the advantage of being non-intrusive while enabling new interaction modalities, not attainable through other means of input.

1.2 Problem Statement

Figure 3: A variety of mobile VR headsets are available to the average consumer at a low price tag. Most mobile VR headsets are equipped with head straps and adjustable lens position for increased comfort. However, user interaction with the Virtual Environment (VE) remains problematic, as it is mostly accomplished through tedious head movements.

Mobile Virtual Reality (VR) headsets, i.e., a mobile phone placed in an inexpensive cardboard or plastic case, represent a cheap and widely available medium for immersive VR experiences, targeting the average consumer. Input methods for interaction in smartphone-based VR are limited to uncom-

fortable head-tracking controlling a pointer on the screen (white cross in Figure 4), hampering user experience. A gamepad may be separately purchased, but these have their own restrictions, such as limited degrees of freedom (they only employ a four-way digital cross) and are only compatible with a few applications. Although this practice may seem functional and is easy to implement (as it requires no additional hardware), it quickly annoys users, offers limited interaction capabilities and thus does not aid in providing a pleasant VR user experience. Making matters worse, a user has to wait a fixed amount of time to trigger an action, to prevent unintended/false selections (yellow circle that fills in Figure 4). This problem is also known as the *Midas* touch problem [24]. Therefore, this technique is inapplicable in use cases that require real-time user action, due to the trigger delay and the time required by the user to adjust his head position.

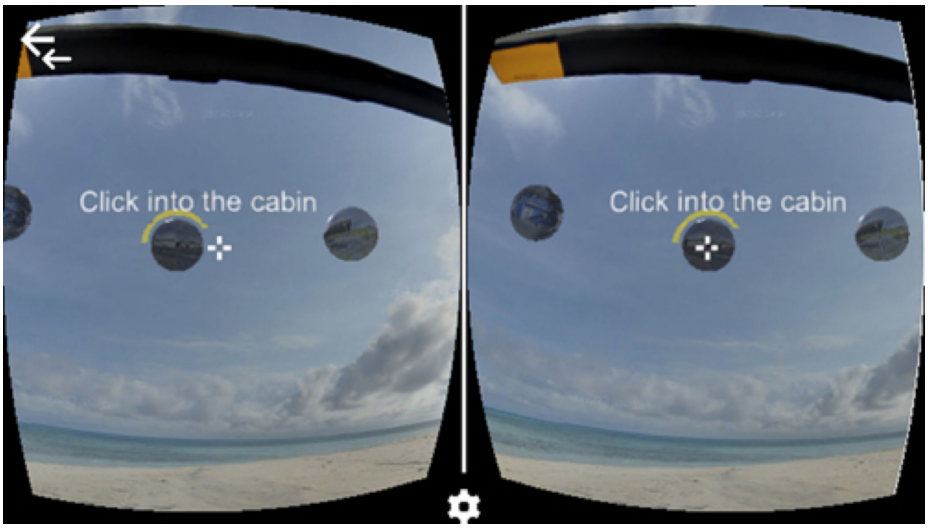


Figure 4: Input methods for interaction in smartphone-based VR are limited to uncomfortable head-tracking controlling a pointer on the screen. The vast majority smartphone-based VR applications require users to rotate their head, so that they position a pointer (usually represented by a cross or a circle in the middle of the screen) on top of the desired digital element. In this case, the pointer (white cross) has been placed on an interactable element and the waiting period has started. Once the yellow curve becomes a full circle, the action triggers.

Prior work has demonstrated that eye fixations is a natural input method for interaction with head-worn displays [5, 25]. We propose, for the first time, an innovative mobile VR eye tracking methodology, utilizing only the captured images of the front-facing (selfie) camera without any modifications or additional hardware. Embedded eye tracking is present only in certain specialized HMDs such as the HTC Vive Pro Eye and FOVE which employ an IR-based gaze detection system.

Mobile head-worn VR is easy to use and low-cost, but, without embedded eye tracking serving as a valuable affordance for interaction. The main reasons for in-existent eye tracking availability in mobile VR is the high production cost of embedded IR emitters and cameras. An initial attempt for smartphone-based eye tracking in mobile VR relied on coating the headset lenses with an anti-reflective layer which is a complicated alteration for the average smartphone user [12]. State of the art eye tracking relies on the enhanced pupil contrast provided by the infrared (IR) light of specialized eye tracking devices. However, eye illumination in mobile VR is based on the light emitted by the smartphone's screen. Robust eye tracking is, therefore, harder as light on the visible spectrum does not provide as much iris-pupil contrast as on the infrared. Furthermore, eye illumination, and thus, successful eye tracking depends on the displayed content. When the illumination is too strong, it produces obtrusive reflections on the headset's lens, making eye detection and tracking challenging. When the illumination is too weak, eye tracking fails as the iris is not visible. A detailed view of related work is presented in Chapter 2.

1.3 Contributions

Our eye tracking methodology for mobile VR enhances the low-quality camera-captured images that suffer from low contrast and poor lighting by combining a computationally efficient pipeline of low level image enhancements to suppress obtrusive reflections due to the headset lenses (Figure 1), and a thin, imperceptible frame of light surrounding the image content to always provide minimum illumination. The light bezel remains outside the visible field of view of the headset. The low level image enhancements facilitate the detection of features that we use to fit a geometric model to the iris, further refining its localization. We devise a computationally efficient circle fitting geometric algorithm to obtain an accurate estimation of the iris center based on customizing a Hough circle transform to be highly sensitive to circular features in the image such as the iris [16]. In order to select the best iris fit, we also devise a *confidence* measure which incorporates the probability of a candidate circle actually matching the iris, by evaluating each candidate based on a set of purpose-specific heuristics and hypotheses. Per-user calibration and gaze mapping algorithms are designed and implemented to convert the detected eye center co-ordinates to screen co-ordinates. To this end, we employ a two-stage, user-friendly calibration procedure which (1) detects and crops the eye movement region and (2) creates a mapping between iris locations and corresponding screen

locations. The aforementioned stages collectively require less than a minute to complete, after which our system is ready to conduct real-time eye-tracking. Per-user calibration is essential, as variable eye color, position, size, as well as HMD placement are factors that significantly affect gaze estimation accuracy and require purpose-specific mappings.

A formal experimental study confirms that the presented computationally efficient eye tracking methodology successfully infers user's gaze, performing comparably to commercial eye trackers when the eyes move in the central part of the headsets field of view and illumination conditions provide an unobstructed, reflection-free view of the iris through the headset's lens. We developed a simplistic, proof of concept VR game and a VR panorama viewer to further evaluate our eye tracking pipeline and compare it with head movement-based interaction. We demonstrate that our eye tracking module does not hamper task performance in comparison to head-tracked input, while, at the same time, it minimizes cumbersome head motions.

We make the following primary contributions:

- For the first time, we collect and thoroughly discuss the inherent challenges and difficulties associated with mobile VR eye-tracking, without the benefits of infrared light. We pave the ground for future research by analyzing and presenting each matter's effect on eye-tracking accuracy based on experimental evaluation.

- We present, for the first time, a novel iris detection and tracking technique for eye-tracking in smartphone-based mobile VR, based on front camera image capture of a standard mobile phone, through the headset's lens and without an added IR light source or other modifications (Section 3).

- Based on an evaluation study, we characterize the accuracy and precision of our method and devise UI and content design guidelines for accurate interaction in gaze-aware VR (Section 5.1).

- We demonstrate the strengths, limitations and potential future improvements of our eye tracking pipeline employed for an experimental study investigating iris detection accuracy and precision and in two proof-of-concept applications; a VR game and a VR panorama viewer (Sections 5.2 & 5.3).

1.4 Thesis Structure

The rest of the thesis is organized as follows:

- Chapter 2 discusses previous work in the field of gaze estimation relevant to the techniques described in this thesis. We demonstrate the strengths and weaknesses of state-of-the-art gaze-tracking algorithms and why they can or cannot be applied in our context. We present image processing operations that are more frequently used in gaze-tracking and discuss how calculated eye coordinates are mapped to screen coordinates.

- Chapter 3 gives a technical overview of our system in conjunction with the involved technical challenges. We define eye visibility as a multi-factorial issue by specifying the factors that affect the ability to perform accurate eye capture. We also discuss the importance of performance and optimization, since the presented system is intended to run on mobile devices.

- Chapter 4 is devoted to the implementation of the proposed system. We present each step of the pipeline in order of execution, from Stage 1 to Stage 3 (ROI detection, calibration and gaze-mapping, real-time iris tracking). Each step is comprehensively analyzed, including relevant code, equations figures and software/hardware used.

- Chapter 5 evaluates the system's accuracy and usability in realistic use-case scenarios by presenting the results of three distinct, purpose-specific experiments. Experiment 1 evaluates system accuracy and precision, experiment 2 evaluates our system's usability compared to 3DoF head-tracking in terms of task completion time and experiment 3 evaluates usability within a 360 VR panorama viewer.

- Chapter 6 concludes this thesis by thoroughly discussing current limitations and potential future work. We clearly state the causes of current limitations and discuss areas that future work could improve on.

2 Related Work

The proposed eye tracking pipeline for mobile VR is based on successful iris detection in real-time. In this section, we review previous image processing methods widely adopted in the field of eye-tracking for iris detection. We analyze why they are inapplicable or fail for eye tracking in mobile VR, mainly because of low lighting, the absence of infrared lighting and reflections due to the headset lenses.

2.1 Introduction to Eye Tracking

In this section we make an introduction to eye characteristics and eye tracking terminology, as well as a brief overview of eye tracking methodologies.

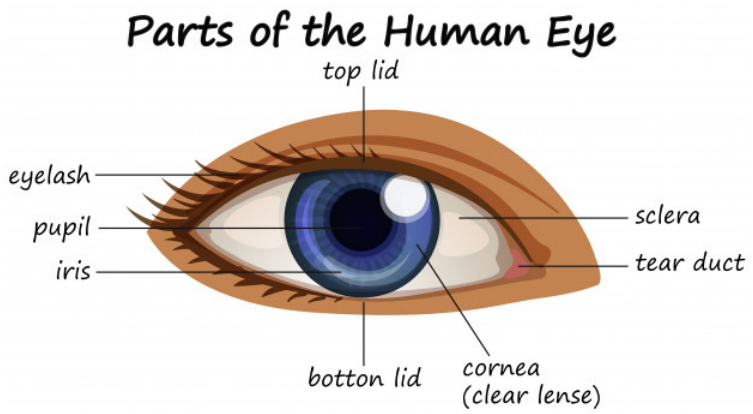
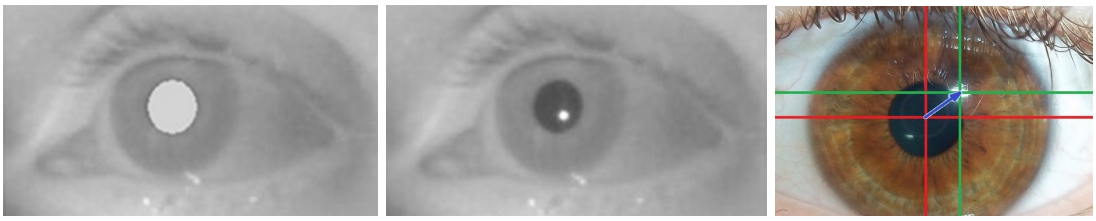


Figure 5: Illustration of basic human eye anatomy. The human eye is a slightly asymmetrical globe, which acts much like a digital camera. Light is focused primarily by the **cornea** — the clear front surface of the eye, which acts like a camera lens. The **iris** of the eye functions like the diaphragm of a camera, controlling the amount of light reaching the back of the eye by automatically adjusting the size of the **pupil** (aperture). Eye-tracking algorithms typically take advantage of iris-sclera or pupil-iris contrast to detect an accurate position of a user's eye.

2.1.1 Fundamentals of Eye tracking

Eye Tracking (also referred to as gaze-tracking) is the process of estimating a user's gaze direction (i.e. gaze estimation), so that it can be utilized as a means of input or for research, medical, scientific and marketing purposes. The most widely used current designs are video-based eye-trackers, in which a camera focuses on one or both eyes and records eye movement as the viewer looks at some kind of stimulus. Eye trackers can be desk-mounted, laptop-mounted, head-mounted



(a) (b) (c)

Figure 6: Eye tracking on the infrared spectrum (a) Infrared / Bright-pupil eye tracking (b) Infrared / Dark-pupil eye tracking (c) Visible light / center of iris (red), corneal reflection (green), and output vector (blue).

or using the front-facing camera of mobile phones and tablets. Near-eye input avoids the problems of head pose and eye-region estimation, and allows use of high-resolution images of the eye. Most eye trackers work in the infrared spectrum as dark irises appear brighter in it, resulting in stronger contrast to the pupil that is used for gaze estimation.

Infrared eye trackers can be divided in bright-pupil or dark-pupil, depending on the position of the infrared illumination source in relation to the eye [39]. In bright-pupil tracking, the illumination source is coaxial with the optical path. In this case, the eye acts as a retroreflector as the light reflects off the retina creating a bright pupil effect similar to red eye (figure 6a). In dark-pupil eye tracking, the illumination source is offset from the optical path; then the pupil appears dark because the retroreflection from the retina is directed away from the camera (figure 6b).

Eye movements are typically divided into **fixations** and **saccades** – when the eye gaze pauses in a certain position, and when it moves to another position, respectively. The resulting series of fixations and saccades is called a scanpath. Smooth pursuit describes the eye motion when following a moving object [34]. During a saccade, peak angular speeds of up to 900°/s [9] can be reached. At the same time, there is a dramatic decline in visual sensitivity, which is referred to as saccadic suppression [44]. As a result, during saccadic eye movements, accurate visual information cannot be acquired. In contrast, fixations describe the state and duration in which visual information is perceived while our gaze remains close to an object of interest. A more detailed summary of the Human Vision System (HVS) and oculomotor motions in relation to eye tracking is presented by Koulieris et al. [32]

2.1.2 Eye tracking algorithms

Eye tracking algorithms can be classified as *feature-based*, *model-based* and *machine learning-based* in relation to how the eye is detected. **Feature-based** algorithms detect a key feature, e.g. the dark pupil, by relying on low-level image analysis such as pixel intensity levels or intensity gradients. A pixel intensity threshold defined by the developer, usually through trial-and-error, decides whether the key feature selected is present or not, based on the application goals. The most common feature-based method in IR eye-tracking utilizes the iris-appearance hypothesis, according to which the pupil is the darkest element to detect [49].

Model-based approaches find the best fitting circle or ellipse of the pupil. Due to exhaustive and iterative searches of the model parameter space, model-based methods are more resistant to illumination changes and image noise; thus yield a more precise estimate of the pupil center than feature-based, but at a higher computational cost. Consequently, their use is challenging in time-critical mobile VR eye-tracking, as resources are limited and dropping frames is not an option. On the other hand, an accurate eye-tracking system cannot solely rely on feature-based algorithms, due to the highly variable illumination conditions, making it impossible to achieve stable and real-time iris detection.

More recently, **machine learning-based** approaches have emerged in the field of eye-tracking, as neural networks have been successfully utilized in a plethora of computer vision applications [51, 35, 16]. Such methods work by using a set of training (calibration) eye images associated with ground-truth gaze positions to train a convolutional or artificial neural network (CNN or ANN). Once the gaze estimation function is trained, it can directly output gaze positions from arbitrary input eye images. The main challenges associated with machine learning -based approached are the complexity, performance and large amount of labeled data required for sufficient training.

Standard eye-tracking on the infrared is based on a trade-off between run-time performance and accuracy by combining model and feature-based approaches [32, 36, 21]. Our VR mobile eye tracking system deploys a purpose-specific feature-based algorithm to determine the approximate position of the eye and then a computationally efficient model-based method in the form of a circle fitting algorithm is deployed to obtain a more accurate estimation of the iris center. In the following chapters, we present state-of-the-art approaches for video-based eye tracking, by primarily focusing on head-mounted eye-trackers which are more relevant to our work.

2.2 Eye, iris & pupil detection

There are several ways to detect an object of a circular shape in an image. Variations of ellipse fitting methods are commonly used in the field of eye tracking based on IR emitters to locate the iris or the pupil. The simplest way to fit an ellipse to a set of data is using the direct least squares method [11], which requires at least five points as input. However, this method is not applicable to most eye-tracking algorithms, as the input edge images contain pixels which do not correspond to the pupil or eye boundary. These pixels are leftover image noise, representing also other noticeable features such as eyelids or reflections, troublesome to remove.

The more robust ellipse-fitting methods can be categorized as voting-based or search-based. Hough transform and Random Sample Consensus (RANSAC) [10] are primary examples of voting-based and search-based methods respectively. Voting-based methods are exhaustive and thus accurate, but computationally expensive. Searching-based methods test subsets of possible ellipses over many iterations and select the best iteration. Proposed feature-based methods for eye tracking include K-means segmentation to segment the dark pupil from the background and locating the darkest area, based on the iris-appearance hypothesis according to which the pupil is the darkest element in the image [49, 45].

2.2.1 Image Pre-processing

Prior to attempting pupil or iris recognition, it is a common approach amongst eye-tracking algorithms to perform image processing operations in the form of edge manipulation and/or image quality improvement for best results. The vast majority of eye-tracking algorithms operate on edge images, by applying an edge detection technique such as the **Canny** edge detector [6], as a means to extract useful structural information from the image, including elliptic shapes corresponding to the eye. Other edge detection methods are also utilized, like for example the Sobel operator [47]. The **Sobel** operator can be particularly useful in detecting circular shapes, since it convolves two kernels, one for the horizontal and one for the vertical direction. Thus, it has the ability to compute a gradient vector for each pixel. The aforementioned characteristic of the Sobel operator is taken advantage of by popular circle-fitting methods such as the Hough Gradient method [40]. Other edge detection techniques also exist, however they are not frequently used in machine vision. Studies and comparison of edge detection techniques have been previously conducted by Maini et al. [38].

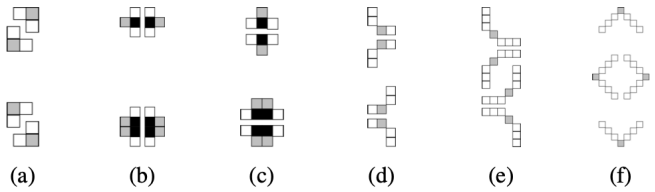


Figure 7: Pixel manipulation filters used in ElSe, ExCuSe and PuRe to manipulate edges, as a pre-processing step prior ellipse-fitting. If the pattern matches an edge segment, gray pixels are removed and black pixels are added to the edge image. Operand (a) thins lines. Operands (b) and (c) are used to straighten lines. (d), (e) and (f) separate straight parts of a line from curved parts. [14, 45, 12]

.

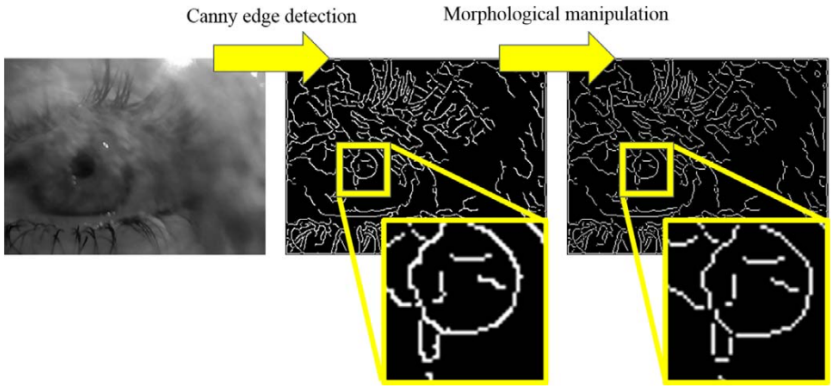


Figure 8: Input image (left), resulting Canny edge detection (middle), and edges after morphological manipulation by applying pixel manipulation filters such as the ones illustrated in figure 7. Notice how the edges are thinned and orthogonal connections are broken [45].

State of the art eye-tracking methods take edge manipulation one step further by applying **morphological** pixel operations on the edge image, such as removing standalone or straight edges and breaking orthogonal connections (figure 7). The main idea is that, according to studies, certain types of edges, such as edges with too many neighbours or straight lines cannot belong to an ellipse. Therefore, these edges can be removed by applying morphological filters so that only edges useful to ellipse-fitting are kept [14, 45]. Other morphological operators are also found to be utilized. For instance, the morphological 'open' (erosion followed by dilation) operation is shown to reduce noise and remove small occlusions without significantly affecting the pupil's contour [49].

Less frequently, **image thresholding** (also referred to as binary thresholding) has been used to segment the eye region [53] or the pupil (see figure 9), given that the pupil is the darkest element in the image, which is often the case in infrared eye-tracking. However, setting an optimal threshold value is challenging, as illumination of the eye changes. For this reason, algorithms making use of image thresholding are required to implement automatic threshold calculation, which adapts to such changes. An implementation of the aforementioned approach has been presented in related work, in which the authors determined the threshold's value by applying k-means clustering on the histogram of the pupil region (figure 9b). The maximum intensity of the dark cluster is set as the threshold's value [49]. It is noteworthy that binary thresholding is often performed as a preprocessing step in combination with other methods, as obtaining a highly accurate pupil center directly from thresholding is often problematic. Hence, eye-tracking algorithms that make use of binary thresholding combine it with additional methods to refine the pupil's center position, such as ellipse-fitting.

In addition to edge manipulation, image quality improvement techniques are also applied for best results, including histogram equalization, noise reduction and reflection suppression. **Histogram equalization** is commonly performed in the field of image processing to enhance the contrast of an image. Captured eye images under sub-optimal illumination tend to suffer from poor contrast. The intensities of such images are confined in a small range of values. Histogram equalization expands low contrast image areas by spreading out their most frequent intensity values to cover the entire available dynamic range, resulting in a more uniform intensity distribution. The result is a more salient eye.

Reflections correspond to one of the brightest regions in the eye image. It is best that they are

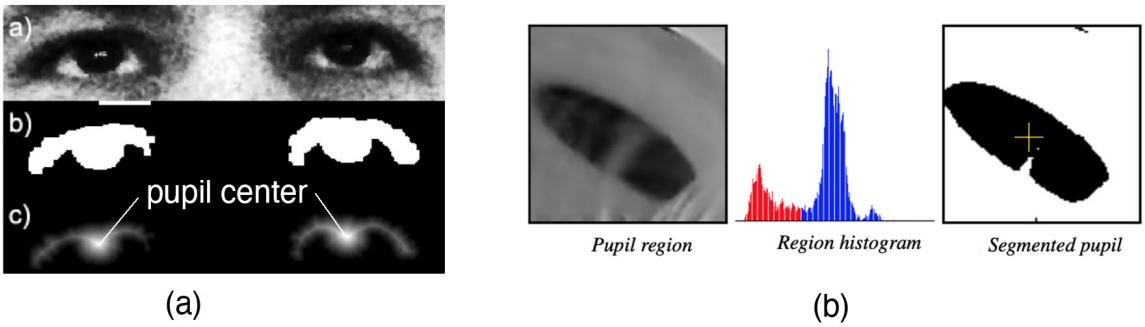


Figure 9: Two examples of using image thresholding to estimate the pupil's center. (a) Locating the pupil's center by calculating the image geometric moments, enclosed in the white area after binarization [53] (b) Locating the pupil's center by adaptively thresholding the image, by performing k-means clustering on the histogram to infer the threshold [49]

removed or suppressed, as their presence can interfere with the process of circle-fitting resulting in failure. However, their precise localization and removal is not trivial, as they vary in size, shape and intensity depending on illumination and surrounding environment. Fortunately, in scenarios were reflections are expected to be of roughly constant size and/or shape, previous work has demonstrated that they can be dealt with. To this end, Li et al. developed a corneal reflection removal algorithm which handles the issue as a minimization problem. The algorithm assumes that the intensity profile of the corneal reflection follows a bivariate Gaussian distribution, given that the constant presence of an infrared light source in front of the eye produces a small circular reflection. To determine the full extent of the corneal reflection, the algorithm computes the radius r where the average decline in intensity is maximal through a gradient descent search that minimizes the value of a parameter.

2.2.2 Locating the iris/pupil via feature-based and model-based methods

The majority of eye detection algorithms achieve a good trade off between run-time performance and accuracy by combining model-based and feature-based approaches. The Starburst algorithm iteratively locates the pupil center as the mean of points which exceed a differential luminance threshold along the rays extending from the last best guess [36]. To achieve correct results, the algorithm first locates and removes corneal reflections (as described in 2.2.1) increasing the brightness of the normally dark pupil. In relation to the SET method, the convex hull segments of thresholded regions are fit to sinusoidal components that compose the circle corresponding to the eye [25]. The ellipse that is closest to a circle, is considered as the pupil. ElSe, PuRe, ExCuSe and Pupil

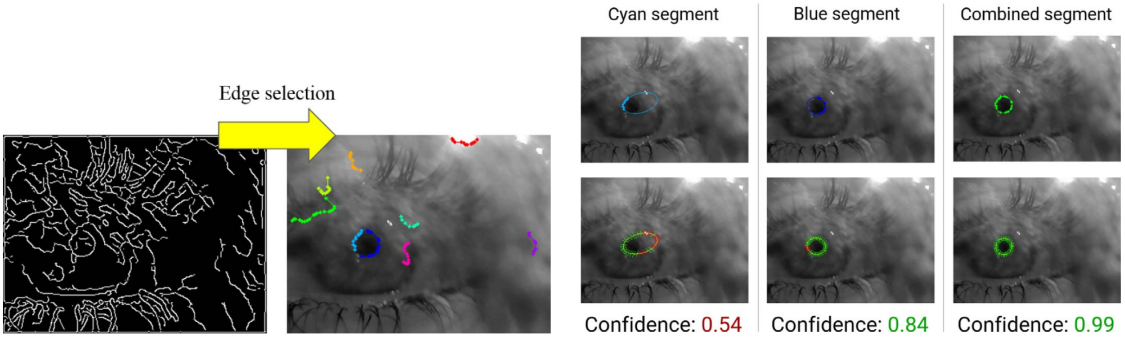


Figure 10: PuRe's state-of-the-art implementation of feature and model-based pupil detection combination [45]. (a) Edges after morphological processing (left) and the resulting selected segments that are candidates for the pupil outline (right). Each segment is represented by its k-cosine chain approximation and illustrated with a distinct color. (b) Confidence value for each edge segment. Segments with confidence lower than 0.5 are omitted, hence the two remaining segments, cyan and blue.

methods use morphological edge filters to thin, straighten or separate lines so that ellipse fitting can be successfully applied [14, 45, 12, 30]. ElSe utilizes a set of heuristics and restrictions for ellipse evaluation, such as the ratio between the two ellipse radii and the intensity of the enclosed area. For instance, the pupil position relative to the eye tracker camera can only distort the pupil ellipse eccentricity to a certain point. Therefore, the ratio between the two ellipse radii can't exceed a certain value. PuRe applies similar evaluation heuristics with the addition of an "angular edge spread" metric, assuming that the better distributed the edges are, the more likely it is that the edges originated from a clearly defined elliptical shape (i.e., a pupil's shape). To further improve results, authors fit an ellipse on each edge segment and take into account the aforementioned metrics and compute a 'confidence' measure for each edge segment (see figure 10b). The higher the confidence value, the more likely is that the edge belongs to the pupil's outline. Candidates that yield confidence value below a specified threshold are discarded, as it is highly unlikely that they correspond to the pupil. It is noteworthy that edge segments are typically approximated by a set of dominant points using the k-cosine chain approximation method [50] before applying ellipse fitting. This approximation reduces computational requirements and results in a better ellipse fit [45].

ScreenGlint utilizes a smartphone and its front-facing camera as an eye tracking device and tracks the user's gaze by exploiting the screen's reflection on the cornea [21]. The method uses the

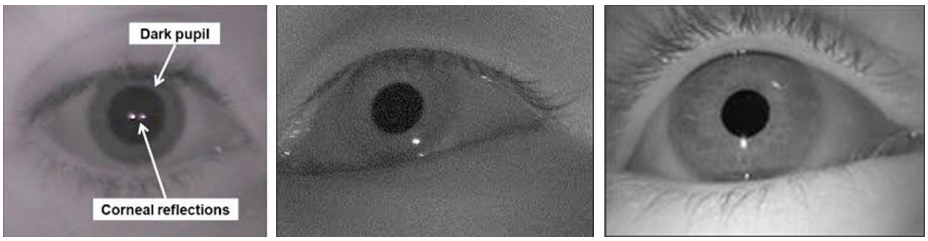


Figure 11: In **dark-pupil eye-tracking**, the pupil appears much darker than its surroundings, hence the name. This aids the pupil recognition process, as it is the most salient part of the image. However, dark-pupil eye-tracking requires an infrared (IR) light source, which involves additional hardware and modifications, not available to the average mobile VR user.

iris-appearance hypothesis which assumes that the iris corresponds to the darkest area to roughly locate its position within the image and then apply ellipse fitting to obtain an accurate position of the iris center. The gaze point is inferred from the relative position between the glint and the iris center (see figure 16b).

The main drawback of the majority of image processing algorithms for standard IR-based eye tracking analyzed above, is that they heavily rely on the enhanced pupil contrast provided by the infrared light of specialized eye tracking devices. For a dark-pupil optical setup, where the illumination is off-axis to the camera, the pupil appears much darker than its surroundings in the infrared spectrum. Light sources on the visible spectrum do not provide as much contrast as on the infrared, hence, eye detection is not as robust as on the infrared. Therefore, the processing pipelines presented in such algorithms will more than likely likely fail if applied to RGB eye-tracking. Wojciechowski et al. presented a novel gaze-estimation method on the visible spectrum by using only a single web camera with 1.5°-3°accuracy [53]. Using a Haar-like feature classifier, the authors detect and crop the eye region on the face. The gaze-estimation algorithm begins by applying histogram equalization, adaptive binary thresholding and morphological opening to reduce noise and improve image quality. The pupil's center is computed by determining the "center of gravity" point on the binarized image (see figure 9a). The authors also developed a custom blink detection function that determines whether the eyes are open or closed, by utilizing the convex hull of the binarized eye region.

2.2.3 Locating the iris/pupil via machine learning-based methods

Previous work has demonstrated that utilizing neural networks for gaze-estimation [4, 46, 16, 51, 55] or classification [54] is possible, even real-time in modern mobile devices, given that the neural network is optimized accordingly [35]. Setting and optimizing a neural network is not trivial, as training requires a large image dataset, often difficult to obtain. Furthermore, training with original sized images significantly increases the number of connections in the neural network and thus hits performance. To this end, authors in related work trained their models with significantly reduced image sizes by cropping and/or downsampling the images in their dataset.

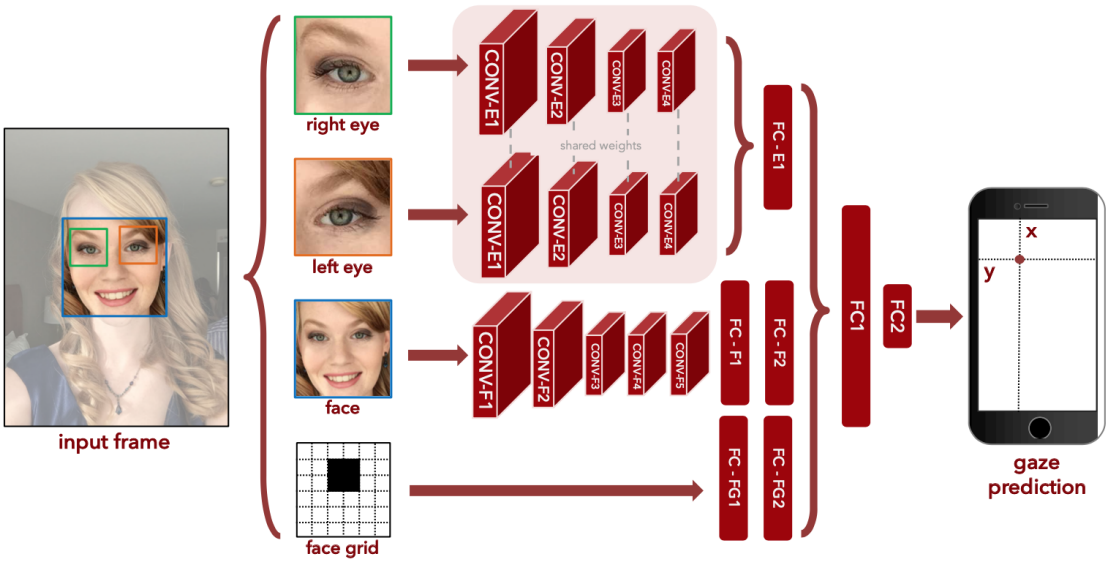


Figure 12: Overview of iTracker CNN, the most robust smartphone-based eye-tracking setup to date that achieves accuracy of 1-2cm [35]. 'CONV' represents convolutional layers, while 'FC' represents fully connected layers.

In 2017, Fuhl et al. proposed a double CNN approach, devoting one CNN for coarse positioning and an additional one to refine the center estimate. By performing evaluation on over 135,000 images, the authors claim that their system outperformed feature-based and model-based state-of-the-art algorithms at that time [13]. In unintrusive gaze-tracking applications where the entire face is captured, it is a common approach to crop the eye region as a first step by using Haar-like feature detection or other facial landmark detection algorithms [3]. Krafka et al. reportedly achieved 10-15fps performance for the first time on a smartphone (iPhone 6S), by training a CNN with

approximately 2.5 million eye images (downscaled to 80x80 pixels in size) with their corresponding fixation locations, collected through crowdsourcing. Their system achieves real-time gaze prediction on phones and tablets with error in the 1-2cm range. Earlier, Tonsen et al. achieved gaze estimation with an accuracy of 1.79°using unobtrusive cameras mounted on a glasses frame [51], a favorable condition for AR setups. To do so, an ANN was trained with an image resolution of only 5x5 pixels. At this point it important to note that unconstrained remote gaze tracking is challenging, as variable head pose can severely impact accuracy. To this end, Ranjan et al. proposed a branched CNN architecture that aims to improve the robustness of gaze classifiers to variable head pose without affecting performance [42]. This is accomplished by clustering the training data in K clusters in combination with having K branches in the final layer of the network.

2.3 From eye coordinates to screen coordinates

Calculating the user's point of gaze requires a conversion of the eye-tracker's estimated iris center to screen locations. The conversion of eye-tracker data to screen coordinates is accomplished by a **mapping function**, so that the estimated gaze position on the screen can be obtained as such:

$$\begin{aligned} f_x(x_{iris}, y_{iris}) &\rightarrow x_{screen} \\ f_y(x_{iris}, y_{iris}) &\rightarrow y_{screen} \end{aligned} \tag{1}$$

The mapping function is determined per user through calibration, i.e. fixation on a set of presented target points of known positions [48, 36, 30], during which the calculated iris and the displayed target point's positions are recorded. Given several of these position correspondences, the mapping function's coefficients can be computed.

2.3.1 Mapping function types

The mapping function should minimize calibration errors and correct distortions between the eye-tracker and screen as much as possible. The selection of the appropriate type of mapping function is a critical factor of eye-tracking accuracy. Mapping functions used are of several types, ranging from piecewise, linear & nonlinear polynomials [48, 29], geometric-based to neural network and support vector regression (SVR) based functions [56, 57]. The piecewise is the most primitive form

of mapping; it divides the screen into a grid of cells, and a target point is presented at each grid junction. The data gathered from the tracker then defines a grid of quadrilaterals, each of which is separately mapped back onto its original rectangular grid cell. This method is not ideal for eye-tracking, as abrupt changes can occur at the boundaries between grid cells, unless a large number of data points is collected.

According to literature, the most common choice for calibration is usage of polynomial functions, which can differ in the degree and number of terms. The most popular representatives of polynomial functions is the linear(x^1) and quadratic(x^2) equation, shown in equations 2 and 3, where (x_s, y_s) are screen coordinates and (x_i, y_i) are the estimated iris center coordinates. The coefficients are computed at the end of calibration, given enough (x_s, y_s) , (x_i, y_i) correspondences.

Linear equation:

$$\begin{aligned} x_s &= A_x x_i + B_x y_i + C_x \\ y_s &= A_y x_i + B_y y_i + C_y \end{aligned} \tag{2}$$

Quadratic equation:

$$\begin{aligned} x_s &= A_x x_i^2 + B_x y_i^2 + C_x x_i + D_x y_i + E_x \\ y_s &= A_y x_i^2 + B_y y_i^2 + C_y x_i + D_y y_i + E_y \end{aligned} \tag{3}$$

The coefficients can be computed by solving an m x m matrix using Cramer's rule, m being the number of equations and unknowns, or using a least square curve fitting method such as the Levenberg–Marquardt algorithm.

Previous research has shown that there is no one-size-fits-all mapping function for all eye-tracking applications. High order polynomial mapping resulting from an increased number of calibration points or complex geometry-based models in scenarios with limited head movement and an adequate number of calibration points do not improve accuracy over simpler linear functions [7]. However, in cases with a considerable amount of head movement, geometry-based methods present higher robustness and outperform polynomial-based functions [29]. Simpler regression functions like x^1 or x^2 seem to operate better than more complicated (like x^3 or SVR) on sets with lower number of points. In the case of nonlinear functions, the nonlinear terms allow curved distortions to be corrected,

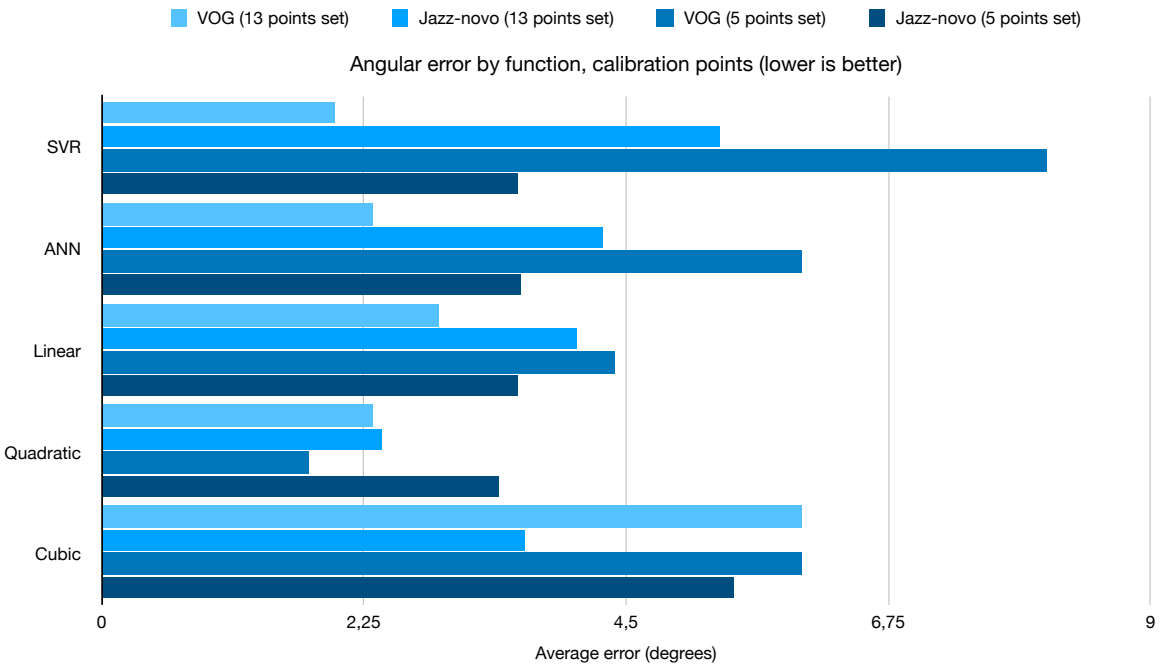


Figure 13: Average angular error (in degrees) in two different eye-tracking systems (VOG, Jazz-Novo) by mapping function and calibration points. Functions compared: Support Vector Regression (SVR), Artificial Neural Network (ANN), Linear, Bicubic, Quadratic. High order polynomials, SVR and ANN are outperformed by linear function when fewer points are used. Chart constructed from the data presented in [29].

accommodating fixation angle changes across the screen. However, the nonlinear characteristics of the function has negative effects as well. The squared terms in the equation become very large as the gaze position moves away from the (0,0) point, and small errors in calibration-point fixation or head movements can result in large errors in screen gaze position there. This is the main reason why nonlinear functions require more calibration points to provide sufficient accuracy, as mentioned in figure 15.

Mapping can also be performed by employing a trained **neural network** with comparable accuracy, although they are usually avoided due to their higher complexity and the additional effort required for sufficient training. Given enough input/output samples, neural networks for such applications can be trained using *Back Propagation* algorithms, until the total train error becomes lower than a heuristically defined threshold.

It should also be emphasized that it takes some time a human eye to react to stimulus position change and thus fixate on another position. Such occurrence is known as **saccadic latency** and

can last up to 400msec [2]. Consequently, an eye-tracker should not record the iris position during the saccade following a stimulus change to avoid the collection of false gaze data that can essentially invalidate calibration.

In summary, calibration results and thus gaze estimation accuracy greatly depends not only on the type of mapping function, but also on the type of eye-tracker used and number of calibration points. Additionally, it is required that subjects properly fixate on the displayed targets and eye-trackers pause capturing during saccades or blinks.

2.3.2 Calibration points

Another factor that should be considered is the number of calibration points presented to the user. Although it may seem obvious that a large amount of calibration points can facilitate better eye-tracking accuracy, this is not necessarily the case. Previous research shows that accuracy increase for over 10 points is minimal. Furthermore, participants may become tired or annoyed by a long calibration procedure that involves a large amount of calibration points, which could result in loss of concentration and thus collection of false data. It is of upmost importance that the subject correctly fixates on the presented gaze points. Failure to do so will result in incorrect calibration and thus inaccurate eye-tracking. Taking into account the data presented in figure 15 and the aforementioned factors, a 9-point calibration is ideal for linear or quadratic functions, while cubic or higher degree functions require at least 12 points to provide the same level of accuracy [29]. As mentioned in the previous chapter, nonlinear terms can exaggerate calibration errors, especially as the gaze position drifts away from the (0,0) point. This side-effect is counteracted by adding more points to the calibration procedure.

As a general rule of thumb, the points should be evenly spread out on the area to be tracked. Taking the aforementioned facts into consideration, in addition to our requirement for computational efficiency and user-friendly procedure, we employ a 12-point calibration with a linear mapping function, as our early lab tests confirmed that more complex mapping methods do not provide any meaningful accuracy improvements worth the effort.

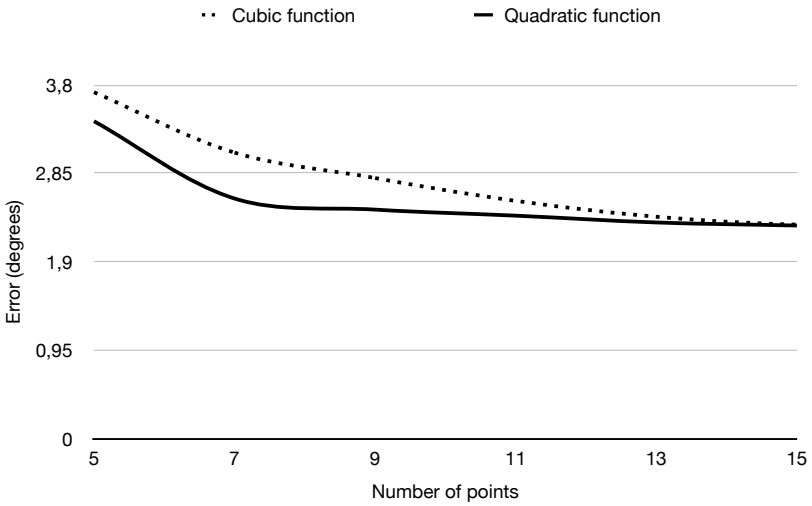


Figure 14: Gaze estimation error (in degrees) in relation to the number of calibration points. Chart derived from the data presented in [29]. Higher degree polynomial functions evidently benefit from more calibration points.

2.4 Eye tracking in mobile VR

A preliminary attempt for smartphone-based eye tracking in mobile VR relied on coating the headset lenses with an anti-reflective layer which is a complicated alteration for the average smartphone user [18]. Another approach, realised on a Google Cardboard headset, compared reflected images of on-screen content on the surface of the eye, known as Purkinje images, to pre-calibrated images to infer an estimated gaze position [17].

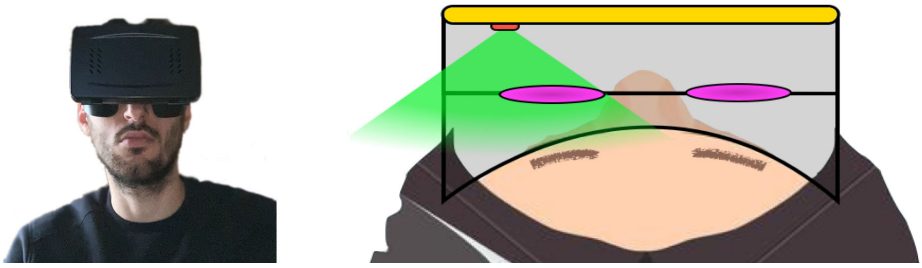


Figure 15: Left: User wearing a mobile VR headset. Right: Top down schematic view of the VR headset when worn. The smartphone (yellow) camera (red) has a field of view (green) that encompasses one of the headset's two lenses (pink), allowing it to see one of the wearer's eyes.

This approach did not work in real-time, was not implemented on the mobile device, required add-ons such as mirrors for achieving wall-clock synchronization between camera frames and display

frames and functioned only on a specific set of calibrated Purkinje images, limiting applicability.

More recently, coarse gaze tracking on a smartphone-based VR headset used a Convolutional Neural Network (CNN) trained on a large number of peri-ocular images. The network was able to predict one of five fixed gaze locations[1]. The reported system's accuracy of nearly 10° when calibrated is prohibitive for real-time gaze-based interaction. We note that commercial eye trackers in VR headsets offer an accuracy of less than 1° in the central FoV. In addition, the approximate gaze location was computed on a laptop, therefore, the system's performance and latency were never tested on a mobile device. Other approaches employing CNNs to infer eye pose relative to the head from a single image did not address the specific challenges of low contrast and illumination of real-time VR mobile eye tracking [35].

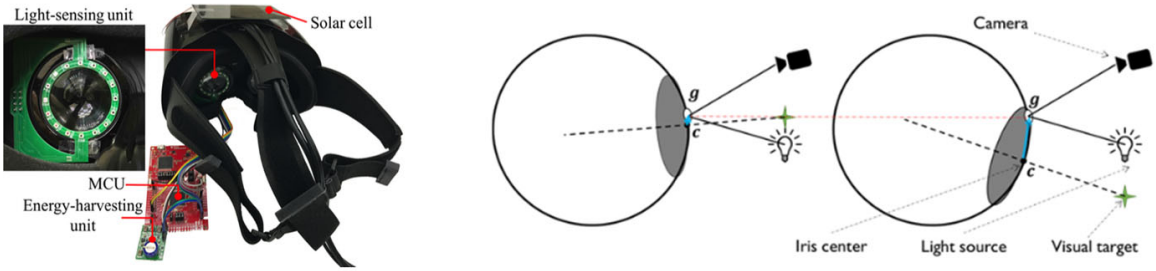


Figure 16: Unconventional VR gaze-estimation methods. (a) Gaze estimation based on the exploitation of pupil's light absorption property with the use of photodiodes (LiGaze), (b) Gaze estimation based on corneal reflection. The location of the glint on the cornea is relatively stable; therefore, the relative location between the glint and iris center can help to indicate the gaze point (ScreenGlint).

Katrychuk et. al. improves on existing Photosensor oculography (PSOG) technology, which captures the IR reflections on the eye using IR detectors by employing a CNN coupled with transfer learning [31]. The authors acknowledge that headset shifts due to facial expressions and free movement are a common issue in VR that degrades eye-tracking accuracy. As mentioned by Rigas et al., the shift of more than 0.5mm can result in more than 1.0°spatial accuracy degradation [43]. To this end, they claim that their setup is robust to sensor shifts, by placing a marked spot at the wearer's nasal bone area as a reference point to detect potential position shifts. The marker is recognized by their preprocessing algorithm, which makes the necessary adjustments in case the marker is found to have moved.

LiGaze [37] presented an interesting gaze-tracking method, by placing passive photodiodes on the perimeter of headset's lens coupled with a small solar panel to harvest energy from indoor lighting (Figure 16a). The key idea of this approach is that the placement of the photodiodes enables measurement of the reflected light in different directions. By exploiting pupil's light absorption property, the authors claim that they were able to detect pupil movement and consequently infer gaze on the fly with roughly 5°accuracy.

Gaze prediction based on machine learning specifically for games, associated game variables with player actions, but cannot be generalized to real-time mobile VR eye tracking in any context [33]. ScreenGlint utilizes a smartphone and its front-facing camera for tracking the user's gaze by exploiting the screen's reflection on the cornea [21], however, this approach was only tested when holding the smartphone at a distance and not inside a VR headset (Figure 16b). A comprehensive review of generic gaze estimation techniques can be found here [28].

3 Challenges and System Overview

3.1 Challenges

In this section we discuss the specific technical challenges of VR mobile eye tracking systems that function on the visible spectrum and then describe a proposed architecture for mobile VR eye tracking to address these challenges:

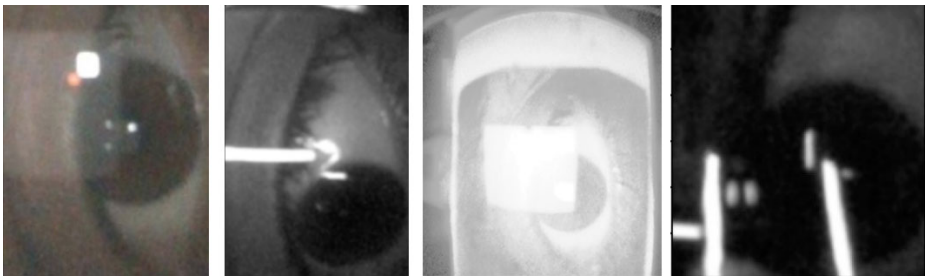


Figure 17: Eye-tracking example on the visible spectrum. As depicted in the figure, eye-tracking is far more challenging than on the IR spectrum, as light sources on the visible spectrum do not provide as much pupil contrast as on the infrared. Making matters worse, light source reflections cast onto the headset lens obscure the eye, making eye-tracking even more difficult.

1. **Headset lens reflections**: The mobile VR head-worn case lens sits between the captured eye and the smartphone's front facing camera. As a result, reflections of variable shape, size and intensity originating from the displayed content are cast onto the lens of commodity VR headsets (plastic lenses of poorer optical clarity). Consequently, a large part of the iris may be occluded (Figure 33), making iris detection a challenging and non-trivial task.

2. **Corneal reflections**: Due to the small distance between the user's eye and the smartphone's screen, severe reflections on the eye cornea are also expected (Figure 32). These reflections introduce noise during the image processing stage and, thus, decrease the accuracy of gaze estimation. We apply a series of image processing operations intended to first suppress reflections and then expand the contrast of the residual eye image to improve iris visibility (Section 4.4).

3. **Content limitations**: Since no additional light sources are used to illuminate the eye in VR mobile eye tracking, in contrast to specialized eye tracking devices, eye illumination solely depends on the brightness of the screen, which is directly related to the colors and shapes of the displayed content. If the displayed content is too dark, the eye will not be illuminated

enough for the iris to be detected. We devise a thin, imperceptible frame of light surrounding the image content to always provide minimum illumination (Section 4.4).

4. **Position of front-facing camera**: Most smartphones have the front camera placed significantly off-center-axis with respect to the eye. This results to an oblique view of the eye, complicating pupil registration. Our geometric circle fitting algorithm iteratively improves upon an initial pupil location estimate to maximize accuracy (Section 4.4).

5. **Smartphone & HMD placement**: Smartphone and head mounting positions vary across HMDs. Consequently, a clear view of the user's eye should not be taken for granted. Nevertheless, per-user calibration is always required to conduct algorithmic adjustments to accompany the HMD's and smartphone placement in respect to the user's eyes and thus calculate the correct eye-to-screen mappings.

6. **Eye characteristics**: Our early lab experiments showed that certain eye characteristics are responsible for eye-tracking errors. In visible spectrum eye-tracking, iris contrast is crucial to accurate eye detection. It was found that darker irises are more salient and thus yield higher eye-tracking accuracy, due to the higher contrast with the eye's sclera. On the contrary, light-colored irises provide less contrast and thus were directly associated with large eye detection errors. Furthermore, eye position was also found to impact eye detection in case it is obstructed by the HMD's plastic frame.

7. **Hardware and performance**: Eye tracking involves simultaneously executing several, time-critical processes including video capture, image cropping & enhancement and coordinate systems re-mapping. Making matters worse, a VR application requires tremendous amounts of processing power to run smoothly. The simultaneous operation of eye tracking and a VR application on a single device can easily overwhelm even the latest generation of smartphones. Our system works without requiring any computationally-heavy classifiers based on machine learning, enabling its fast execution even on older generation smartphones or cheap current generation ones.

3.1.1 Eye visibilty through the headset's lens

Once we had set up our development environment and had our tools ready, the early stages of development were mainly focused on collecting eye image samples from the front facing camera in different lighting conditions, so that we could infer what kind of challenges and difficulties we would face.

For this matter, we developed a simplistic iOS application in XCode to capture the iPhone's front facing camera stream. By gaining access to the stream, we could view and save the data as photos at will. Having mounted the headset with the phone in it, the application captures the eye through the headset's lens while displaying a set of pre-selected images of varying brightness and contrast, to simulate different lighting conditions.

By analyzing the captured images, we came to the following realisations:

- The light emitted from the phone's screen casts reflections on the headset's lens, thus obstructing the eye and making it challenging for a computer vision algorithm to identify.
- The **intensity** of the reflections is proportional to the brightness of the screen and thus the displayed content.
- The **shape** and **size** of reflections is dependent on the shapes displayed and especially the contrast between them. The higher the contrast between shapes, the more pronounced are the shapes of the reflections, and thus more obtrusive.
- On the one hand, darker displayed content may reduce reflections to a minimum, but on the other hand, the eye is poorly lit and barely visible. Figure 8 shows that screen brightness lower than 70-80% does not provide sufficient illumination for the eye when the on-screen content is dark, despite the addition of a thin light bezel at the top of the screen to aid illumination.

The above findings clearly indicate early-on that gaze-mapping under these lighting conditions is remarkably challenging, as in the vast majority of lighting scenarios the iris remains non-salient and is obstructed more or less by the reflections cast on the headset's lens. It is reminded that the above challenges would not be present, had we utilized the infared spectrum of light instead of the visible, which requires additional hardware, as thoroughly discussed in the introduction.

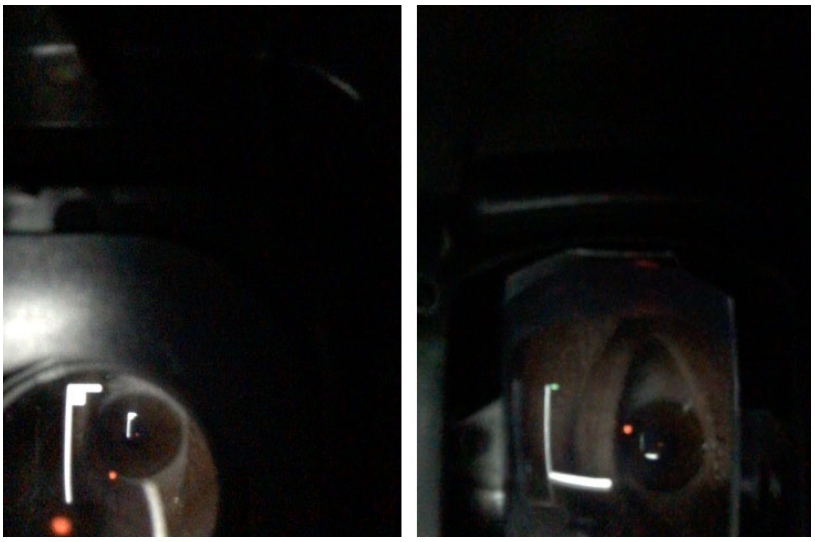


Figure 18: Raw front-facing camera frames, from two different mobile VR headsets. Our study shows that eye-tracking is possible with any mobile VR headset, as long as the iris remains visible to the front-facing camera at all times. In this case, we observe that the plastic frame of the HMD on the left partially obstructs the eye region, while the HMD on the right allows the front-facing camera a clearer view.

3.1.2 Performance & Optimization

Performance is a critical factor of all applications that directly affects the stability and operation of hardware and software. Software has to be designed and developed by balancing available processing power and computational requirements of the tasks to be executed. It is often the case that trade-offs have to be made, as computational resources are finite and thus may not be able to handle computationally expensive tasks without introducing unwanted behaviour, such as overheating and crashes.

Processing power on mobile devices has massively increased the past decade, thus allowing for previously impossible applications, such as mobile Virtual Reality. Smartphone manufacturers provide significant hardware improvements, even over each phone generation. Regardless, software intended for mobile devices has to be carefully optimized. Due to their compact size and lack of a cooling mechanism, mobile devices can easily overheat if their processing power is utilized to its maximum for more than a few seconds. Furthermore, battery drain is drastically increased as CPU utilization increases. To this end, smartphone manufacturers implement built-in mechanisms to reduce the heat generated by the chip and conserve energy, so that the chip's temperature is kept

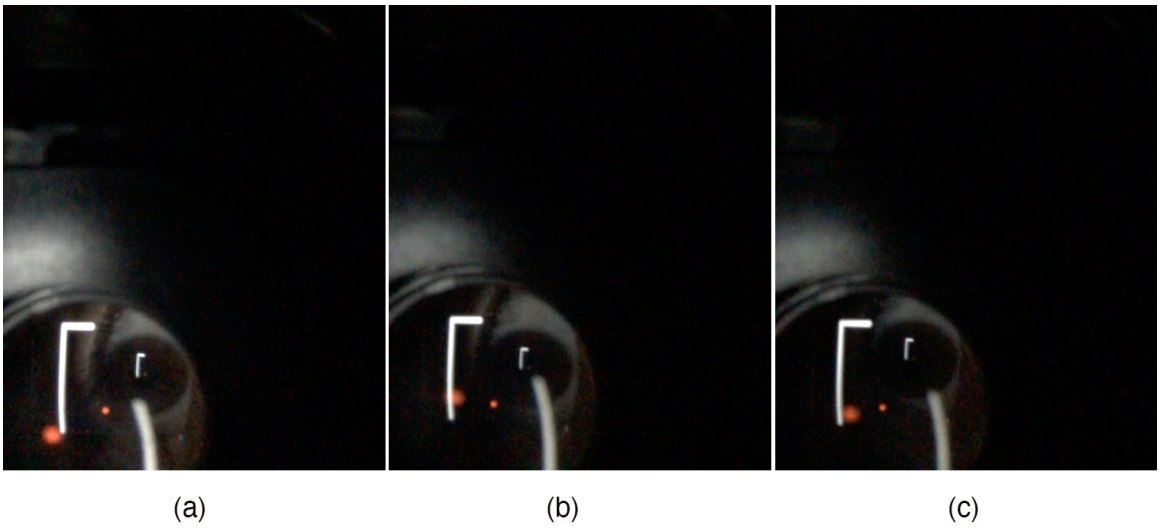


Figure 19: Effect of screen brightness in eye saliency when on-screen content is dark. (a) 100% brightness (550 nits), (b) 80% brightness, (c) 70% brightness. Images taken on iPhone 6S.

within a safe range. This is accomplished by automatically reducing the frequency of the microprocessor on the fly, until the chip cools down. This technique is known as **thermal throttling** or dynamic frequency scaling (figure 21).

Software developers have to keep computational load as low as possible, as thermal throttling causes undesired performance reduction. In consequence, we conducted performance experiments early on to investigate the impact of the most computationally expensive image processing techniques used not only for eye-tracking, but also in a variety of machine vision application. Figure 22 clearly shows that in order to attain real-time eye-tracking, frame size must be as low as possible. Therefore, we apply custom techniques to reduce the image size, as presented in the following section.

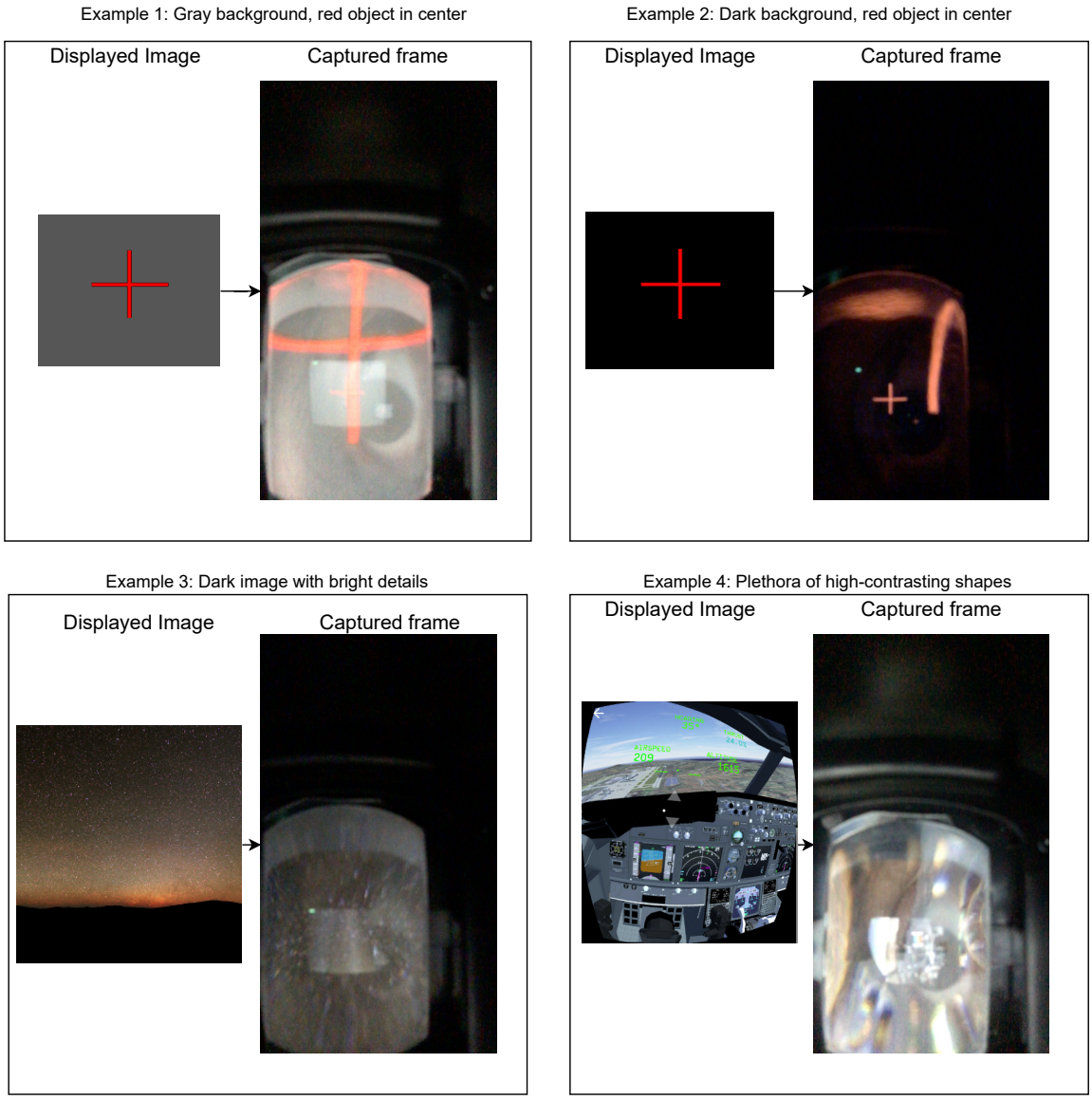


Figure 20: Examples of eye visibility under different displayed content. The intensity of reflections produced is proportional to the intensity and contrast of the displayed shapes.

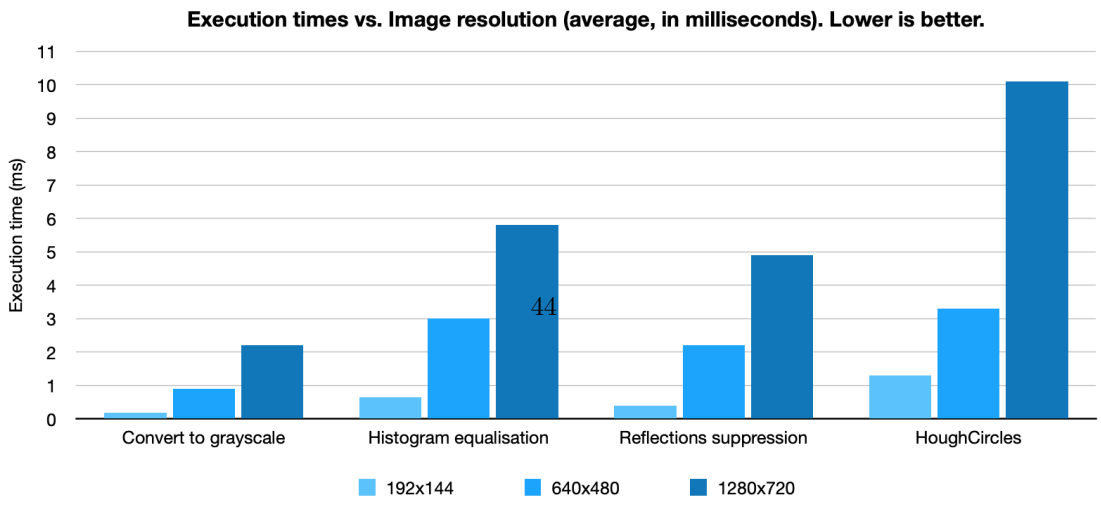
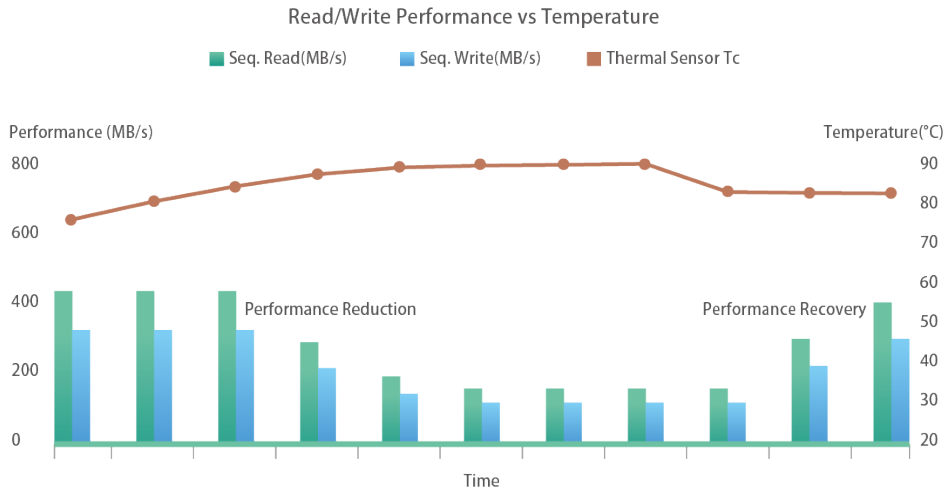
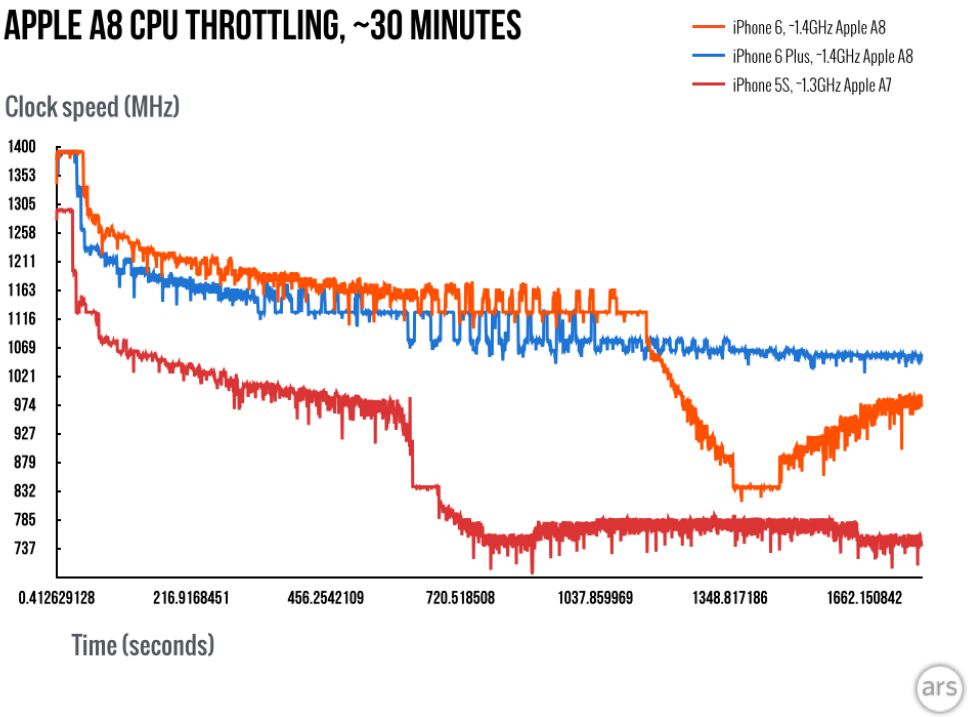


Figure 22: Performance benchmark of computationally expensive image processing functions in correlation to different image sizes. Measurements taken using OpenCV's image processing library on iPhone 6S.



((a)) Illustration of an SSD's performance reduction during thermal throttling (source: https://industrial.apacer.com/)



((b)) Apple's A8 chip thermal throttling. Performance is gradually reduced under a sustained CPU load. (source: https://arstechnica.com/)

Figure 21: Thermal throttling examples.

3.2 System overview

We now describe the main system components of our mobile VR eye tracking system in order to address the challenges analyzed above (Figure 40).

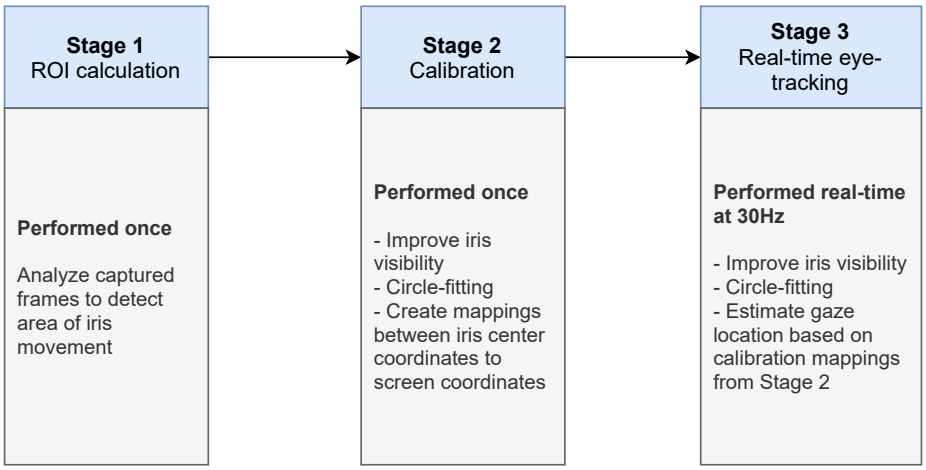


Figure 23: Overview of the 3 stages incorporated in our system.

The system operates in three distinct stages:

Stage 1 - Iris ROI detection: We detect the approximate region in which the eye moves within the captured images to reduce the search space in the following stages. This is accomplished by comparing a set of frames captured by the front camera with each other, as a means to detect movement. The Region of Interest (ROI) is calculated once, before calibration (Figure 24b, resulting image 24c) (Section 4.3).

Stage 2 - Calibration and gaze mapping: We proceed with system calibration. We crop the captured images to isolate the iris ROI region, determined by the first stage. We improve iris visibility by enhancing the contrast of the iris and suppressing the intensity of bright highlights (Figure 1c, resulting image 1d). Circle fitting occurs now on the ROI region, as determined in the first stage, not on the whole image. We calculate the mapping functions, which are used to convert the detected iris center positions to actual screen coordinates (Section 4.4).

Stage 3 - Real-time iris tracking: Now the system is ready to conduct iris tracking utilizing new frames of the eye as captured by the front camera. We re-apply the iris visibility enhancements similarly to the previous stage, to the cropped images which include the ROI region determined in

the first stage and re-apply circle fitting to determine the iris centre. Using the mapping function, we determine the final gaze point on the screen, resulting to real-time iris tracking (Section 4.5).

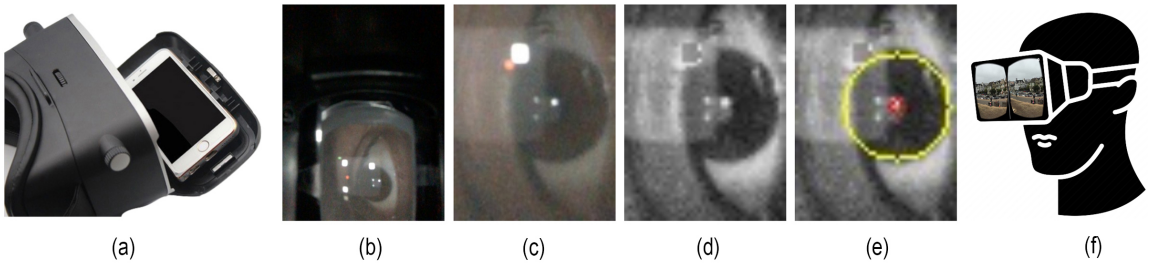


Figure 24: Our system's processing steps, presented in execution order from the left to right: (a) Unmodified VR headset with smartphone (b) Original image as captured from the front-facing camera with visible reflections on headset's lens and eye cornea (c) Cropped RoI (d) Enhanced image (e) Iris contour (yellow) and center (red) estimation (f) User interacts with the gaze-aware VR environment.

In the following section we describe each stage in detail.

4 Implementation

4.1 Hardware and Software used

4.1.1 Our setup

The eye-tracking and gaze-mapping software presented in this thesis was developed entirely on an Apple Macbook Pro (2015 model), powered by an Intel Core i7 CPU and 16GB of RAM, alongside with an iPhone 6S (released in 2015) and iPhone XS (released in 2018) as our main mobile VR eye-tracking test devices. During development, the Macbook Pro was running macOS Mojave and the iPhones were on iOS 13. We opted for Apple devices as Apple's ecosystem allows for seamless communication and deployment across multiple iOS devices. We assume that a similar implementation pipeline would be applicable for Android-based development.

The reason we chose a significantly older generation phone as one of the main mobile VR devices was to prove the robustness and computationally efficiency of our gaze-mapping software, by demonstrating high levels of gaze-mapping accuracy while running on less powerful hardware with an inferior camera sensor. Furthermore, the usage of more than one devices confirms that our system can adapt to different hardware setups, with different screen size, resolution and camera position.

Regarding the VR headsets, we tested our system with two commodity headsets with head straps, available for under $20 in local computer stores. Our study shows that any mobile VR headset should be compatible, as long as the eye remains clearly visible to the front-facing camera through the headset's lens (Figure 18).

Figure 25: Our two commodity headsets used for development and testing.

4.1.2 Software Tools

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial or academic projects [https://opencv.org/about/]. The library includes over 2500 optimized algorithms, allowing not only for basic image processing, but also more sophisticated operations such as face recognition, object identification and classifications.

OpenCV can be downloaded and installed on Mac computers as a standalone package, accessible via terminal commands. It can also be integrated into an Xcode project as a framework, allowing our iOS application to gain access to the library's algorithms.

Xcode is an integrated development environment (IDE) for macOS containing a suite of software development tools developed by Apple for developing software for macOS, iOS, iPadOS, watchOS, and tvOS. It was first released in 2003. Xcode includes Command Line Tools (CLT), which enable UNIX-style development via the Terminal app in macOS.

Xcode supports source code for a variety programming languages, including C, C++, Objective-C and Swift. We utilized Xcode's iOS SDK to compile and debug our iOS applications that run on iPhone's ARM architecture processors.

Swift is a general-purpose, object-oriented, compiled programming language developed by Apple Inc. and the open-source community. It was introduced at Apple's 2014 Worldwide Developers Conference (WWDC) with version 1.0. Swift was developed as a replacement to the out-of-date Objective-C. Prior to Swift, iOS developers were required to used the old school Objective-C, which remains unchanged since the 1980s, posing difficulties and lacking features associated with modern programming languages. Swift's main advantages are its simplistic, expressive and appealing syntax, as well as its speed and built-in safety mechanisms, making it easier to avoid programming mistakes. Swift

can be used in conjunction with other programming languages, as it uses the Objective-C runtime library. This allows C, Objective-C, C++ and Swift code to run within one program. At the writing of this thesis, Swift's latest version is 5.3.

Swift's key features [23] include:

- Type Inference: Swift can explicitly infer the data type of a variable, by using the initial value the developer has assigned. For example, in the expression `var i = 10`, i is Int type as integer value has been assigned to it.
- Typesafe: Prevents passing different data type value to a different data type variable. The compiler catches the exception in compile time if such conflict is found.
- Optional data type: The optional data type is used when a variable has not been assigned a value, intentionally or unintentionally. Consequently, an optional represents two possibilities: Either there is a value (so we can *unrwrap* the optional to access that value) or there is no value at all. Optionals exist to prevent errors caused by accessing nil variables.

4.2 Programming our eye-tracker in XCode

We start off by creating a new empty XCode project and importing the XCode framework. We are going to use the Swift language to program the functionality of our eye-tracking application, due to the advantages described in X. However, OpenCV's library is written in C++, so we can't access the open source computer vision library's functions directly from Swift. Fortunately, due to Swift's great interoperability with Objective-C we will use Objective-C for bridging Swift with C++. We note that Objective-C++ is essentially Objective-C with the ability to link with C++ code.

To this end, we create a new Objective-C class, "OpenCVWrapper" along with a "bridging header". The bridging header is necessary to declare the Objective-C functions which will be visible in Swift, while the Objective-C implementation file contains the actual image processing code. In order to use C++ inside Objective-C, we have to change the file extension from ".m" to ".mm". This lets Xcode know that the file contains Objective-C++ code. In the bridging header .h file we declare the functions implemented in the .mm file.

To access our Objective-C++ image processing functions, we create an instance of OpenCVWrapper in our Swift file and call the functions on the created instance, e.g. OpenCVWrapper.ProcessImage().

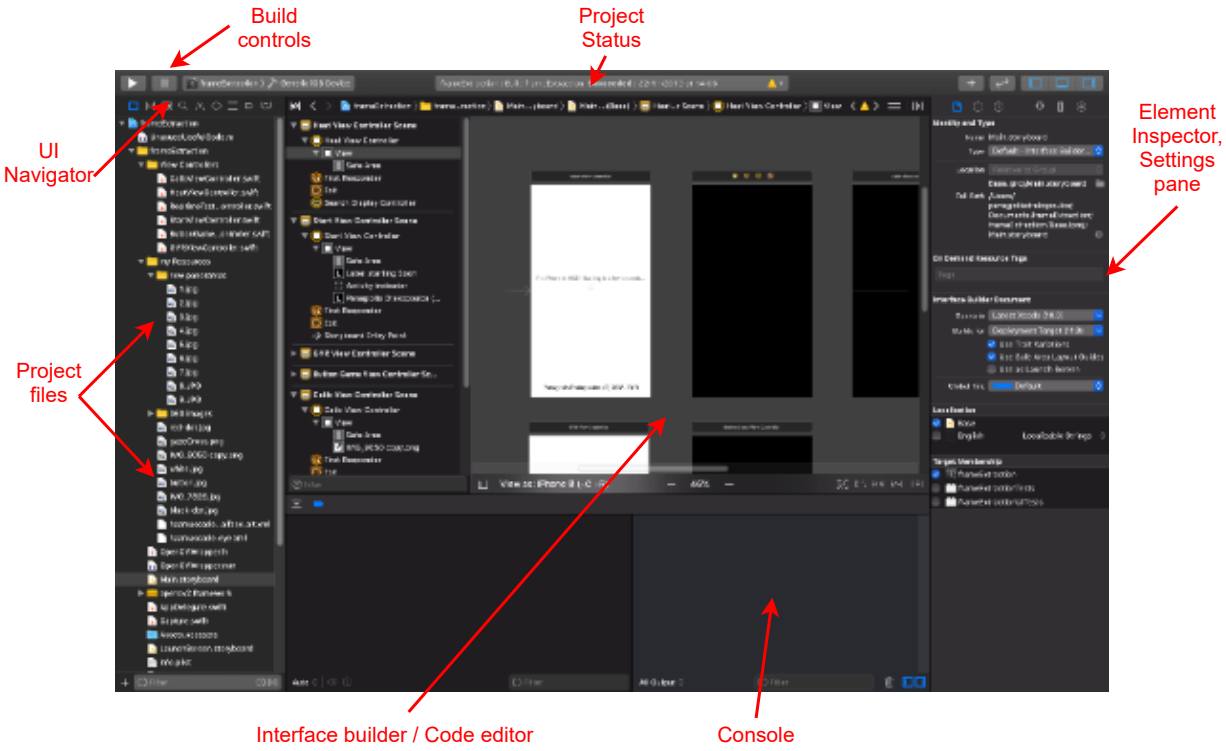


Figure 26: XCode, Apple's native IDE for developing iOS applications.

4.2.1 Initializing real-time capture

In order to capture the iPhone's camera stream and manipulate images, we need to import the UIKit, AVKit and AVFoundation frameworks in our swift file:

```
import UIKit
import AVKit
import AVFoundation
```

To perform real-time capture, we instantiate an `AVCaptureSession` object, `captureSession`. We can now change the parameters of real-time capture to fit our requirements, set its input, output and frame resolution. In our case, we set the input capture device to the front-facing camera and **frame resolution** to **480x360**. Our lab experiments showed that increasing the image quality/resolution can facilitate improved gaze estimation accuracy, however such improvements are minor and were not worth the additional CPU workload. We found that 360p is a balanced setting considering accuracy and performance factors.

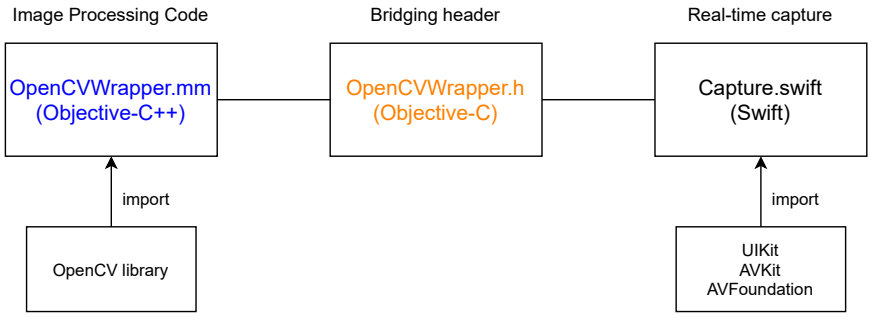


Figure 27: XCode project structure. Image processing operations are performed in OpenCVWrapper.mm using OpenCV's Objective-C++ library. The main code of the application is included in the Capture.swift code file. Communication between Swift and Objective-C++ is performed via a bridging header, which exposes our custom Objective-C++ image processing functions to Swift.

```
captureSession = AVCaptureSession() //initialize AVCaptureSession
captureSession.sessionPreset = .medium  //frame quality/resolution
let captureDevice = AVCaptureDevice.default(.builtInWideAngleCamera, for: .video,
position: .front) //set front-facing camera as capture device
guard let input = try? AVCaptureDeviceInput(device: captureDevice!)
else { return }
captureSession.addInput(input)
```

For the output, we create a new `AVCaptureVideoDataOutput` object and link it to the `captureSession` object, as such:

```
let videoOutput = AVCaptureVideoDataOutput()
    if (captureSession.canAddOutput(videoOutput))
        captureSession.addOutput(videoOutput)
```

We configure the videoOutput object so that frames captured while the dispatch queue is already handling an existing frame are discarded. This helps reduce memory usage by dropping late frames.

```
videoOutput.alwaysDiscardsLateVideoFrames = true
videoOutput.setSampleBufferDelegate(self, queue: DispatchQueue.main)
```

Finally, we initialize the connection of the videoOutput object, apply the aforementioned configuration to the `captureSession` object and start the real-time capture.

```
let connection = videoOutput.connection(with: AVFoundation.AVMediaType.video)
            connection?.videoOrientation = AVCaptureVideoOrientation.portrait
```

```
captureSession.commitConfiguration()
captureSession.startRunning()
```

The presented code is executed only once, at the launch of our eye-tracking application.

4.2.2 Acquiring captured frames

Upon real-time capture initialization, our system is ready to receive frames as captured in real-time from the phone's front-facing camera. To this end, we construct a callback function, `captureOutput`, which is called on the dispatch queue specified by the output's `sampleBufferCallbackQueue` property. This function is called repeatedly, so it must be efficient to prevent performance problems. Ideally, this function should be called 30 times per second, as the iPhone's front-facing camera processess frames at a rate of 30fps. However this might not always be the case, depending on the CPU's workload.

```
func captureOutput(_ output: AVCaptureOutput,
                       didOutput sampleBuffer: CMSampleBuffer,
                       from connection: AVCaptureConnection) {
    //print("Got a new frame!")
```

Once the `captureOutput` function is called, iOS stores the captured frame data in `CVPixelBuffer`, a pixel buffer in main memory. We then convert the pixel buffer data to a usable image format (UIImage) to allow for image processing operations. This is performed as follows:

```
    //print("Got a new frame!")
    guard let pixelBuffer: CVPixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer)
     else {
        return
    }
    //Convert CIImage to UIImage
    let ciimage : CIImage = CIImage(cvPixelBuffer: pixelBuffer)
    lastImage = self.convertToUIImage(cmage: ciimage)
```

CIImage is an iOS data type which contains image data, but it is not directly readable. OpenCV requires input images to be in UIImage format. To this end, we convert the frame from CIImage to UIImage format, as presented in the last two lines of above code snippet.

The described process can be represented by flowchart in fig. 8.

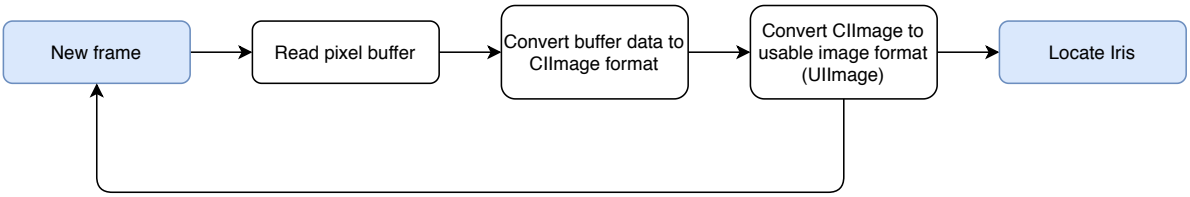


Figure 28: Flowchart depicting the processing steps upon receiving a new frame during real-time capture.

4.3 Stage 1: Iris ROI detection

The original images, as captured from the front-facing camera of the mobile phone contain irrelevant information, such as of the plastic of the VR headset surrounding the lenses (Figure 1b). Only a small section of the captured image is useful for the detection of the iris (Figure 1c). To this end, the size of the images can be reduced by cropping the Region of Interest (ROI), which corresponds to the eye and its surrounding area. This operation has a direct effect on the iris detection speed, as the algorithm has to search a substantially smaller area to locate the iris. Consequently, less system resources are used, making real-time eye tracking possible. This stage can be divided in two distinct phases for easier presentation.

4.3.1 Phase 1: Obtain frames from front-facing camera

To calculate the ROI, the user is required to adjust their gaze at a set of 12 target points uniformly spread across the entire screen, presented one at a time for $t = 1s$ (Figure 29). While the user is fixating on a $i \in [1, n]$ target point, we capture a f_i frame through the front-facing camera. During this phase, precise fixation on each target is not of big importance, as we only want to detect eye movement. To illuminate the eye, we display two thin bezels of white color along the top edges of the screen. Their reflection is visible in Figure 30.

By the end of the procedure, 12 frames have been collected, each corresponding to a different target point, and thus different iris position within the image. (Figure 30). The process takes 12 seconds to complete.

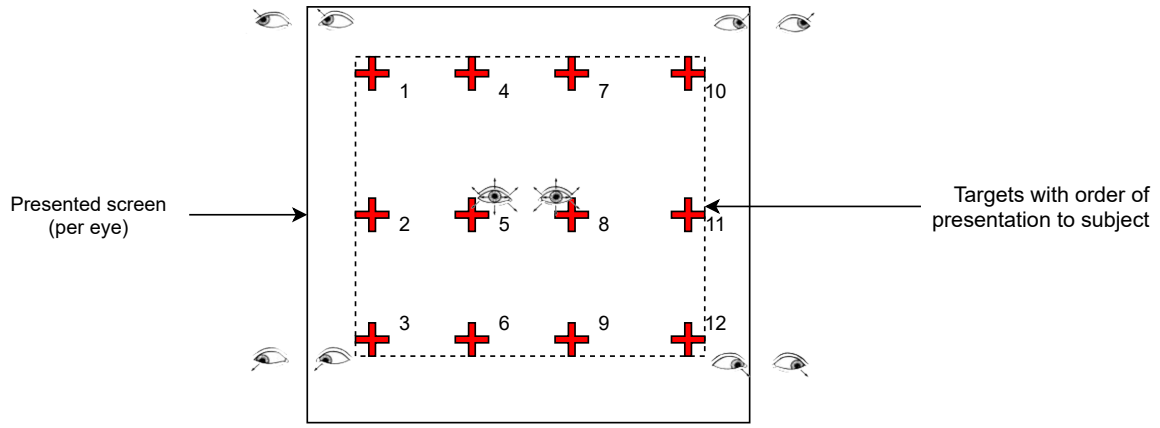


Figure 29: Displayed screen illustration, showing target positions and presentation order. It should be noted that since ROI detection occurs while the subject is wearing the head mounted display, the calibration screen is rendered twice, one for each eye.

4.3.2 Phase 2: Analyze frames and calculate ROI

To determine the rectangle in which the eye moves, we run a **subtraction** operation on the collected images, as a means to detect eye movement through pixel intensity differences. The uniformly distributed targets ensure that the calculated rectangle coordinates enclose all possible eye positions, as the user looks around from left to right and top to bottom. This validation is important, as eye-tracking would fail in case the iris is mistakenly cropped out of the preserved ROI. Through this process, we create a **heatmap**, which essentially depicts the change frequency of each pixel (Figure 30). We are interested in the pixels towards the red end of the color spectrum, representing high change frequency as the eye moves.

In order to compare all frames between them without repetition, we need to perform 66 subtraction operations, as the combination formula indicates. More specifically, we have $n = 12$ frames in our collection and we compare two at a time, so $k = 2$. By solving the formula, we obtain:

$$C_k(n) = \binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{12!}{2!(12-2)!} = 66 \tag{4}$$

Alternatively, our lab tests showed that we do not need to make that many operations, as comparing only a fraction of the frames is sufficient to compute a heatmap image that accurately depicts eye movement areas. Thus, we only make 12 comparisons to spare time and computational resources.

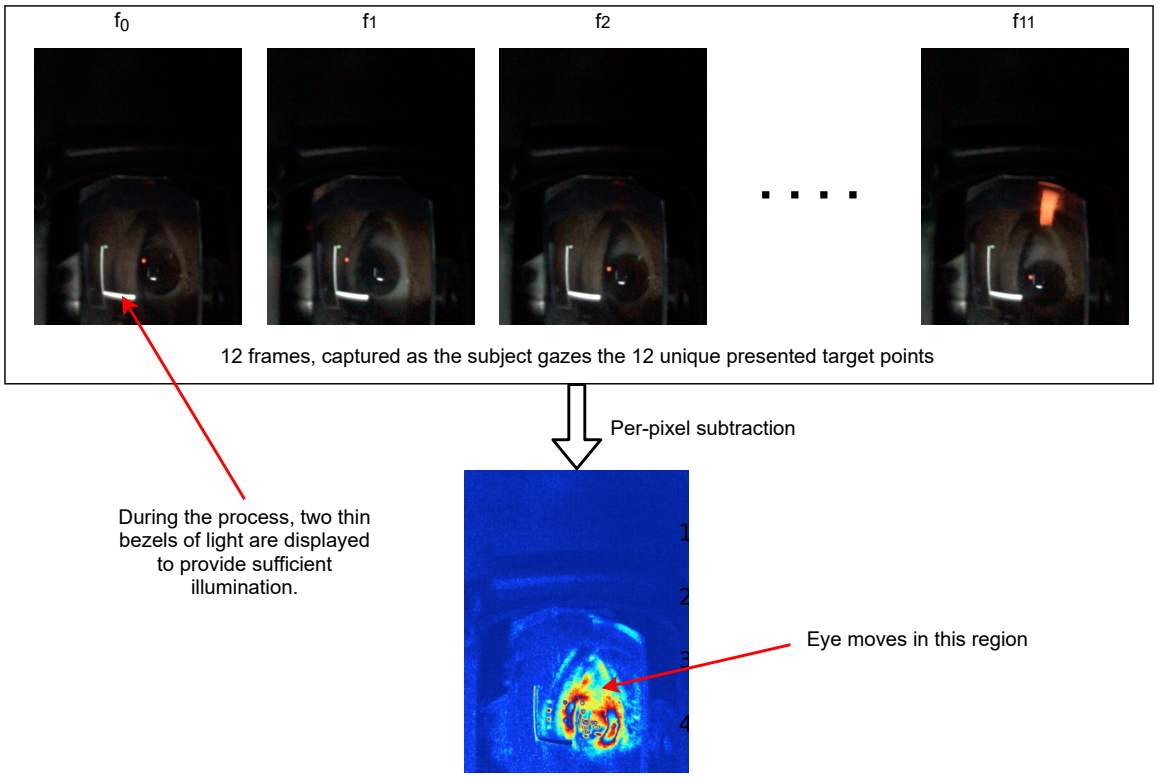


Figure 30: Calculating the eye-movement heatmap, as the subject gazes 12 presented points, distributed on the screen. Areas towards the red end of the spectrum represent movement. Areas of blue color remain exactly the same, regardless of where the user looks.

Comparison/subtraction between two frames produces a new image, in which high intensity pixels represent large differences between the frames and vice versa. The pixel values of this image are added in an *accumulator* image, which as its name suggests, accumulates the pixel differences of all comparisons. It is noteworthy that due to the fact that pixel values must be in the 0-255 range, we check that the addition's result is lower or equal to 255. In case it is higher, we set it to 255, otherwise the image will end up corrupted. Pixels with intensity of 255 are illustrated as red in the heatmap image, where as pixels with intensity of zero are displayed as blue.

```
// the below operation is performed to calculate pixel differences between images X1
    and X2.
// X1 and X2 are the compared frames
// the accumulator represents the heatmap image at the end of the oparation.
for (int row = 0; row < rows; row++)
{
```

```
    for (int col = 0; col < cols; col++)
    {
        int difference = abs(imgX1.at<uchar>(row,col) - imgX2.at<uchar>(row,col));

        // pixel values can't be higher than 255
        if (accumulator.at<uchar>(row,col) + difference < 255)
            accumulator.at<uchar>(row,col) += difference;
        else
            accumulator.at<uchar>(row,col) = 255;
    }
}
```

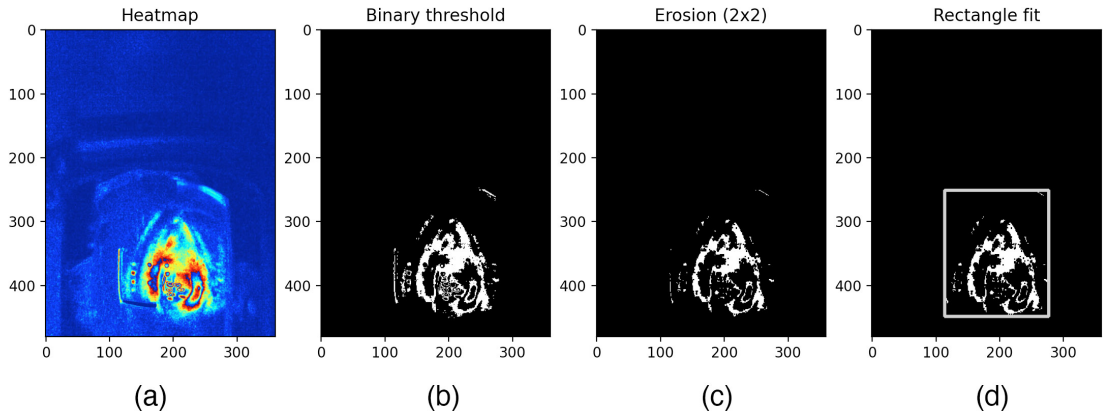


Figure 31: Calculating the ROI rectangle using the eye-movement heatmap.

The size and position of the crop rectangle is dynamically determined per user as to safely encapsulate all possible iris positions, represented by the highest intensity areas of the heatmap. To do so, nonessential areas of low change frequency are removed by applying **binary thresholding** to the heatmap (Figure 31b) followed by a morphological operation of **erosion** (Figure 31c) to remove leftover noise. We then fit a bounding rectangle to the remaining non-zero pixels (Figure 31d). The resulting rectangle represents the ROI cropping window which will be used in every subsequent step to reduce the image's size and thus accelerate iris detection. This allows for a drastic decrease in the input's size, by almost 8 times, greatly contributing to the reduction of CPU usage, thus, permitting real-time eye tracking on a mobile device.

```
threshold(accumulator,accumulator,177,255, CV_THRESH_BINARY); // binary thresholding
```

```
erode(accumulator,accumulator,kernel); // we use a 2x2 kernel for erosion
```

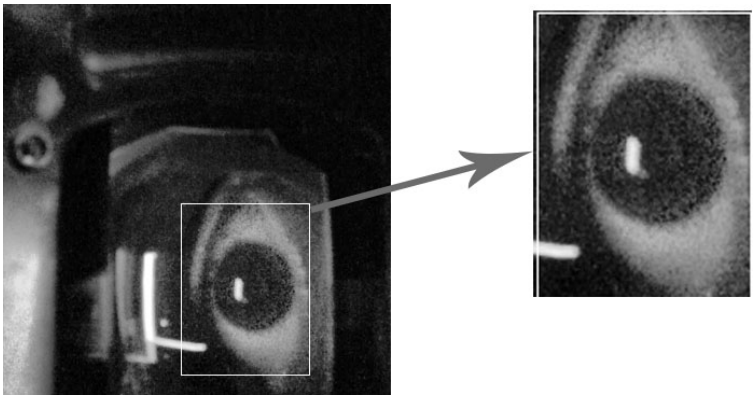


Figure 32: ROI detection provides as input to the next process a subsection of the captured images, representing the detected ROI rectangle. This step is vital to accomplish real-time eye tracking on a mobile device, as our iris-tracking algorithm has to process a significantly smaller area in all the subsequent steps.

Algorithm 1: Stage 1 pseudocode

Input: Raw eye images stream from front-facing camera
Output: ROI coordinates

for $i = 0; i < N; i++$ **do**
    Display n_i target point;
    Capture frame f_i;
    Wait 1500ms ;
end
for $k = 0; k < N; k++$ **do**
    for $j = N - 1; j == 0; j--$ **do**
        difference $= f_k - f_j$;
        accumulator += difference;
    end
end
Threshold(accumulator);
Erode(accumulator);
Fit rectangle to non-zero pixels of accumulator ;
ROI = rectangle coordinates;
Proceed to Stage 2;

4.4 Stage 2: Calibration and gaze mapping

Initially, images are captured by the smart phone's camera of head-worn mobile VR, at 30 fps. Directly applying circle fitting does not accurately detect the iris center, as the images suffer from poor contrast and obtrusive reflections. We apply a series of image enhancement operations that result in a more salient iris. We perform: image cropping, down-scaling and down-sampling; reflections suppression; histogram equalization.

4.4.1 Image cropping and resizing

Our system operates on gray scale images reducing computational cost without reduction of the visual information required. Image cropping removes unnecessary peripheral context from the image (plastic case, lenses, etc.) and isolates the eye's Region of Interest (ROI). The cropping is based on the ROI calculated from Stage 1, as described in Section 4.3. Image resize operations reduce the image size, consequently accelerating the subsequent steps of reflection suppression, histogram equalization and circle-fitting, due to the smaller search space.

OpenCV's iOS SDK is implemented in Objective-C. Consequently, image processing functions for locating the iris had to be written in Objective-C instead of Swift.

As mentioned in the previous section, OpenCV's iOS SDK recognizes `UIImage` format images. However in order to apply OpenCV image processing functions, we need to convert the image to OpenCV's native image format, called `Mat`. `Mat` is the main matrix class in OpenCV and is contained in the OpenCV namespace `cv`. Given that UIimg is the captured frame in `UIImage` format, we convert it to a `Mat` object, as such:

```
+(UIImage*) ProcessFrame: (UIImage *)UIimg
{
    Mat img;
    UIImageToMat(UIimg, img);
    ...
```

Next, we crop the image to the ROI (calculated in...) in order to reduce its size and remove areas of no use. The `CV_BGR2GRAY` flag instructs `cvtColor()` to convert the image to grayscale. There is a variety of flags for conversions to other color spaces. Converting to grayscale is necessary before applying the next image processing operations, as we will only have to work on one channel instead

of three. Furthermore, color does not provide any significant detail in our case. We also downscale the image to 35% of its original size to further reduce CPU workload of subsequent of operations.

```
...
img = img(eyeROI_rect);   // crop
cvtColor(img, img, CV_BGR2GRAY);    // convert to grayscale
resize(img, img, cv::Size(), 0.35, 0.35);  // downscale
```

4.4.2 Reflection Suppression

The eye may be occluded by reflections on the HMD lens, and thus, iris detection is hindered. Although removing reflections is not trivial, salient highlights can be detected as an abrupt peak in high intensity values in the histogram. The intensity of pixels with values higher than a threshold are suppressed, by being averaged with the neighbouring pixels of lower intensity. These neighboring pixels have values lower than the threshold and are not considered to be part of the reflection. The threshold is determined in real-time per frame.

We perform reflection suppression to reduce circle fitting errors caused by reflections obstructing the iris. Firstly, we compute an intensity threshold value over which a pixel will be considered part of a reflection and will be suppressed. To accomplish this, we check the image's histogram for an abrupt peak in high intensity values, and set the threshold value equal to intensity value in which the peak begins. If such peak is detected, it is most likely that bright highlights are present in the image. We then feed the resulting threshold value to a Canny edge detection function, to extract the contours of the reflections present in the image. We then fill the enclosed area by thickening the detected contours. This results to a binary mask, where white pixels represent reflection areas to be suppressed. Masking is frequently used in image processing algorithms to restrict a point or arithmetic operator to an area defined by the mask. In our specific case, the created mask is used on the original image to suppress high intensity pixels, by averaging each pixel that corresponds to the white area of the mask with neighbouring pixels that do not belong to the white area. To counteract the enlarged footprint caused by thickening, we apply morphological erosion with an equal kernel, as shown in figure 33.

Once the binary mask is calculated, we can suppress high intensity values, by averaging each pixel that exceeds the threshold with neighbour pixels. This is performed by applying a 16x16 size

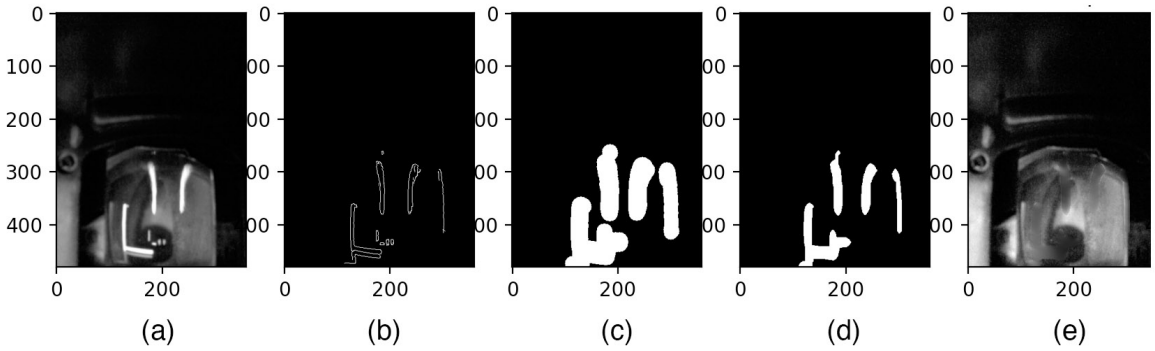


Figure 33: Our reflection suppression pipeline. (a) Original camera frame, (b) Canny edge detection, (c) Contour thickening to fill enclosed areas, (d) morphological erosion to counteract the expanded footprint caused by thickening, (e) reflection suppression by averaging high intensity pixels defined by the binary mask created in step d.

window. Averaging windows of similar size may also be applicable.

```
    + (cv::Mat) RemoveReflections: (cv::Mat) img {
    int kernelSize = 16;
    for (int row = 0; row < img.rows; row++)
    {
        for (int col = 0; col < img.cols; col++)
        {
            if (img.at<uchar>(row,col) > 240) {
                cv::Rect roi;
                roi.x = [self Clamp:col-kernelSize/2:0:img.cols];
                roi.y = [self Clamp:row-kernelSize/2:0:img.rows];
                roi.width = [self Clamp:kernelSize:0:img.cols - roi.x];
                roi.height = [self Clamp:kernelSize:0:img.rows - roi.y];
                img.at<uchar>(row,col) = cv::mean(img(roi))[0];
            }
        }
    }
    return img;
}
```

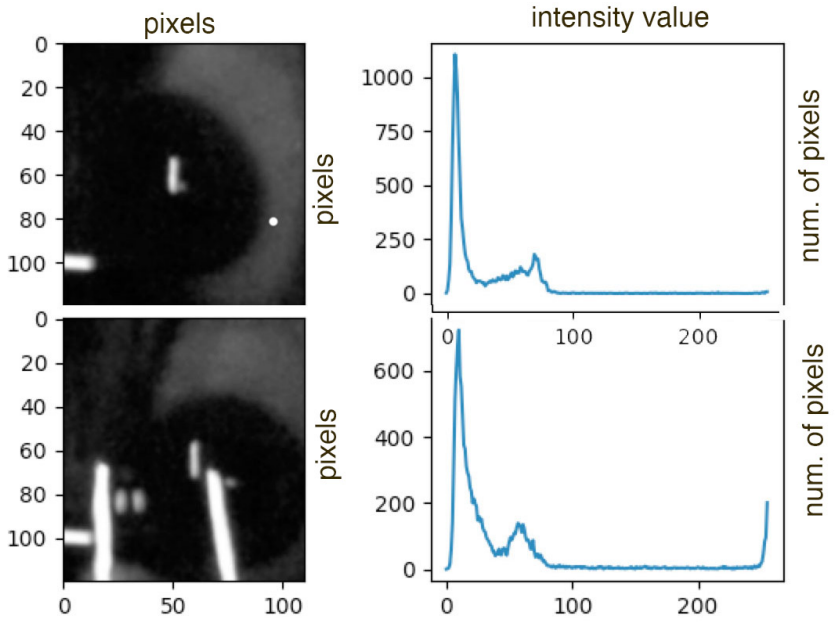


Figure 34: Image histogram with: no lens reflections (top), strong lens reflections (bottom) with a visible peak of high intensity values.

4.4.3 Histogram equalization

This operation is commonly performed in image processing to enhance the contrast of an image. Captured eye images from within the headset suffer from poor contrast due to insufficient illumination. The intensities of such images are confined in a small range of values. Histogram equalization expands low contrast image areas by spreading out their most frequent intensity values to cover the entire available dynamic range, resulting in a more uniform intensity distribution. The outcome of the operation is a more salient iris (Figure 1d).

We experimented with the standard histogram equalization method, as well as a more sophisticated variant, known as Contrast Limited Adaptive Histogram Equalization (CLAHE) [58]. Lab tests showed improved iris saliency compared to the ordinary histogram equalization (see Figure 35). This can be attributed to the fact that CLAHE computes several histograms, each corresponding to a distinct image section; it is therefore suitable for locally enhancing the definitions of edges in each region of the image. We apply contrast limited adaptive histogram equalization (CLAHE) by using a 4x4 size kernel. The larger the kernel, the less aggressive the local contrast enhancement

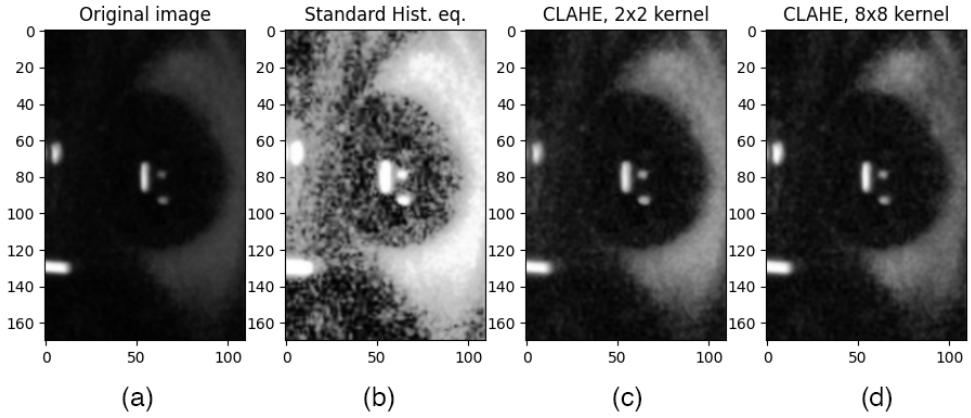


Figure 35: Comparison of the most commonly used histogram equalization methods, applied to our use case scenario with poor eye visibility. We can clearly observe that in our case CLAHE provides superior results compared to the standard histogram equalization method.

.

(see Figure 35c, d).

```
cv::Ptr<cv::CLAHE> clahe = cv::createCLAHE(3, cv::Size(4,4));
clahe->apply(img, img);
```

4.4.4 Circle fitting - Hough gradient method

The image is now ready to undergo our circle fitting operation, a customized version of the Hough Transform. We chose circle fitting over ellipse fitting due to the headset's close proximity to the eye. Thus, the eye's eccentricity is expected to be close to zero (i.e. a circle) at all times. Captured images vary in brightness, reflections and iris saliency. Thus, we customised the Hough gradient function, a cost-efficient version of the Hough transform [52], to be highly sensitive to circular features in the image so that circle fitting is successful under all illumination conditions.

The original Hough transformation was found to be computationally inefficient when it comes to circle detection [22, 52], therefore, we developed a customized accelerated Hough transformation which is highly sensitive to circular features such as the iris. The Hough gradient method utilizes the slope information of edges, by calculating the Sobel derivatives [15] in the background. Using this gradient, every point along an imaginary line indicated by the slope, within a specified minimum and maximum distance, is incremented in the accumulator as a potential center. A center is kept

if it has sufficient support from the nonzero pixels in the edge image, also referred to as “votes” in bibliography, and if it is a sufficient distance from previous centers [5]. This helps reduce the computational cost of the problem, by requiring a 2D accumulator, e.g., a data structure constructed by the algorithm to obtain the object candidates as local maximas, as opposed to a 3D accumulator required by the classic Hough transform method.

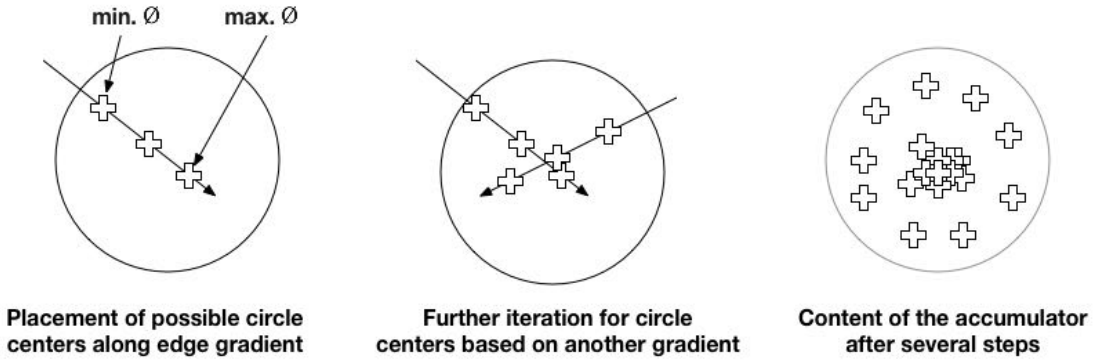


Figure 36: Identifying circle centers using the Hough Gradient method. We can see that at the end of all iterations, the center of the circle has collected the most "votes", as expected. Image source: [27].

The Hough Gradient method requires several parameters as input [**?**], as implemented by the OpenCV framework, explained further below :

```
cv.HoughCircles (image, circles, method, dp, minDist, param1, param2, minRadius,
    maxRadius)
```

- **image**: 8-bit, single-channel, grayscale input image.
- **circles**: output vector of found circles. Each vector is encoded as a 3-element floating-point vector (x,y,radius).
- **method**: detection method flag. (e.g. HOUGH_GRADIENT).
- **dp**: inverse ratio of the accumulator resolution to the image resolution. For example, if dp = 1 , the accumulator has the same resolution as the input image. If dp = 2 , the accumulator has half as big width and height.

- **minDist**: minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
- **param1**: first method-specific parameter. In case of HOUGH_GRADIENT , it is the higher threshold of the two passed to the Canny edge detector.
- **param2**: second method-specific parameter. In case of HOUGH_GRADIENT , it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first.
- **minRadius**: minimum circle radius (in pixels).
- **maxRadius**: maximum circle radius (in pixels).

We create a vector variable to accommodate the detected circles, set the method parameter flag to `HOUGH_GRADIENT`, $dp = 1$ and $minDist$ equal to 7% of the image's width. The aforementioned parameter values were selected upon conducting lab tests to determine the effect of each parameter, not only on its own, but also in combination with the rest of the parameters. The aforementioned values yielded the best accuracy results in our tests; similar values may be applicable.

To correctly set the $minRadius$ and $maxRadius$ parameters, knowledge of the expected minimum and maximum iris radius in pixels is required. To this end, we conducted a simplistic lab experiment by recruiting the students of our University's lab (5 male, 2 female). The participants were required to wear the head mounted display with the smartphone attached as a VR display, until we manualy recorded their iris radius in pixels, as captured by the front-facing camera. We found that the average iris radius is equal to $r_{iris} = 39$ pixels (in the original-sized image, before downscale). It is noteworthy that this number is subject to change, as it is correlated to the front-facing camera's distance to the eye, as well as the image resolution itself. For example, if the image is scaled by a factor of s, the iris radius will also be scaled by the same factor s. In consequence, we set the $minRadius = r_{iris} - 3$ and $maxRadius = r_{iris} + 3$ to compensate for circle-fitting inaccuracies and various headset positions in relation to the user's eye.

Finally, we experimentally set $param1 = 55$ and $param2 = 9$, so that the function is highly sensitive to circular features. Naturally, this will result in a high number of false positives, which

we deal with in the next steps of our algorithm, selecting the best circle through the *confidence* measurement.

Concluding, we execute the HoughCircles function as such:

```
std::vector<Vec3f> circles;
cv.HoughCircles (image, circles, HOUGH_GRADIENT, 1, 0.07*image.width, 55, 9, (r_iris
    -3)*scale, (r_iris+3)*scale)
```

The accuracy of the derived results in relation to candidate iris circles depends greatly on the given function's parameters and especially the accumulator threshold. The higher the threshold value, the fewer and more accurate candidate iris circles are returned. However, too high a value can lead to detection failure. Our experiments showed that there is no specific optimal accumulator threshold, as captured images from within the headset significantly vary in brightness, visible reflections, iris contrast and saliency. We therefore use a fixed fail-safe threshold which makes the function extremely sensitive to any circular feature. Even in the most challenging scenarios of poor eye saliency, the algorithm will always return $k \geq 1$ candidate iris circles $C_{1..k}$. However, most of these candidates will be false positives. To determine which one of the k candidates is the actual iris, we devise a *confidence* metric that indicates each candidate's probability to match the circumference of an iris, as demonstrated in section 4.4.5.

It should be noted that we experimented with alternative image processing techniques, such as binary thresholding and morphological operations, which did not provide additional image improvements to our challenging lighting scenarios. Consequently, it was shown by our in-lab testing that the presented image processing pipeline constitutes the most optimized combination of operations to handle images with varying lighting, contrast and iris saliency for the purpose of eye-tracking.

4.4.5 Confidence measure

The confidence value is calculated upon two purpose-specific, experimentally validated hypotheses h_1 and h_2, described below. Based on these hypotheses, the algorithm decides whether the candidate circle corresponds to the iris or not. Figure 3 depicts a set of random iris frames and their respective confidence values. The circle with the highest confidence value is selected as the best circle (C_a), with its center $\vec{e_a} = (x_e, y_e)$ representing the estimated iris center position. In cases when none of the candidate circles' confidence value fails to pass an experimentally defined threshold T, it is

assumed that the gaze estimation error is unacceptable and gaze estimation for that frame fails. Consequently, the frame is rejected and the returned gaze gaze position is identical to the latest successful prediction. T was selected as such, to maximize iris detection accuracy in lab tests and is always fixed at $T = 65$. We discuss the potential of a dynamic T threshold in Future work. A balanced T value is essential; too high a value of T could result in rejecting accurate fits, while a too low value of T could result in accepting poor iris fits, thus degrading accuracy. We calculate the confidence value of each circle by taking into account two hypotheses:

- h_1: In dark-pupil eye tracking, where the light source is off-axis with respect to the camera, it is assumed that the darkest region of a captured image corresponds to the iris. Our lab tests showed that this hypothesis is also valid for eye tracking in the visible spectrum, as long bright highlights/reflections do not cover the iris.

- h_2: The iris appears as a darker circular region surrounded by a brighter region, regardless of the illumination level. If the mean intensity of the candidate iris segment is lower than the mean intensity of the surrounding region, the candidate supports this hypothesis. The higher the contrast between the two segments, the higher the probability the candidate iris segment actually corresponds to the iris.

Both hypotheses require that reflections have been previously suppressed or removed. as bright highlights create intensity abnormalities that invalidate these hypotheses, especially if the fall on the iris. We compute the confidence value ω of each candidate circle C_i by multiplying h_1, h_2 with their corresponding weights w_1, w_2:

$$\omega(C_i) = h_1(C_i) \cdot w_1 + h_2(C_i) \cdot w_2 \tag{5}$$

Weights w_1 and w_2 were experimentally determined by investigating the effect of each hypotheses on iris detection accuracy, in lab tests, using eye images with varying eye saliency scenarios as input. It was shown that h_2 is more robust compared to h_1. This can be largely attributed to the fact that h_1 fails in scenarios where bright highlights fall onto the iris, increasing the brightness of the otherwise dark iris and thus invalidating the hypothesis. We set $w_1 = 1$ and $w_2 = 1.9$, although similar value-pairs of similar ratio may also be applicable. The iris center coordinates $\vec{e}$ for any given frame are equal to:

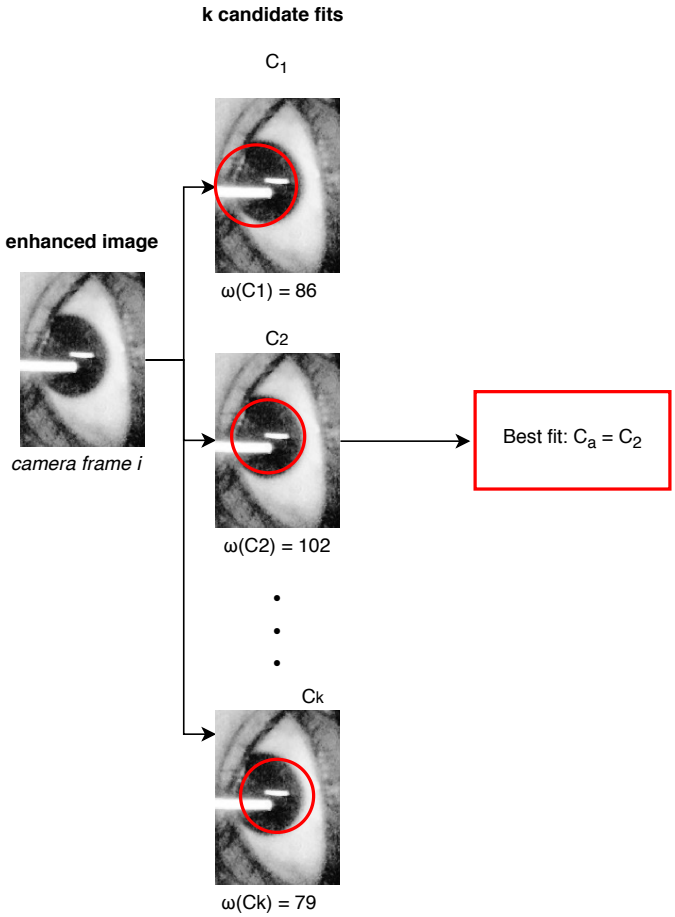


Figure 37: The best circle fit to the iris is the candidate that yields the higher confidence value ω by finding $\max_{i=1..k} \omega(C_i)$.

$$\vec{e} = \begin{cases} \vec{e_a}, & \text{if } \omega(C_a) \geq T \\ \vec{e}_{previous}, & \text{otherwise} \end{cases}, \omega(C_a) = \max_{i=1..k} \omega(C_i) \tag{6}$$

4.4.6 Overview of the calibration procedure

Calculating the point of gaze requires a conversion of the estimated iris center coordinates to locations on the user's screen. The conversion is accomplished by a mapping function, determined through a calibration procedure (Figure 41). To compute the mapping function, the user is required to fixate at a set of $n = 12$ targets in known positions. While the viewer is fixating on each target $i \in [1, n]$ with

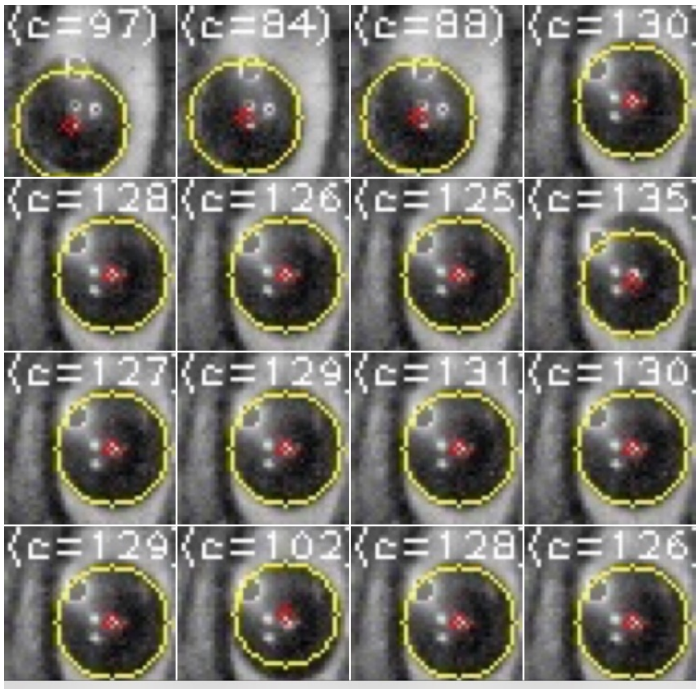


Figure 38: The higher the confidence value ω, the higher the probability the candidate circle corresponds to the iris. Challenging conditions (e.g. obtrusive reflections, eye angle in respect to the camera) yield lower confidence scores.

screen coordinates $\vec{s}_i = (x_s, y_s)$ the iris center position $\vec{e}_i = (x_e, y_e)$ is computed and recorded along with the corresponding target screen coordinates $\vec{s}_i$. To compute the iris center, we apply the image enhancements (Section 4.4) and our circle-fitting technique (Section 4.4.4) to the cropped ROI. By the end of the procedure taking nearly 30 seconds, $n = 12$ pairs of $\{\vec{s}_i, \vec{e}_i\}$ have been obtained.

Algorithm 2: Stage 2 (calibration) pseudocode

```
Input: Raw eye images stream from front-facing camera
Output: Mapping function

i=0;
while i < total number of target points do
    Display n_i target point;
    Capture frame f_i ;
    Crop ROI;
    Downscale;
    Downsample;
    Suppress reflections;
    Apply histogram equalisation (CLAHE);
    Attempt circle-fitting via the Hough Gradient method;
    for each candidate iris circle C_i do
        Calculate confidence ω_i;
    end
    Find max ω_a of ω_i values;
    if ω_a > T and t_i > t_p + t_m then
        Record eye center coordinates;
        i=i+1;
    else
        Keep displaying n_i target point to user;
    end
end
Calculate mapping function coefficients;
Calibration completed;
Proceed to Stage 3;
```

The system's accuracy relies heavily on the precision and quality of the collected data. It is crucial that the viewer is fixating on the presented targets during the entire process. However, the inevitable occurrence of involuntary blinks or saccades can lead to imprecise data collection. To attain the highest data quality possible, we perform a number of validation checks:

1. Eye capturing is paused for $t_p = 500$ ms once a new target gaze point is presented, so that the user has adequate time to react to the stimuli and adjust their gaze accordingly. Related research has shown that saccades take 200ms to initiate and 20-200ms to complete [2]. Consequently, we consider 500ms an adequate time period to ensure that the user will be fixating on each new target when eye capturing and recording is performed.

2. After eye capture resumes, the new target point is presented to the user for at least t_m seconds,

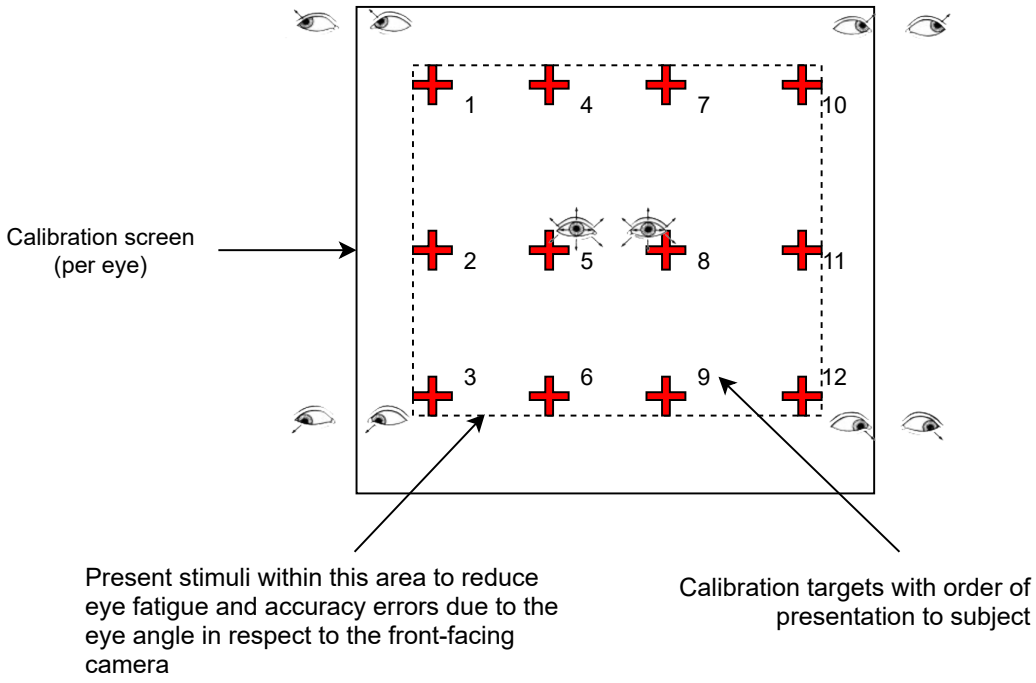


Figure 39: Illustration of the calibration screen, showing target positions, presentation order and the used eye-tracking area (in approximation). It should be noted that since calibration occurs while the subject is wearing the head mounted display, the calibration screen is displayed twice, once for each half of the screen.

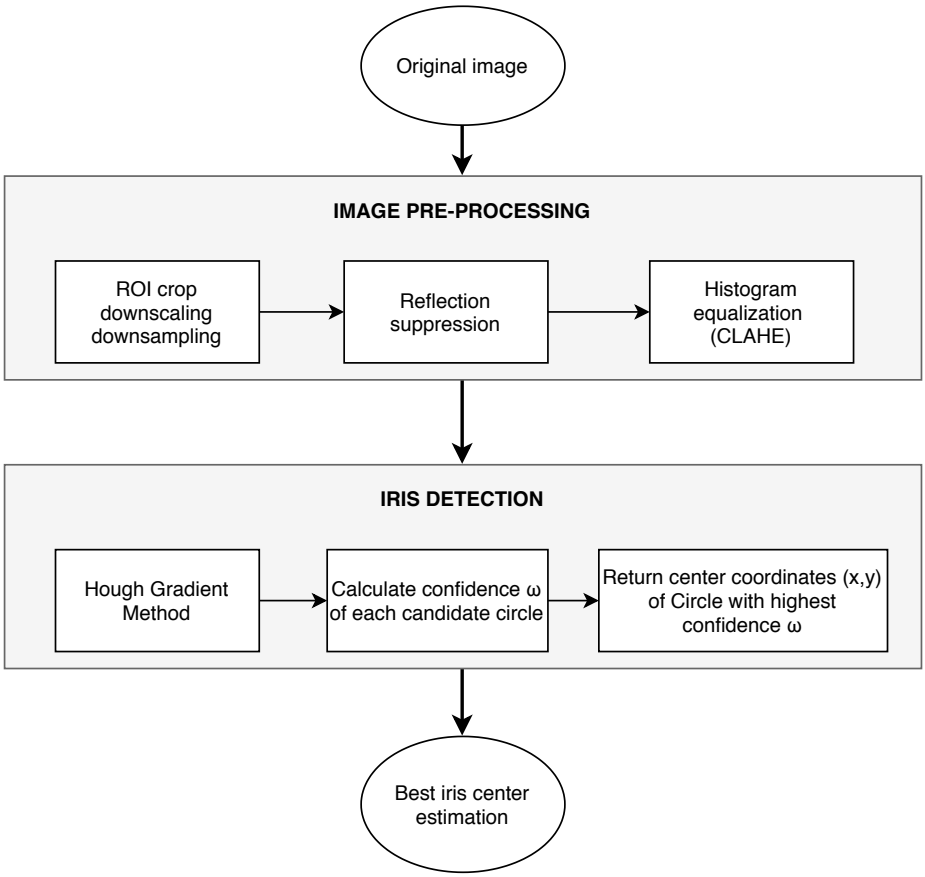


Figure 40: Flowchart depicting the steps involved in processing the original image to calculate the iris center position.

set heuristically to $t_m = 1500$ms. Estimated iris center coordinates of high confidence ($\omega(\vec{e}_i) \geq T$) are recorded, along with the coordinates of the corresponding target point displayed on the screen. Iris center coordinates not meeting the confidence criteria are considered inaccurate and, thus, rejected.

3. The system proceeds to the next target point $i+1$ when $t_i > t_p + t_m$, t_i the total time target point i is displayed to the user. This ensures the user has adequate time to accurately fixate on the new target.

The process is repeated n times until all $\{\vec{s}_i, \vec{e}_i\}$ pairs have been collected for the n target gaze positions. After calibration, we crop the captured images to isolate the iris ROI region, determined by the first stage. We then improve iris visibility. Circle fitting occurs now on the ROI region, as

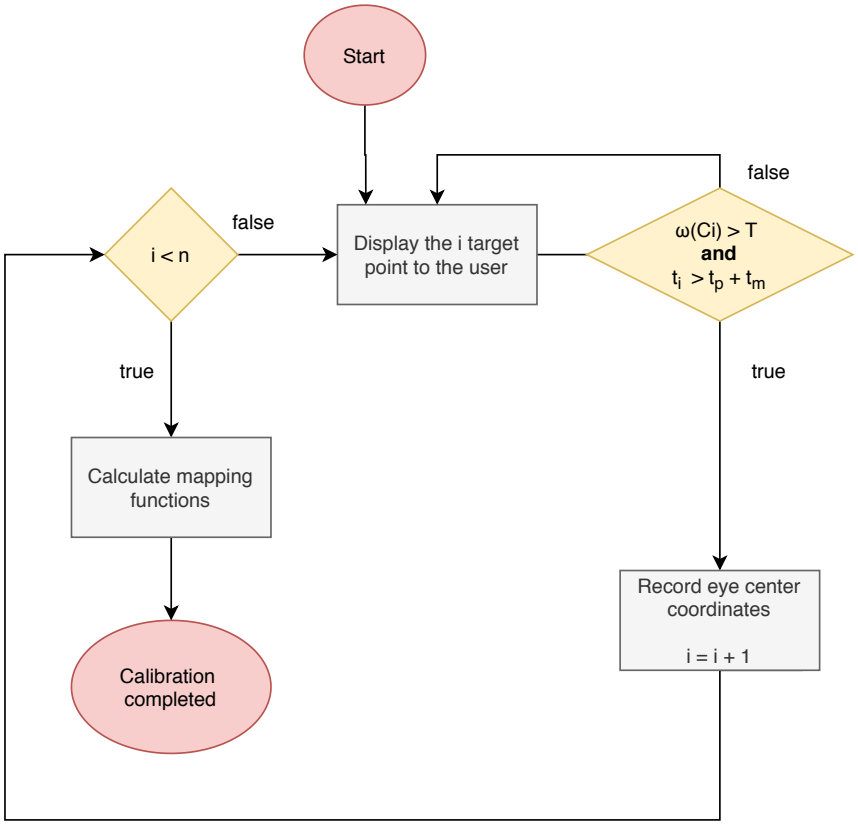


Figure 41: Calibration procedure flowchart (n=calibration points).

determined in the first stage. We proceed to the calculation of the mapping functions, converting iris center positions to screen coordinates.

4.4.7 Mapping eye center co-ordinates to screen co-ordinates

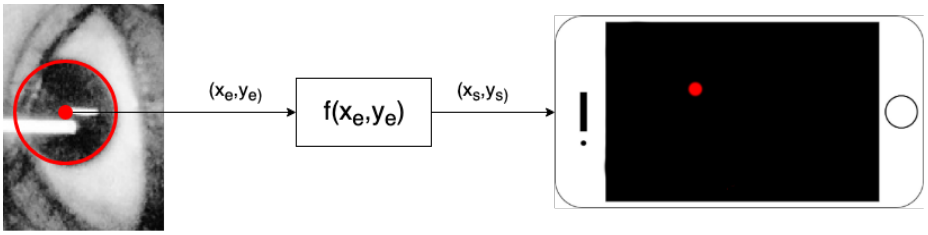


Figure 42: Mapping function illustration.

To estimate the coordinates of the gazed point on screen, a mapping function from eye coordinates to screen coordinates needs to be defined. It is desirable that the mapping function minimizes calibration errors and corrects geometric distortions between the eye-tracker and the screen as much as possible [48]. Once calibration is completed and the mapping function f is determined, the user's gaze coordinates on the screen $\vec{s} = (x_s, y_s)$ can be obtained as $\vec{s} = f(\vec{e})$, or more specifically:

$$x_s = f_x(x_e, y_e) \qquad y_s = f_y(x_e, y_e) \tag{7}$$

We use linear interpolation to determine the user's gaze point. Linear mapping functions divide the screen into a grid of cells and infer the point of regard (PoR) in a linear fashion, interpolating on the known coordinates of the grid cells junctions. We experimented with both linear and non-linear functions and observed that a linear approach better suited our requirements, as the relatively small region to be tracked can be accurately mapped using a dozen of target points. Non-linear mapping functions proved to be highly prone to calibration inaccuracies, either caused by the algorithm's failure to detect the iris center or the users' failure to correctly direct their gaze to the presented calibration points.

4.5 Stage 3: Real-time iris tracking

Our eye tracking pipeline for mobile VR without modifications or IR lighting is ready to track the iris utilizing new eye frames as captured by the front camera (Figure 1b). We begin processing of each

new frame by isolating the ROI in which the iris moves (Figure 1c), using the window determined in Stage 1 (Figure 32). To determine the iris center, we apply the same image processing techniques as in Stage 2 to suppress the intensity values of salient highlights and enhance low contrast areas (Figure 1d) so our customized circle fitting technique can be applied (Figure 1e). Finally, once the iris center has been successfully detected, we convert the iris center coordinates to screen coordinates, using the mapping function determined by the calibration procedure (Stage 2).

As a final step we perform a window averaging of past estimates. Preliminary testing indicated that immediately mapping the estimated gaze points to screen coordinates can exaggerate small tracking errors. Before finalizing a gaze location, we average the last N_s screen points. We experimented with several N_s values but recommend $N_s = 8$ (Section 5.1.4). This results to smoother transitions between estimated gaze points by suppressing sudden error spikes. Higher N_s values result in even smoother transitions but at the cost of a lower gaze position refresh rate. The performance impact of the described averaging operation is negligible ($< 0.2ms$).

Algorithm 3: Stage 3 (real-time iris tracking) pseudocode

Input: Raw eye image, as captured from front facing camera
Output: Estimated gaze location in screen coordinates

Crop ROI;
Downscale;
Downsample;
Suppress reflections;
Apply histogram equalisation (CLAHE);
Attempt circle-fitting via the Hough Gradient method;
for *each candidate iris circle* C_i **do**
    Calculate confidence ω_i;
end
Find max ω_a of ω_i values;
if $\omega_a > T$ **then**
    Iris center coordinates = center coordinates of circle C_a;
else
    Set iris center coordinates to the most recently accepted position;
end
Convert iris center coordinates to screen coordinates using the mapping function;
Final gaze position = average of last N_s successful gaze estimation positions;

Table 1 shows our system's end-to-end latency, from the time a new frame is captured by the front facing camera until the computation of the final estimated gaze location. Intermediate processing steps timings are shown in Table 1. Our computationally efficient algorithm requires less than 20ms per frame to estimate gaze, which corresponds to an upper frequency of about 50Hz, capped to 30Hz due to the camera refresh rate. In practice, the maximum gaze prediction rate of 30Hz may drop when sub-optimal conditions exist, e.g. obtrusive eye lashes, low iris-sclera contrast due to light eye color and very high contrast screen content evoking extreme reflections. In such cases, the algorithm may miss the iris in some frames, an issue present in most eye trackers.

	iPhone 6S		iPhone XS	
	Mean(ms)	**SD(ms)**	**Mean(ms)**	**SD(ms)**
Image cropping	0,01	0,001	0,01	0,001
RGB to grayscale	0,38	0,15	0,072	0,013
Histogram Equalisation	0,94	0,16	0,11	0,02
Image scaling	0,12	0,004	0,027	0,002
Reflection suppression	0,077	0,014	0,01	0,003
Hough Gradient circle-fitting	1,1	0,24	0,229	0,74
Find best circle	0,074	0,008	0,013	0,008
Gaze mapping & averaging	0,2	0,02	0,06	0,01
Camera image acquisition	15,6	4,8	12,87	3,9
Total end-to-end latency	**18,5**	**1,57**	**13,4**	**1,28**

Table 1: Execution times per image proccessing step on iPhone 6S and iPhone XS, total end-to-end latency (applied on cropped images as defined by ROI window, less than 360p in size).

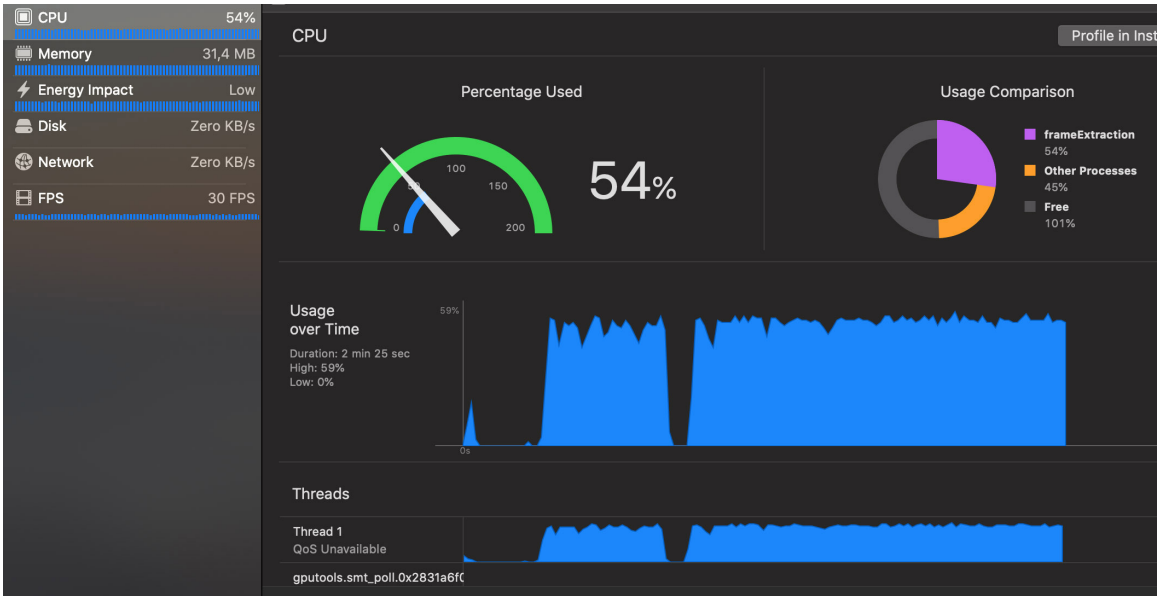


Figure 43: Xcode performance report during real-time iris tracking, on iPhone 6S. Xcode confirms the efficiency of our system, as energy impact and CPU usage remain low over a sustained period of time. If we deduct CPU usage due of other system applications running in the background, our application average CPU usage corresponds to approximately 15% of available CPU resources.

5 System & Applications Evaluation

We evaluated the performance of our system by conducting three experiments: a complete set of performance measurements for our system (Experiment 1), as well as two experiments involving 2 VR apps, a VR game (Experiment 2) and a 360 VR panorama (Experiment 3), developed specifically to incorporate our eye tracking pipeline. Experiment 1 investigated the level of accuracy and precision of the estimated gaze locations derived from our system. Experiment 2 explored the effect of gaze-driven interaction by our system in comparison to head-tracked interaction in a VR game, measuring task completion time. Experiment 3 qualitatively evaluated the usability of our system's gaze-based interaction for viewing 360 VR panoramas.

5.1 Experiment 1: Accuracy and precision of eye tracking in mobile VR

Accuracy and precision can be used to evaluate the validity of the estimated gaze locations of our eye tracking system [20]. Accuracy, also known as systematic error, is defined as the average difference between the real stimuli position and the measured gaze position. Precision, also known as variable error, indicates the ability of the system to consistently reproduce the same gaze point from the same eye input image. One way to characterize precision is to estimate the variation of the recorded data as the root mean square error [19].

Acceptable levels of accuracy and precision depend on the nature of the eye tracked application. In our context we can tolerate larger uncertainties compared to an eye tracker that is based on specialized hardware. We circumvent the larger than usual inaccuracies by proposing design guidelines for the eye tracked GUI elements (Section 5.1.6). It should be noted that given the nature of our setup (smartphone and cardboard-based plastic headset) it is not straightforward to obtain ground truth data. We assume that the users participating in the evaluation experiment were actually fixating on the points as instructed (Section 5.1.5). Users not fixating to the points as instructed, can only *lower* our prediction accuracy measurements, never improve them.

5.1.1 Apparatus & stimuli

We used an Apple iPhone 6S both for displaying the stimuli and tracking the eye. We developed an eye tracking experiment that presents 12 red (RGB: 255,0,0) 20x20px target points on a 3x4 grid

on black background, one at a time. The luminance of the screen was maxed out (500 cd/m^2). The target points are evenly distributed so that the entire visible screen space is covered.

5.1.2 Participants

16 users (3 female, mean age 23.9, SD 3.19) were recruited from our university to participate in the experiment. We ensured that all users exhibited good tracking properties, i.e., without wearing glasses, having droopy or lazy eyes, or other known eye defects. We excluded two such users, one with strabismus and one with a cataract. Two users had blue, two green and the rest had brown-colored eyes.

5.1.3 Procedure & data recording

Users were informed they would use a head-mounted VR display including a smartphone. They were informed about the experimental procedure during which they fixate their gaze at the set of the 12 target points displayed in a specific order one by one. Each target point is presented for $t = 3s$, including a $t = 1s$ data recording pause, so that users have enough time to fixate on them. Target locations can be seen as blue asterisks in Figure 45. Each target point is displayed three times per user for validity. As the users gazed at the target points, the application recorded the following data: *userID*: a unique identifier for each user of the experiment. *recordID* a unique incremental identifier for each tuple of recorded data. *pointID*: an identifier of each of the 12 presented target points in the range [0,11]. *trialID*: number in the range [1,3] which identifies the trial number the tuple belongs to. *elapsedTimePoint*: elapsed time a specific target point is being presented, in milliseconds. *elapsedTimeTotal*: elapsed time since the beginning of the experiment, in milliseconds. *Xpoint, Ypoint*: x,y coordinates of the presented target point. *Xgaze, Ygaze*: x,y coordinates of the estimated gaze point. *Ns*: numbers of averaged samples.

5.1.4 Data analysis & results

Accuracy and precision data across all users are plotted in Figures 44 and 45. We calculated the average accuracy and precision for each of the 12 targets and the mean accuracy across all targets, separately in X and Y axes (Table 2). In the X axis the mean accuracy across all targets was 1.17° (7.9 pixels or 0.06cm on screen). In the Y axis the mean accuracy across all targets was

1.86° (12.5 pixels or 0.1cm on screen). Regarding individual targets, accuracy ranged from 0.04° in the X axis at best to 3.53° in the Y axis at worst. Regarding precision measures, the average standard deviation across participants was 2.87° (19.3 pixels).

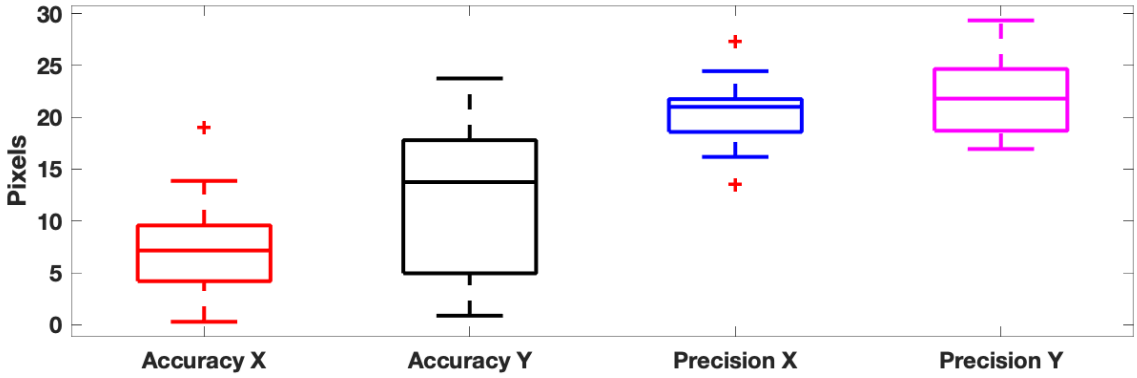


Figure 44: Accuracy and precision of the estimated gaze points in pixels. On each box, the central mark indicates the median. Bottom and top edges indicate the 25th and 75th percentiles. Higher is worse. The whiskers extend to the most extreme data points not considered outliers. Outliers are plotted using the '+' symbol.

Sample averaging did not have a significant effect on precision. In our 30Hz tracking, given that 30 samples per second are processed, 8-sample averaging results in less then $\frac{1}{3}$ of a second ($8/30 = 0.266$s) delay for the gaze point to shift to the latest estimated gaze position. Given that our eye tracking method is intended to be used as a means of user interaction, not for taking ground truth measurements, e.g., for research purposes, this simple computationally inexpensive technique acts as a low-pass filter, with the negligible drawback of introducing a barely noticeable delay in updating the gaze point. Additionally, it should be noted that if a sample yields a low confidence value, e.g. due to a blink, it is excluded from the averaging window.

An one-way ANOVA was conducted to compare the effect of the averaging window Ns on gaze estimation error for $Ns = 1$, 8 and 12. There was not a significant effect of the averaging window size at the $p < .05$ level for the three conditions [$F(3, 12) = 22.1$, $p = 0.27$].

5.1.5 Discussion

Experiment 1 examines the accuracy and precision of our front camera, mobile eye tracker, deployed without specially coated lenses, external IR light, IR cameras or other modifications. Our eye tracker performs best and similarly to eye trackers in commercial VR headsets when the eyes move in the

	avgX	avgY	minX	maxX	minY	maxY
pixels	7.9	12.5	0.3	10.9	0.9	23.7
degrees	1.17	1.86	0.04	1.62	0.13	3.53
cm	0.06	0.1	0	0.08	0.01	0.18

Table 2: Mean, minimum, maximum gaze estimation error (averaged across all targets and subjects) in pixels,degrees of visual angle and cm on screen.

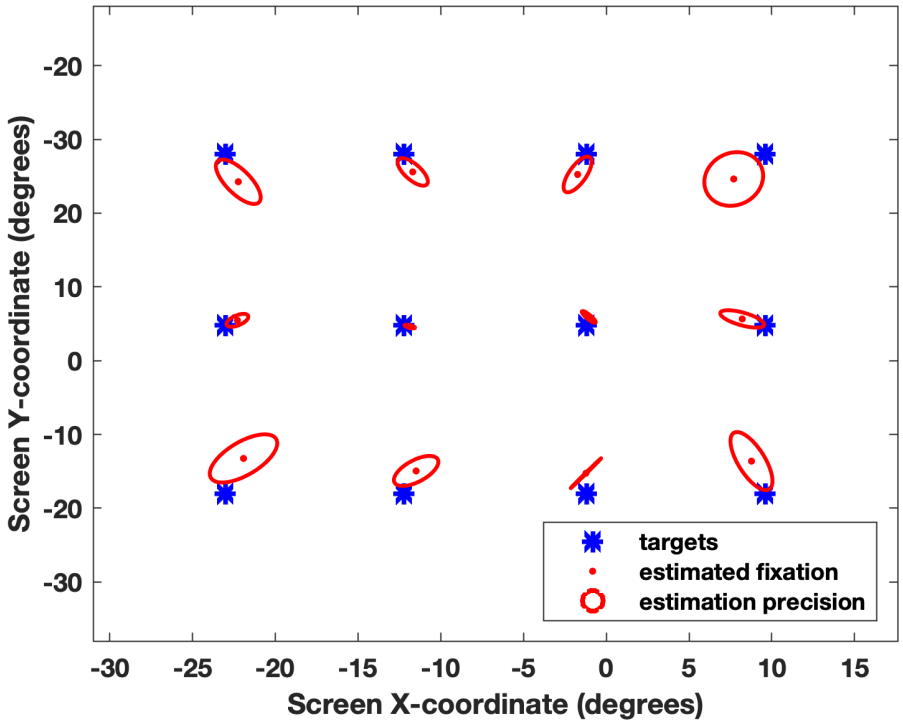


Figure 45: Accuracy and precision of the estimated gaze points in degrees of visual angle. Blue asterisks indicate gaze targets. Red dots denote the mean estimated position. Red ellipses denote root mean square error for the estimated gaze positions.

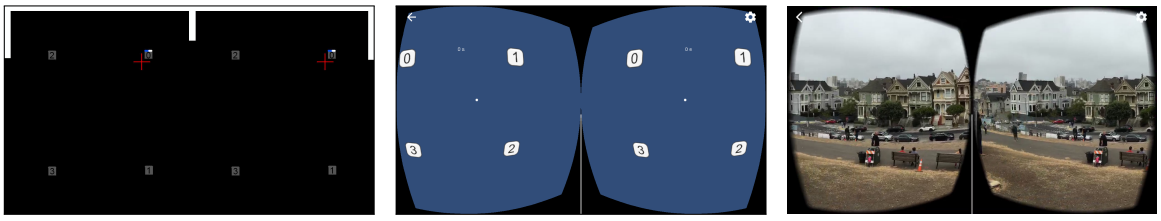


Figure 46: Experiment 2 - eye tracking (left): VR application employing our eye-tracking system. Red crosses are estimated gaze points. Thin bezel of light on upper edge of screen. Experiment 2 - head tracking (middle): Users position the 'reticle' visible as a white dot on GUI elements by rotating/tilting their head. Experiment 3 (right): Eye-tracked 360 VR panorama.

Table 3: Mean, minimum and maximum gaze estimation error in degrees of visual angle per participant.

	avgX	**avgY**	**minX**	**maxX**	**minY**	**maxY**
P1	0.4419	0.8974	0	1.8053	0	6.2620
P2	3.8657	3.4382	0.0186	15.5288	0.6789	8.4543
P3	2.1463	1.7489	0.0461	9.9941	0	7.8432
P4	2.5499	3.6846	0.5128	6.5836	0.2328	8.1267
P5	1.9930	1.6264	0	14.3166	0.0935	6.4079
P6	0.2685	0.8367	0	0.9539	0.0081	3.7304
P7	0.9887	1.0135	0.0646	3.4018	0.0644	3.1764
P8	1.9677	2.8553	0.0656	4.8081	0.1452	6.6017
P9	0.8042	2.2598	0.0092	1.6829	0.1186	6.5690
P10	1.9303	1.7936	0	7.2897	0.0581	7.3437
P11	1.1453	2.3978	0	4.8214	0	6.1706
P12	0.8517	1.5673	0.0631	2.9059	0.3189	3.3240
P13	1.5501	1.5713	0.1661	2.9407	0.0218	4.8277
P14	1.4538	3.6420	0	4.9255	0.8184	6.1966
P15	4.9873	5.2571	0.0342	12.5118	0.5354	13.4429
P16	4.3012	3.3915	0.2338	10.8940	0.0427	8.4977

central part of the headset's FoV (about 20°of visual angle). In a recent study, the accuracy and precision of gaze estimates in head-restrained conditions for the HTC Vive Pro Eye were evaluated [**?**, **?**]. Average accuracy over 50 degrees of the visual field was 4.16°, SD: 3.23 while the precision had a mean of 2.17°, SD: 0.75. In the central part of the visual field with which we compare, accuracy for the HTC Vive Pro was 2.26°, SD: 0.73. For our mobile VR method, in the x-axis the mean accuracy across all targets was 1.17°. In the y-axis the mean accuracy across all targets was 1.86°. Therefore, it is shown that, indeed, our mobile VR system's precision and accuracy in the central FoV is similar to that of commercial systems in the central FoV.

Results consistently demonstrate a slightly higher accuracy on the x-axis than on the y-axis. We have two, potentially inter-related, hypotheses about the slightly higher accuracy on the x-axis: (i) Due to the relative position between the selfie camera and the eye and due to the aspect ratio of the screen, the movement of the eye on the x-axis traces a greater distance compared to the eye's movement on the y-axis. Movement on the x-axis, therefore, can be discriminated better by the image processing pipeline. (ii) Due to the aspect ratio of the camera, having a higher resolution available in the x-axis results to a higher discrimination ability compared to the y-axis.

Mean accuracy and precision decreases with eccentricity (Figure 45). This is expected and does

not affect tracking much, as the users usually move their head instead of their eyes for visual angles larger than 20° [41]. In Section 5.1.6 we provide guidelines for GUI design that, taking the system's accuracy into account, minimize false positive or false negative interactions with GUI elements.

For our proof-of-concept implementation we opted to test our system on the oldest smartphone that we could use, an iPhone 6S (manufactured 2015). During real-time evaluation we observed an average CPU usage of 54% (as reported by XCode) for this device, which showcases the computational efficiency of our system. Image processing operations are accelerated by approximately 5 × times on a relatively newer device (iPhone Xs, Hexa-core (2x2.5 GHz + 4x1.6 GHz)) in comparison to the iPhone 6s employed (dual-core 1.84GHz). Image acquisition performance is marginally improved by 2,73ms (12,87ms vs 15,6ms). Overall end-to-end latency is 5ms lower on the iPhone Xs. These measurements were acquired with the same settings for both phones and 360x480 input image resolution. Temporal averaging of samples did not improve accuracy nor precision (Section 5.1.4), regardless of using 1, 8 or 12 samples. Still, we recommend that 8 samples are averaged instead of no averaging, as this eliminates sudden jumps in tracking due to e.g., blinks.

During Experiment 1, we trust that the users were actually gazing targets as instructed. Users not accurately fixating on targets, resulting to human error, would only further decrease the accuracy and precision measures of our system. It was noted that the system's accuracy is highly sensitive to changes in the VR headset's relative position with respect to the head. The headset must be firmly mounted onto the user's head so that it does not move much during usage. This is feasible and was not an issue during user testing. Only the left eye is tracked based on the smartphone's camera location, therefore, there is no binocular tracking, thus, no depth acquisition.

5.1.6 UI & content design guidelines

Tracking accuracy and precision directly relate to the size of any region (target) in a user interface that the system should recognize if the user fixates on, e.g., a GUI button. A strict estimate would be to require at least 95% of the gaze points to fall inside a target region [8]. We obtained a different accuracy score in the X and Y axes and as such we will estimate a different recommended target/button size for each dimension. Given that 95% of values lie within two standard deviations of the mean for normally distributed data, the suggested S_{width} and S_{height} of the GUI elements should be at least 99px ($\approx 0.8cm$ at 326ppi) and 125px ($\approx 1cm$ at 326ppi) respectively in the X and

Y axis, according to [8]:

$$S = 2 \times (lowestAccuracy + 2 \times precision) \tag{8}$$

$$S_{width} = 2 \times (23.7 + 2 \times 19.3) \approx 125 \tag{9}$$

$$S_{height} = 2 \times (10.9 + 2 \times 19.3) = 99 \tag{10}$$

Our system relies the eye's saliency which is largely dependent on its visibility through the HMD lens and the displayed content. It is shown to function in poor eye illumination conditions and in the absence of IR light and cameras. In cases when the displayed content is very bright, the reflections cast onto the lens of the HMD may partially or completely occlude the eye decreasing prediction accuracy. Thus, the presented system optimally functions when designed content to be displayed does not include large areas of very high luminance. An optimal not extremely bright background and low contrast content could keep obtrusive reflections to a minimum.

5.2 Experiment 2: Eye vs head tracking in a VR game

Experiment 2 compares task completion time employing our system with standard gyroscope-based head tracking to investigate if mobile eye tracking hampers user performance. Two identical VR apps were created, one employing our eye tracking pipeline and the other using standard head tracking, commonly used in "Cardboard" VR apps as the means of input. Users were asked to complete the same task in both apps for fair comparison.

5.2.1 Procedure

We deployed a real-time eye tracking test application on our iOS device. The task emulated a simple memory test game. The application employed a VR User Interface, splitting the smartphone's screen in half and displaying four buttons to be interacted using eye gaze, as shown in Figure 46.

The users were informed that they would be wearing a HMD and were presented with a set of numbered buttons which they should memorize. Each button contained a randomly generated number, present for $t = 2s$. Then, the numbers disappeared and the users were asked to select each button from the lowest to the highest number, as memorized. In the eye tracked condition users

were asked to fixate on the target button. They were not receiving any feedback in relation to their current gaze, i.e., there was no crosshair target. In the head tracked condition a static pointer at the center of the screen could be controlled and aligned with the targets via head movements. The order of the two conditions was randomised. We recorded task completion time for each user. In both conditions the timer started when the first button was activated. The buttons were activated after being fixated/pointed at for 1000ms in both conditions ($t \geq 1s$) to eliminate false selections, also known as the "Midas' touch problem" [24].

5.2.2 Participants

13 users (1 female, mean age 25.5, SD 4.1) were recruited from our university to participate in the experiment. We ensured that all users exhibited good tracking properties as in Exp. 1 (Section 5.1.2).

5.2.3 Results & Discussion

An independent-samples t-test was conducted to compare task completion time between the eye tracked and the head tracked interface. There was not a significant difference in the completion time scores for eye tracked (M=6.9s, SD=2.82s) and head tracked (M=6.88, SD=0.56s) conditions; t(24)=0.01, p = 0.98. This showcases that our eye tracking module does not hamper task performance, while at the same time minimizing cumbersome head motions. We observe that the SD is greater for the eye-tracked interface. We hypothesize that this is because participants were used to head tracked interfaces in mobile VR as this is the de facto standard form of interaction, whereas the eye tracked condition was a new form of interaction that they were not used to. Their lack of experience, however, can only hamper their performance, not improve it.

5.3 Use case study: Eye tracking in a 360 VR panorama

We investigated perceived usability of gaze-driven interaction in a 360 VR panorama as shown in Figure 9 (right). The goal was for users to freely express their views as regards the usability of our system based on the think aloud usability evaluation methodology, as opposed to formal gathering of quantitative data as in Experiments 1 and 2.

Apparatus & Discussion. Several undergraduate and postgraduate computer engineering students in a university laboratory experimented with our eye tracked VR 360 panorama viewer. The panorama viewer incorporated two gaze-driven actions: users could look right to proceed to the next 360 image or left to go back to the previous one. Similarly to previous tasks, the user was required to fixate their gaze left or right for a short amount of time to prevent unintentional actions. Most users were able to successfully cycle between different panoramas by using their gaze. 3 users with light eye color (2 green-eyed, 1 blue-eyed) yielded lower eye-tracking accuracy than users with dark eye color. 1 user of unfavorable eye position leading to poor iris visibility from the smartphone's front camera, was excluded from the study. We noted that when panoramas with very high contrast and color variations in small areas were present, the possibility of the eye being partially or completely occluded from reflections increased, reducing the accuracy of eye tracking.

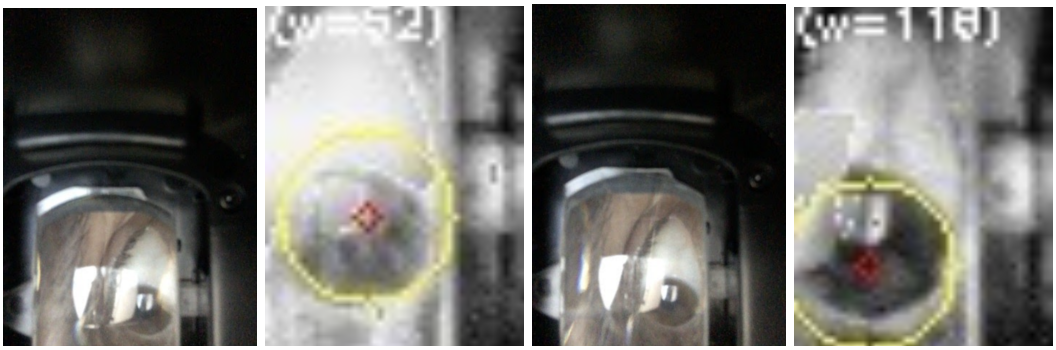
Qualitative evaluation demonstrated that users were able to switch panoramas only by fixating on panoramas' edges, based on users' self-report while immersed in the panorama. No buttons or other UI elements, such as dots or rays, usually visible when VR controllers are employed for interaction were present. They were excited regarding the ease of use of the system. In addition, the ability to freely examine the panoramas in-between gaze-driven actions resulted in positive remarks regarding the clean interface.

6 Conclusions, Limitations and Future Work

We presented a cost-free, mobile VR eye tracking solution based on front camera image capture, able to estimate users' on-screen fixations in real-time, *without* any additional hardware, infrared lighting or other modifications such as added mirrors. Our eye tracker performs similarly to eye trackers in commercial VR headsets when the eyes move in the central part of the headset's FoV. In the x-axis the mean accuracy across all targets was 1.17°. In the y-axis the mean accuracy across all targets was 1.86°. Mean accuracy and precision decreases when drifting away from the centre of the viewer's FoV. This does not affect tracking much, as users usually move their head instead of their eyes when items of interest are placed at larger visual angles.

When using our system, users required the same time to complete a task, as when using head tracked interaction, without the need for laborious consecutive head motions. In a 360 VR panorama, users could successfully switch between panoramas using only gaze, freely exploring the displayed content without obtrusive GUI elements such as buttons or dots, common when VR controllers are used. A general recommendation is that for robust eye tracking in mobile VR using our system, the suggested size for width and height of GUI elements to be gazed at for interaction should be at least 99 and 125 pixels respectively in the X and Y axis. For successful mobile VR eye tracking, the used smartphone should have its front camera in such a position that the eye is visible through the headset's lens. Our pipeline is directly applicable to video-based MR, i.e., MR that overlays virtual elements over the smartphone's main camera feed pointed to the world. An additional benefit of our solution directly relates to user privacy and security. By enabling eye tracking without using IR imaging, the user's iris image is not directly usable for iris-based authentication, shown recently to be a significant security vulnerability for IR based eye trackers [26].

Limitations. An inherent limitation of eye tracking without IR lighting is that our system's performance depends on iris visibility through the HMD lens as a function of the displayed content. Our system maintains its accuracy regardless of poor eye illumination and the absence of infrared light and cameras by displaying a thin bezel of bright luminance along the edge of the screen. However, in cases when the displayed content becomes very bright (see Figure 47), the reflections cast onto the lens of the HMD may partially or completely occlude the eye, decreasing prediction accuracy temporarily. Thus, the presented system optimally functions when displayed content does



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

Figure 47: Two examples of sub-optimal content that decreases accuracy, such as images with high intensity colors and high contrast shapes used as panoramas (see Section 5.3). Such content inevitably produces obtrusive reflections in the raw eye images (a, c), reducing eye saliency and thus hampering accurate gaze estimation (b, d). In both cases our algorithm detected the approximate position of the iris with substantial error, due to the highlights covering the iris' edges, and thus altering its perceived shape. In cases of severe error this is reflected in the low confidence value (ω=52) (a, b).

not include large areas of very high luminance. To keep obtrusive reflections to a minimum, avoiding bright backgrounds is suggested.

Future Work. Future work could introduce automatically-adjusted bezel illumination. This would allow the system to adapt to e.g., significantly different lighting conditions, thus further reducing eye tracking precision errors. We hypothesize that this could work by exploiting content characteristics to predict the expected level of eye illumination. Furthermore, the smartphone's accelerometer could be used to detect abrupt head movements that shift the headset with respect to the head. Even slight headset shifting could negatively impact gaze estimation accuracy, as shown by preliminary testing. We could then initiate a swift re-calibration procedure to counterbalance the headset's shift so that eye tracking accuracy is maintained. Additional future work could evaluate the potential benefits of dynamically adjustable algorithm thresholds per user, based on eye saliency and/or lighting conditions. For instance, if a user yields high confidence values (due to salient iris, dark eye color etc.) over a sustained time period, T can be increased so that sub-optimal fits are rejected and only exceptionally accurate fits are kept (thus further improving overall accuracy). On the other hand, if a user yields low circle fitting confidence scores (due to "lazy" eyes, light eye color, unfavorable eye position in respect to the front-facing camera), T could be decreased to compensate

for the lack of “good fits”. However, dynamically manipulating thresholds is non-trivial, due to the fact that iris saliency is not only dependent on eye morphology, but also on the displayed content. Finally, it is worth mentioning that today’s high-end smartphones come equipped with infrared front-facing cameras. Potential future work could assess the possibility of taking advantage of such hardware to obtain clearer, reflection-free eye images, thus eliminating the presented limitations.

A Appendix

AR	Augmented Reality
ANN	Artificial Neural Network
API	Application Programming Interface
CNN	Convolutional Neural Network
DoF	Degree of Freedom
FoV	Field of View
FPS	Frames Per Second
GUI	Graphical User Interface
HMD	Head-Mounted Display
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IR	Infrared
MR	Mixed Reality
ROI	Region Of Interest
SDK	Software Development Kit
SVM	Support Vector Machine
UI	User Interface
VR	Virtual Reality
XR	Cross Reality

References

[1] K. Ahuja, R. Islam, V. Parashar, K. Dey, C. Harrison, and M. Goel. Eyespyvr: Interactive eye sensing using off-the-shelf, smartphone-based vr headsets. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(2):57, 2018.

[2] R. W. Baloh, A. W. Sills, W. E. Kumley, and V. Honrubia. Quantitative measurement of saccade amplitude, duration, and velocity. *Neurology*, 25(11):1065–1065, 1975.

[3] T. Baltrusaitis, P. Robinson, and L.-P. Morency. Constrained local neural fields for robust facial landmark detection in the wild. In *Proceedings of the IEEE international conference on computer vision workshops*, pp. 354–361, 2013.

[4] S. Baluja and D. Pomerleau. Non-intrusive gaze tracking using artificial neural networks. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1994.

[5] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[6] J. Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[7] J. J. Cerrolaza, A. Villanueva, and R. Cabeza. Study of polynomial mapping functions in video-oculography eye trackers. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 19(2):10, 2012.

[8] A. M. Feit, S. Williams, A. Toledo, A. Paradiso, H. Kulkarni, S. Kane, and M. R. Morris. Toward everyday gaze input: Accuracy and precision of eye tracking and implications for design. In *Proc. of 2017 CHI conf. Human Factors in Computing Systems*, pp. 1118–1130. ACM, 2017.

[9] B. Fischer and E. Ramsperger. Human express saccades: extremely short reaction times of goal directed eye movements. *Experimental brain research*, 57(1):191–195, 1984.

[10] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis & automated cartography. *Communications of ACM*, 24(6):381–395, 1981.

[11] A. Fitzgibbon, M. Pilu, and R. B. Fisher. Direct least square fitting of ellipses. *IEEE Transactions on pattern analysis and machine intelligence*, 21(5):476–480, 1999.

[12] W. Fuhl, T. Kübler, K. Sippel, W. Rosenstiel, and E. Kasneci. Excuse: Robust pupil detection in real-world scenarios. In *Intl Conference on Computer Analysis of Images and Patterns*, pp. 39–51. Springer, 2015.

[13] W. Fuhl, T. Santini, G. Kasneci, W. Rosenstiel, and E. Kasneci. Pupilnet v2. 0: Convolutional neural networks for cpu based real time robust pupil detection. *arXiv preprint arXiv:1711.00112*, 2017.

[14] W. Fuhl, T. C. Santini, T. Kübler, and E. Kasneci. Else: Ellipse selection for robust pupil detection in real-world environments. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, pp. 123–130. ACM, 2016.

[15] W. Gao, X. Zhang, L. Yang, and H. Liu. An improved sobel edge detection. In *2010 3rd International Conference on Computer Science and Information Technology*, vol. 5, pp. 67–71. IEEE, 2010.

[16] A. George and A. Routray. Real-time eye gaze direction classification using convolutional neural network. In *2016 International Conference on Signal Processing and Communications (SPCOM)*, pp. 1–5. IEEE, 2016.

[17] S. W. Greenwald, L. Loreti, M. Funk, R. Zilberman, and P. Maes. Eye gaze tracking with google cardboard using purkinje images. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pp. 19–22. ACM, 2016.

[18] H. Hakoda, W. Yamada, and H. Manabe. Eye tracking using built-in camera for smartphone-based hmd. In *30th Annual ACM Symposium on User Interface Software and Technology, UIST 2017*, pp. 15–16. Association for Computing Machinery, Inc, 2017.

[19] K. Holmqvist, M. Nyström, R. Andersson, R. Dewhurst, H. Jarodzka, and J. Van de Weijer. *Eye tracking: A comprehensive guide to methods and measures*. OUP Oxford, 2011.

[20] A. J. Hornof and T. Halverson. Cleaning up systematic error in eye-tracking data by using required fixation locations. *Behavior Research Methods, Instruments, & Computers*, 34(4):592–604, 2002.

[21] M. X. Huang, J. Li, G. Ngai, and H. V. Leong. Screenglint: Practical, in-situ gaze estimation on smartphones. In *Proc. 2017 CHI Conference on Human Factors in Computing Systems*, pp. 2546–2557. ACM, 2017.

[22] J. Illingworth and J. Kittler. A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.

[23] https://docs.swift.org/swift-book/LanguageGuide/TheBasics.htmlA. Inc. https://docs.swift.org/swift-book/LanguageGuide/TheBasics.htmlSwift basics.

[24] R. Jakob. The use of eye movements in human-computer interaction techniques: what you look at is what you get. *Readings in intelligent user interfaces*, pp. 65–83, 1998.

[25] A.-H. Javadi, Z. Hakimi, M. Barati, V. Walsh, and L. Tcheang. Set: a pupil detection method using sinusoidal approximation. *Frontiers in neuroengineering*, 8:4, 2015.

[26] B. John, S. Koppal, and E. Jain. Eyeveil: degrading iris authentication in eye tracking headsets. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, pp. 1–5, 2019.

[27] P. Kampmann. Development of a multi-modal tactile force sensing system for deep-sea applications. 02 2016.

[28] A. Kar and P. Corcoran. A review and analysis of eye-gaze estimation systems, algorithms and performance evaluation methods in consumer platforms. *IEEE Access*, 5:16495–16519, 2017.

[29] P. Kasprowski, K. Harężlak, and M. Stasch. Guidelines for the eye tracker calibration using points of regard. In *Information Technologies in Biomedicine, Volume 4*, pp. 225–236. Springer, 2014.

[30] M. Kassner, W. Patera, and A. Bulling. Pupil: an open source platform for pervasive eye tracking and mobile gaze-based interaction. In *Proc. of 2014 ACM intl joint conference on pervasive and ubiquitous computing: Adjunct publication*, pp. 1151–1160. ACM, 2014.

[31] D. Katrychuk, H. K. Griffith, and O. V. Komogortsev. Power-efficient and shift-robust eye-tracking sensor for portable vr headsets. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, pp. 1–8, 2019.

[32] G. A. Koulieris, K. Akşit, M. Stengel, R. Mantiuk, K. Mania, and C. Richardt. Near-eye display and tracking technologies for virtual and augmented reality. In *Computer Graphics Forum*, vol. 38, pp. 493–519, 2019.

[33] G. A. Koulieris, G. Drettakis, D. Cunningham, and K. Mania. Gaze prediction using machine learning for dynamic stereo manipulation in games. In *2016 IEEE Virtual Reality (VR)*, pp. 113–120. IEEE, 2016.

[34] E. Kowler. Eye movements: The past 25 years. *Vision research*, 51(13):1457–1483, 2011.

[35] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik, and A. Torralba. Eye tracking for everyone. In *Proc. IEEE conf. on computer vision and pattern recognition*, pp. 2176–2184, 2016.

[36] D. Li, D. Winfield, and D. J. Parkhurst. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *2005 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR'05)-Workshops*, pp. 79–79, 2005.

[37] T. Li, Q. Liu, and X. Zhou. Ultra-low power gaze tracking for virtual reality. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pp. 1–14, 2017.

[38] R. Maini and H. Aggarwal. Study and comparison of various image edge detection techniques. *International journal of image processing (IJIP)*, 3(1):1–11, 2009.

[39] K. Nguyen, C. Wagner, D. Koons, and M. Flickner. Differences in the infrared bright pupil response of human eyes. In *Proceedings of the 2002 symposium on Eye tracking research & applications*, pp. 133–138, 2002.

[40] https://docs.opencv.org/3.4/d3/de5/tutorial$_{j}s_{h}oughcircles.htmlOpenCV.https://docs.opencv.org/3.4/d3/de5/$

[41] https://doi.org/10.1007/978-3-540-29678-2$_{3}257D.\ PelissonandA.\ Guillaume.https://doi.org/10.1007/978-3-$

[42] R. Ranjan, S. De Mello, and J. Kautz. Light-weight head pose invariant gaze tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2156–2164, 2018.

[43] https://doi.org/10.1109/JSEN.2017.2762704I. Rigas, H. Raffle, and O. V. Komogortsev. https://doi.org/10.1109/JSEN.2017.2762704Hybrid ps-v technique: A novel sensor fusion approach for fast mobile eye-tracking with sensor-shift aware correction. https://doi.org/10.1109/JSEN.2017.2762704*IEEE Sensors Journal*, https://doi.org/10.1109/JSEN.2017.276270417(24):8356–8366, https://doi.org/10.1109/JSEN.2017.27627042017. https://doi.org/10.1109/JSEN.2017.2762704 doi: 10.1109/JSEN.2017.2762704

[44] https://doi.org/https://doi.org/10.1016/S0166-2236(00)01685-4J. Ross, M. Morrone, M. E. Goldberg, and D. C. Burr. https://doi.org/https://doi.org/10.1016/S0166-2236(00)01685-4Changes in visual perception at the time of saccades. https://doi.org/https://doi.org/10.1016/S0166-2236(00)01685-4*Trends in Neurosciences*, https://doi.org/https://doi.org/10.1016/S0166-2236(00)01685-424(2):113 – 121, https://doi.org/https://doi.org/10.1016/S0166-2236(00)01685-42001. https://doi.org/10.1016/S0166-2236(00)01685-4 doi: 10.1016/S0166-2236(00)01685-4

[45] T. Santini, W. Fuhl, and E. Kasneci. Pure: Robust pupil detection for real-time pervasive eye tracking. *Computer Vision and Image Understanding*, 170:40–50, 2018.

[46] https://doi.org/10.1145/1753846.1754048W. Sewell and O. Komogortsev. https://doi.org/10.1145/1753846.1754048Real-time eye gaze tracking with an unmodified commodity webcam employing a neural network. https://doi.org/10.1145/1753846.1754048In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, https://doi.org/10.1145/1753846.1754048CHI EA '10, https://doi.org/10.1145/1753846.1754048pp. 3739–3744. https://doi.org/10.1145/1753846.1754048ACM, https://doi.org/10.1145/1753846.1754048New York, NY, USA, https://doi.org/10.1145/1753846.17540482010. https://doi.org/10.1145/1753846.1754048 doi: 10.1145/1753846.1754048

[47] I. Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[48] D. M. Stampe. Heuristic filtering and reliable calibration methods for video-based pupil-tracking systems. *Behavior Research Methods, Instruments, & Computers*, 25(2):137–142, 1993.

[49] L. Świrski, A. Bulling, and N. Dodgson. Robust real-time pupil tracking in highly off-axis images. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pp. 173–176. ACM, 2012.

[50] C.-H. Teh and R. T. Chin. On the detection of dominant points on digital curves. *IEEE Transactions on pattern analysis and machine intelligence*, 11(8):859–872, 1989.

[51] https://doi.org/10.1145/3130971M. Tonsen, J. Steil, Y. Sugano, and A. Bulling. https://doi.org/10.1145/3130971Invisibleeye: Mobile eye tracking using multiple low-resolution cameras and learning-based gaze estimation. https://doi.org/10.1145/3130971*Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, https://doi.org/10.1145/31309711(3):106:1–106:21, https://doi.org/10.1145/3130971Sept. 2017. https://doi.org/10.1145/3130971 doi: 10.1145/3130971

[52] Tsuji and Matsumoto. Detection of ellipses by a modified hough transformation. *IEEE Transactions on Computers*, C-27(8):777–781, 1978.

[53] A. Wojciechowski and K. Fornalczyk. Single web camera robust interactive eye-gaze tracking method. *Bulletin of the Polish Academy of Sciences Technical Sciences*, 63(4), 2015.

[54] https://doi.org/10.1109/ICMLA.2018.00085Y. Yin, C. Juan, J. Chakraborty, and M. P. McGuire. https://doi.org/10.1109/ICMLA.2018.00085Classification of eye tracking data using a convolutional neural network. https://doi.org/10.1109/ICMLA.2018.00085In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, https://doi.org/10.1109/ICMLA.2018.00085pp. 530–535, https://doi.org/10.1109/ICMLA.2018.000852018. https://doi.org/10.1109/ICMLA.2018.00085 doi: 10.1109/ICMLA.2018.00085

[55] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling. Mpiigaze: Real-world dataset and deep appearance-based gaze estimation. *IEEE transactions on pattern analysis and machine intelligence*, 41(1):162–175, 2017.

[56] Z. Zhu and Q. Ji. Eye and gaze tracking for interactive graphic display. *Machine Vision and Applications*, 15(3):139–148, 2004.

[57] Z. Zhu, Q. Ji, and K. P. Bennett. Nonlinear eye gaze mapping function estimation via support vector regression. In *18th Intl. Conference on Pattern Recognition (ICPR'06)*, vol. 1, pp. 1132–1135. IEEE, 2006.

[58] K. Zuiderveld. Contrast limited adaptive histogram equalization. In *Graphics gems IV*, pp. 474–485. Academic Press Professional, 1994.