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

PANAGIOTIS DRAKOPOULOS, Technical University of Crete, Greece
GEORGE-ALEX KOULIERIS, Durham University, United Kingdom
KATERINA MANIA, Technical University of Crete, Greece

Input methods for interaction in smartphone-based virtual and mixed reality (VR/MR) are currently based on uncomfortable head 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 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. 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 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.

CCS Concepts: • **Computing methodologies** → **Virtual reality**;

Additional Key Words and Phrases: Mobile VR, eye tracking

ACM Reference format:
Panagiotis Drakopoulos, George-Alex Koulieris, and Katerina Mania. 2021. Eye Tracking Interaction on Unmodified Mobile VR Headsets Using the Selfie Camera. *ACM Trans. Appl. Percept.* 18, 3, Article 11 (May 2021), 20 pages.
https://doi.org/10.1145/3456875

Authors' addresses: P. Drakopoulos and K. Mania, School of Electrical and Computer Engineering, Technical University of Crete, University Campus - Kounoupidiana 731 00 Chania - Crete, Greece; emails: panosdrak@hotmail.com, k.mania@ced.tuc.gr; G.-A. Koulieris, Mathematical Sciences & Computer Science Building, Durham University, Upper Mountjoy Campus, Stockton Road, Durham, DH1 3LE, United Kingdom; email: georgios.a.koulieris@durham.ac.uk.

1544-3558/2021/05-ART11 $15.00
https://doi.org/10.1145/3456875

1 INTRODUCTION

Low-cost mobile **virtual and mixed reality (VR/MR)** headsets, i.e., a mobile phone placed in an inexpensive cardboard or plastic case, represent a cheap, widely available medium for immersive VR/MR. Accurate eye tracking requires infrared illumination and cameras, unavailable on cheap mobile VR headsets. In this article, we propose a mobile VR eye tracking methodology, utilizing eye images from the front-facing (selfie) camera *without* headset modifications or add-on hardware, contrary to previous work. The achieved eye tracking accuracy is similar to eye trackers in commercial VR headsets in the central **Field-of-View (FoV)** of about 20° of visual angle.

Prior work has demonstrated that eye fixations are a natural input method for interaction with head-worn displays [8, 35]. Until now, input methods for interaction in mobile VR are usually limited to uncomfortable head tracking controlling a pointer on the screen or occasionally, using the side button of the case. Separately purchased gamepads have limited degrees of freedom employing a four-way digital cross, compatible with only a few applications.

Current eye tracking in headsets, such as the HTC Vive Pro Eye, relies on the enhanced pupil contrast provided by **infrared (IR)** light emitters. In mobile VR, illumination only emanates from 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 infrared. Thus, eye illumination and eye tracking depend on the displayed content. When the illumination is too strong, obtrusive reflections emerge on the headset's lens making eye detection hard. When the illumination is too weak, eye tracking fails as the iris is barely visible. Initial attempts for mobile VR eye tracking either involved add-on hardware such as mirrors [15], anti-reflective layers to combat reflections [16], or low-accuracy deep learning running on an external computer [3] (Section 2). Our proposed technique improves upon previous work by customising an eye tracking pipeline from image acquisition to processing to gaze mapping to address mobile VR's challenges for iris detection, e.g., the lens reflections, the oblique placement of the camera and the mobile hardware performance. A preliminary outline of our work was presented as a poster [7].

We first improve eye illumination. We display a thin, imperceptible bezel of light placed outside of the observer's field-of-view for adequate illumination. As screen content *and* illumination generate obtrusive lens reflections, we assemble a pipeline of low-level image enhancements specifically to suppress them (Figure 1). The now-refined captured images, otherwise suffering from low contrast and poor lighting, carry easily detected features used to fit a geometric model to the iris. For this, we utilize an accelerated version of the original Hough Transform [40], the "Hough Gradient" method that detects elliptical shapes efficiently in terms of computing time and customize it further to be highly sensitive to circular features, i.e., the iris as seen through the phone camera. Our model, based on a set of fitting confidence metrics, obtains an accurate estimation of the iris center in most cases. We conclude our pipeline by proposing purpose-specific calibration and gaze mapping algorithms to map the detected eye center co-ordinates to screen coordinates.

Our contribution is threefold. First, we present a computationally efficient, eye tracking technique for smartphone-based VR, also directly applicable to video-based MR. Our method uses eye images captured from the front camera of a smartphone, through the headset's lens and *without* an added infrared light source or other modifications (Sections 3 and 4). Second, we perform an eye tracking accuracy and precision study. Our eye tracker performs similarly to trackers in commercial headsets when the eyes move in the central field-of-view. We devise **user interface (UI)** and content design guidelines to assist accurate interaction in gaze-aware VR (Section 5.1). Third, we demonstrate the strengths of our eye tracking pipeline over head tracking in a VR evaluation study. Eye-driven task completion time is at par with head-tracked interfaces without the cumbersome head motions (Section 5.2). We showcase a practical use of the system in a VR panorama viewer (Section 5.3).

2 RELATED WORK

The proposed eye tracking pipeline for mobile VR is based on successful iris detection and tracking in real time. We start by reviewing previous image processing methods widely adopted in eye tracking (Sections 2.1 and 2.2)

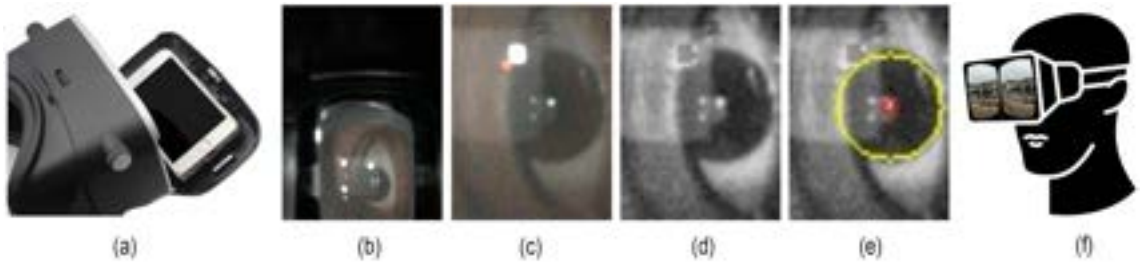

Fig. 1. (a) Unmodified mobile VR headset with smartphone. (b) Original image as captured from the front-facing camera with reflections on headset's lens and eye cornea. (c) Cropped RoI. (d) Enhanced image. (e) Iris contour (yellow) and center (red) estimation. (f) User interacts with the gaze-aware VR environment.

that, however, often fail to work for eye tracking in mobile VR mainly because of low light and lens reflections. We proceed by discussing gaze mapping models for calculating users' point of gaze (Section 2.3). We conclude by detailing previous work on eye tracking in mobile VR (Section 2.4).

2.1 Model-based vs. Feature-based Algorithms

Eye tracking techniques can be classified as *feature based* and *model based* [31]. Feature-based detect a key feature, e.g., the dark pupil, by relying on pixel intensity levels or intensity gradients. Whether the key feature selected is present or not is decided upon a user-selected pixel intensity threshold based on application goals. According to the iris-appearance hypothesis for dark-pupil eye tracking, the pupil is the darkest element to detect [39].

Model-based approaches find the best fitting circle or ellipse of the pupil. Due to iterative searches of the model parameter space, model-based methods yield a more precise estimate of the pupil center than feature based, but at a higher computational cost. Therefore, their use is limited in time-critical mobile VR eye tracking, as resources are scarce and dropping frames is not an option. Also, a real-time eye tracking system cannot solely rely on feature-based methods due to the highly variable illumination, resulting in highly variable appearance of key features searched, making stable iris detection impossible. Model-based approaches have been employed for eye tracking on tablets from a hand-held distance [41]. Mobile VR eye tracking suffers from an additional set of challenges compared to eye tracking on tablets, such as dealing with lens reflections and low visibility.

Standard eye tracking using infrared light aims for a tradeoff between runtime performance and accuracy by combining model and feature-based approaches [28]. The Starburst algorithm achieves a good tradeoff between runtime performance and accuracy by combining model and feature-based methods [31]. The algorithm iteratively locates the pupil center as the mean of points that exceed a differential luminance threshold along the rays extending from the last best guess. ScreenGlint extracts a rectangle based on the assumption that the darkest area in the image corresponds to the iris, in which ellipse fitting and iris segmentation techniques are then applied [19].

Inspired by desktop eye tracking, our mobile VR methodology first deploys a feature-based algorithm to determine the approximate position of the iris and then a purpose-specific, computationally efficient model-based approach to obtain an accurate estimation of the iris center.

2.2 Eye, Iris, and Pupil Detection

There are several ways to detect a circular shape in an image. Ellipse-fitting methods are commonly used in the field of eye tracking 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 systems, as the input images contain pixels that do not correspond to the pupil

or eye boundary. These pixels are leftover image noise, representing other noticeable features such as eyelids or reflections, troublesome to remove.

Feature-based methods 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 [34, 39]. The more robust ellipse-fitting methods are either voting or search based. Hough transform and Random Sample Consensus are primary examples of voting-based and search-based methods, respectively [10]. 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.

In the SET method, the convex hull segments of thresholded regions are fit to sinusoidal components that compose the circle corresponding to the eye [22]. ElSe [13], PuRe [34], ExCuSe [12], and Pupil [26] methods use morphological edge filters to thin, straighten, or separate lines so that ellipse fitting can be successfully applied.

The main drawback of most image processing algorithms above is that they heavily rely on the enhanced pupil contrast provided by the infrared light of specialized eye trackers. Our eye tracking pipeline strives for accurate eye detection on the visible spectrum based only on the front camera capture of a standard mobile phone without infrared lighting.

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 to screen coordinates is accomplished by a mapping function, determined per user through calibration, i.e., by fixating on a set of target points of known positions [26, 31, 38]. 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 mapping function is a critical factor for high eye tracking accuracy. Mapping functions range from linear and nonlinear polynomials [25, 38], geometric based to neural network based, and support vector regression-based functions [42, 43]. Previous research has shown that no single mapping function is ideal for all eye tracking applications. Counterintuitively, high-order polynomials or complex geometry-based models do not improve accuracy over simpler linear functions in eye tracking scenarios with limited head movement [6]. Furthermore, increasing the number of calibration points beyond approximately 12 does not yield any meaningful improvements [25]. Our initial lab tests corroborate previous work, as high-order polynomials did not improve accuracy, nor did an increased amount of calibration points. Based on the above and driven by our requirement for high accuracy and low latency, we employ the computationally more efficient linear mapping function to infer the gaze point.

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 [16]. 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 [15]. 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 [3]. 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, and 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 [30].

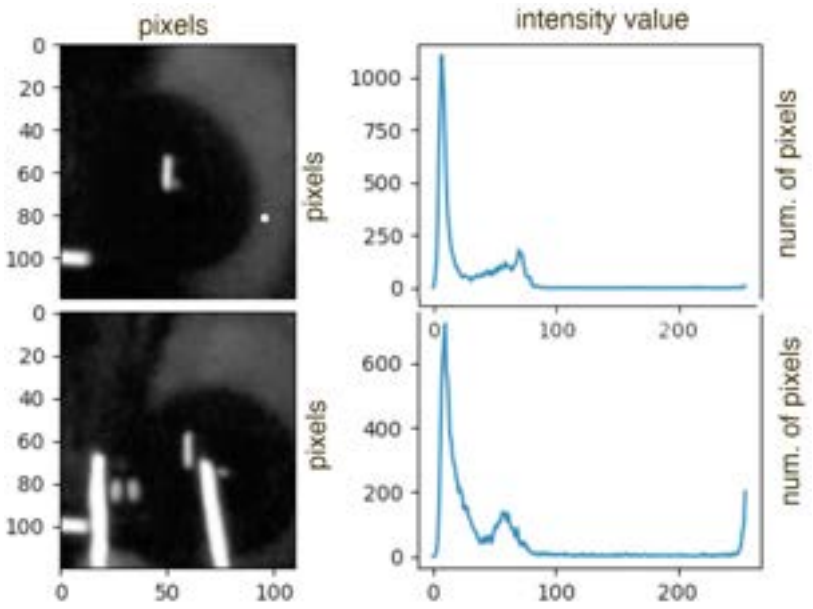

Fig. 2. Image histogram with no lens reflections (top) and strong lens reflections (bottom) with a visible peak of high intensity values.

Photosensor oculography [27] and LiGaze [32] require a new type of sensor that is unavailable in smartphones. 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 [29]. ScreenGlint utilizes a smartphone and its front-facing camera for tracking the user's gaze by exploiting the screen's reflection on the cornea [19]; however, this approach was only tested when holding the smartphone at a distance and not inside a VR headset. A comprehensive review of generic gaze estimation techniques can be found here [24].

3 CHALLENGES AND SYSTEM OVERVIEW

In this section, we describe the technical challenges of mobile VR eye tracking, along with our proposed solutions.

Headset lens reflections: The lens of the mobile VR head-worn case is situated between the captured eye and the smartphone's front-facing camera. As a result, reflections of variable shape, size, and intensity emerging from the displayed content are cast onto the lens. Consequently, a large part of the iris may be occluded (Figure 2). Iris detection becomes a non-trivial task.

Corneal reflections: Due to the small distance between the user's eye and the smartphone's screen, reflections on the eye's cornea are also expected (Figure 3). These reflections introduce noise during the image processing stage and decrease gaze estimation accuracy. 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 4.2).

Content limitations: In contrast to specialized eye tracking devices, eye illumination solely depends on displayed content. If the displayed content is too dark, then the eye will not be illuminated enough for the iris to be detected. We display a thin, imperceptible, bright-white bezel of light along the upper edge of the screen, 18 pixels high (0.14 cm at 326 **pixels-per-inch (ppi)**) providing sufficient illumination at all times. The total area of the thin bezel is only 2.4% of the entire FoV, and thus it is unlikely it alters the perceived brightness of the scene.

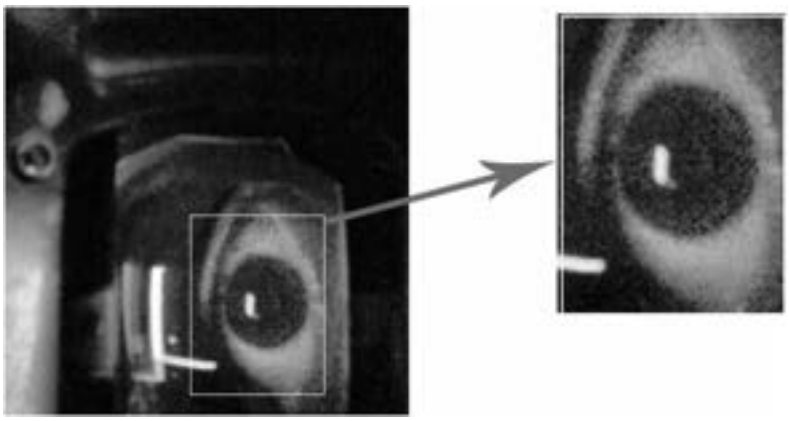

Fig. 3. ROI detection provides as input to the next process a subsection of the captured images, representing the detected ROI rectangle. Two strong reflections are also visible—one on the headset's lens and one on the cornea.

Position of front-facing camera: Most smartphones' front camera is at an oblique position with respect to the eye. This results to a distorted view of the eye, complicating pupil registration. Our customised circle fitting algorithm iteratively improves upon an initial pupil location estimate to maximize accuracy (Section 4.2).

Hardware and performance: Eye tracking involves simultaneously executing several, time-critical processes including video capture, image cropping and enhancement and coordinate systems re-mapping. The simultaneous operation of eye tracking and a VR application can easily overwhelm even the latest generation of smartphones. Our system works without requiring computationally heavy machine learning classifiers, enabling its fast execution even on older generation smartphones or cheap current generation ones.

We now describe the main components of our mobile VR eye tracker addressing the challenges analysed above (Figure 4). To demonstrate that our technique works on older generation hardware, an Apple iPhone 6S was employed as the eye tracking device. Its 4.7″ display has a resolution of 1334 × 750, 326 ppi. The device comes with a 5-MP front camera with 720p@30fps video capture capability, along with a 1.84-GHz dual-core CPU and 2 GB RAM. We placed the mobile phone in a commodity head mounted case for mobile VR with head straps, without infrared lighting or any modification. We employed the OpenCV iOS framework for image processing [1]. The eye tracking evaluation environment and VR apps were developed in Xcode, Apple's native integrated development environment for iOS applications [2]. The system operates in three distinct stages.

Stage 1: Iris Region of Interest detection: We first detect an approximate eye position in the captured images to reduce the search space in the following stages. We employ circle fitting on the images as captured by the front camera. The **Region of Interest (ROI)** is calculated and cropped only once—before calibration (Figure 1(b), resulting image (c)) (Section 4.1).

Stage 2: Image enhancement and calibration: We improve iris visibility by enhancing the contrast of the iris and suppressing the intensity of bright highlights (Figure 1(c), resulting image (d)). We proceed with system calibration. Any further circle fitting occurs now only on the cropped ROI region. Based on calibration data, we calculate the mapping functions, which convert the detected iris center positions to screen coordinates (Section 4.2).

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, as in the previous stage, to the cropped images that include the ROI region determined in the first stage. We then re-apply circle fitting to determine the iris centre. Using the mapping function, we determine the final gaze point on the screen, resulting in real-time iris tracking (Section 4.3).

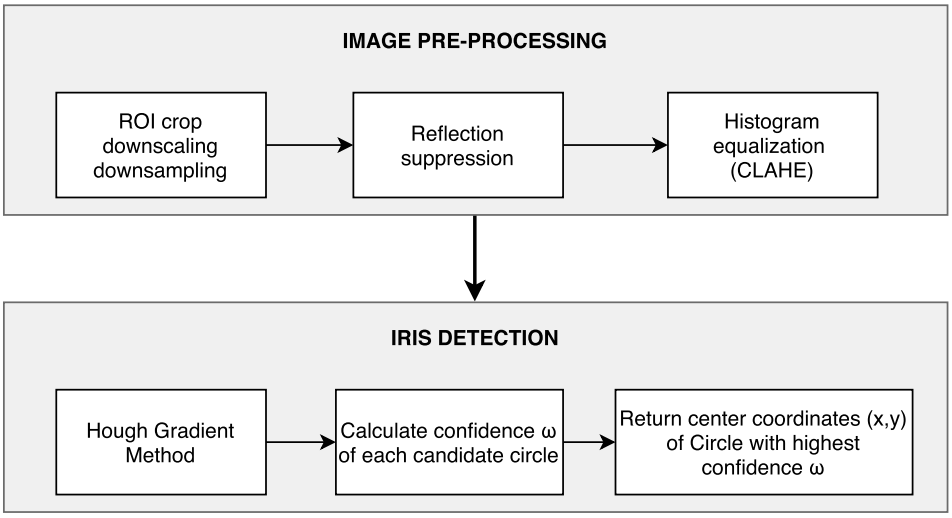

Fig. 4. Flowchart depicting the steps involved in processing the original image to calculate the iris center position.

In summary, Stages 1 and 2 run only once before entering Stage 3. During Stages 1 and 2, visibility enhancement and circle fitting are conducted to perform calibration. Visibility enhancement and circle fitting are re-applied during Stage 3 but this time for gaze mapping.

4 MOBILE VR EYE TRACKING

In this section, we describe each stage in detail.

4.1 Stage 1: Iris ROI Detection

Images of the eye, as captured from the mobile phone front-facing camera, contain irrelevant information, such as the plastic parts surrounding the lenses (Figure 1(b)). Only a small section of the captured image is useful to track the iris (Figure 1(c)). We drastically reduce image size by cropping the ROI that corresponds to the eye and its surrounding area. This operation, designed to only be run once, has a positive effect on the iris tracking speed in subsequent stages, as the iris search space is significantly smaller. Consequently, less system resources are required, making real-time mobile eye tracking possible.

To calculate a conservative ROI, that ensures the iris is visible irrespective of gaze, the user is asked to fixate on a set of 12 target squares (20×20 pixels each) that cover the entire screen and presented one at a time. We apply our customised circle-fitting algorithm to detect the iris position within the captured images, as described below.

4.1.1 Circle Fitting on the Iris Contour. Captured images vary in brightness, reflections, and iris saliency. Thus, we customised the Hough gradient function, a cost-efficient version of the Hough transform [40], 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 [20, 40], and, therefore, we developed a customized accelerated Hough transformation that 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 [14] 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. An estimated 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 at a sufficient distance from previous centers [5]. This helps reduce the computational cost of the problem, by requiring a two-dimensional (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.

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 that 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.

4.1.2 Confidence Metric. The confidence metric is based upon two purpose-specific, experimentally validated indicators, h_1 and h_2.

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.

h_2: The iris appears as a darker circular region surrounded by a brighter region, regardless of the lighting conditions. If the mean intensity of the candidate iris segment is lower than the mean intensity of the surrounding region, then the candidate is a possible iris. The higher the contrast, the higher the confidence.

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:

$$\omega(C_i) = h_1(C_i) \cdot w_1 + h_2(C_i) \cdot w_2. \tag{1}$$

Weights w_1 and w_2 were experimentally determined by investigating the effect of each indicator on iris detection accuracy, in lab tests, using eye images with varying eye saliency scenarios as input. Though both indicators influenced detection accuracy, it was shown that h_2 is a more robust indicator compared to h_1. This can be largely attributed to the fact that h_1 is invalidated in scenarios where bright highlights fall onto the iris or for lighter eye colors, increasing the brightness of the otherwise dark iris. We experimentally set $w_1 = 1$ and $w_2 = 1.9$, although value-pairs of a similar ratio are also applicable. 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. When none of the candidate circles' confidence value $\omega(C_a)$ passes an experimentally defined threshold T, gaze estimation for that frame fails. The frame is rejected and the returned 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 $T = 65$. 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. Figure 5 depicts a set of random iris frames and their respective confidence values. Thus the iris center coordinates $\vec{e}$ for any given frame are equal to the following:

$$\vec{e} = \begin{cases} \vec{e_a}, & \text{if } \omega(C_a) \geq T \\ \vec{e}_{previous}, & \text{otherwise} \end{cases}, \omega(C_a) = \max_{i=1..k} \omega(C_i). \tag{2}$$

4.1.3 ROI Calculation. Upon fixating on all 12 targets, the iris ROI is defined as a rectangle, with its center being the average of the 12 recorded iris center positions. ROI length and width are calculated per user based on the minimum and maximum x and y coordinates of the estimated iris center positions and an average iris pixel radius. This is to ensure that the iris survives cropping for all possible eye positions.

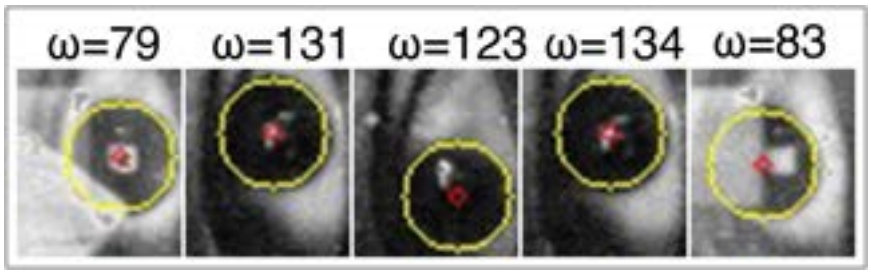

Fig. 5. The higher the confidence value ω, the higher the probability the candidate circle corresponds to the iris. Challenging conditions (e.g., obtrusive reflections, eye angle in respect to the camera) yield lower confidence scores.

For real-time eye tracking the defined iris ROI (calculated only once, Figure 3) is cropped from all input images, before they are fed forward to the pipeline. This allows for a drastic decrease in the input's size by almost 8 times, greatly reducing CPU usage, thus, enabling real-time eye tracking on a mobile device.

4.2 Stage 2: Image Enhancement and Calibration

4.2.1 Improving Iris Visibility. The smartphone's camera captures the eye ROI at a rate of 30 fps. We convert input images to greyscale and downscale them by a factor of two to further reduce computational cost without reduction of the visual information required. Accurately tracking the iris is impossible as images suffer from poor contrast and obtrusive reflections. We apply a pair of selected low-level image enhancements that we found result in a more salient iris in mobile VR. We perform reflections suppression and histogram equalisation.

Reflection suppression: The iris may be occluded by screen reflections on the HMD lens or the eye's cornea and sclera. Accurately removing reflections is not trivial. However, for eye tracking it is unnecessary, as we only need to accurately *suppress* reflections to ensure the iris circumference is identified as accurately as possible.

To suppress reflections, salient highlights can be detected as an abrupt peak of high intensity values in the histogram. We first estimate a high intensity threshold in real time per frame equal to the local maximum of the histogram derivatives in the high intensity part of the spectrum. The intensity of pixels with values higher than the intensity threshold (i.e., reflections) is suppressed by averaging such pixels with their lower intensity neighbouring pixels. These neighboring pixels have intensity values lower than the threshold and as such are not considered part of the reflection.

Histogram equalisation: Eye images captured from within the headset suffer from poor contrast due to the low light conditions. The intensities of such images are restricted in a narrow range of values. Histogram equalisation can expand 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. In our system we employ **Contrast Limited Adaptive Histogram Equalisation (CLAHE)** [44], a more sophisticated variant of histogram equalisation. CLAHE locally improves contrast by segmenting the image and individually applying histogram equalisation to each segment. Usage of CLAHE resulted in higher eye tracking accuracy in comparison to standard equalisation. The outcome of the operation is a more salient iris (Figure 1(d)).

4.2.2 Eye Tracker Calibration. Calculating eye fixations requires mapping the estimated iris center coordinates to pixel locations on the user's screen. The conversion is accomplished by a mapping function, whose parameters are determined through a calibration procedure (Figure 6). During calibration the user is asked to fixate on 12 additional targets in known positions. These 12 targets are different to the 12 targets fixated during stage 1: For accurate calibration, the procedure is repeated after ROI detection and image enhancement operations. The calibration procedure lasts for about 30 s.

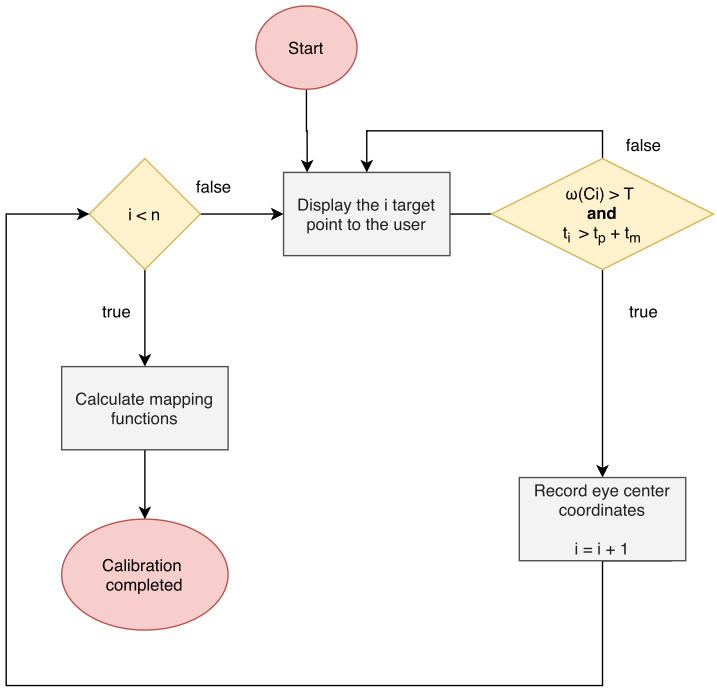

Fig. 6. Calibration procedure flowchart (n=calibration points).

As the viewer fixates on each target $i \in [0, 11]$ with 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 $\vec{s}_i$, resulting in 12 pairs of $\{\vec{s}_i, \vec{e}_i\}$.

Eye tracking accuracy heavily relies on the quality of the calibration procedure. For maximum accuracy all 12 targets must be successfully fixated and recorded (i.e., iris detected with high confidence ($\omega(\vec{e}_i) \geq T$), see Section 4.1.2). Involuntary blinks or saccades can inevitably occur contaminating the results. To attain the highest data quality possible, we adjust calibration timings as follows:

1. Saccades take 200 ms to initiate and 20–200 ms to complete [4]. Eye position recording is paused for $t_p = 500$ ms each time a new target gaze point is presented; this gives the user adequate time to re-adjust their gaze to the new target.

2. Each new target point is presented for at least t_m seconds, set experimentally to $t_m = 1500$ms, enough time for the target to be fixated accurately.

To estimate the gazed point on screen, a mapping function from iris center coordinates to screen coordinates must be defined. It is desirable that the mapping function is minimally affected by calibration errors and geometric distortions between the eye tracker and the screen [38]. For our mobile VR eye tracking we experimented with both linear and non-linear mapping functions. We 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, in addition to being more computationally intensive.

To determine the user's fixation we resort to linear interpolation. A linear mapping function divides the screen into a grid of cells and infer the point of regard by linearly interpolating the coordinates of the grid cells junctions. Once calibration is completed, the user's gaze coordinates on the screen (x_s, y_s) can be obtained as $\vec{s} = f(\vec{e})$, f

Table 1. Execution Time of Processing Steps, Total End-to-end Latency

	Mean (ms)	**SD (ms)**
Image cropping	0.01	0.001
RGB to grayscale	0.38	0.15
Histogram equalisation	0.94	0.16
Image scaling	0.12	0.004
Reflection suppression	0.077	0.014
Hough circle fitting	1.1	0.24
Find best circle	0.074	0.008
Gaze mapping and averaging	0.2	0.02
Camera image acquisition	15.6	4.8
Total end-to-end latency	**18.5**	1.64

the linear mapping function:

$$x_s = f_x(x_e, y_e) \qquad y_s = f_y(x_e, y_e). \tag{3}$$

4.3 Stage 3: Real-time Eye Tracking

Our eye tracking pipeline for mobile VR without modifications or infrared lighting can now track the iris based on new incoming eye frames as captured by the front camera. In each incoming frame (Figure 1(b)), we isolate the eye ROI in which the iris moves (Figure 1(c)), using the window estimated during stage 1. As a result of this step, both the calibration and real-time gaze mapping speed and accuracy are improved, as the algorithm has to search a smaller space to locate the iris, thus reducing the possibility of failed eye detection. To determine the iris center, we apply the same image processing techniques as in Stage 2 to suppress reflections and enhance low image contrast (Figure 1(d)). The customised circle fitting technique can then be applied (Figure 1(e)). Once the iris center has been successfully detected, we convert the iris center coordinates to screen coordinates, using the mapping function determined during the calibration procedure (Stage 2).

Preliminary testing indicated that immediately mapping the estimated iris centers to screen coordinates can exaggerate small tracking errors. As a means of low pass filtering the estimated gaze points, we average the last N_s screen points. We experimented with several N_s values but recommend $N_s = 8$ (Section 5.1.4), resulting 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 lower accuracy of the gaze positions. The performance impact of the described averaging operation is negligible (<0.2 ms).

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. Our computationally efficient algorithm, when run simultaneously with a VR application, requires less than 19 ms per frame to estimate gaze, which corresponds to an upper frequency of about 52 Hz, capped to 30 Hz due to the camera refresh rate. In practice, the maximum gaze prediction rate of 30 Hz may drop momentarily 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. Our system evaluation and user study demonstrated that gaze-tracking delays were rare and imperceptible.

5 EVALUATION OF TRACKER AND APPLICATIONS

We evaluated the performance of our system by conducting two experiments and a use-case study. Experiment 1 investigated the level of accuracy and precision of the estimated gaze locations. Experiment 2 explored the

effectiveness, in terms of task completion time, of gaze-driven interaction using our eye tracking pipeline compared to head-tracked interaction in a VR game. A final use case study 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 measurements can be used to evaluate the validity of the estimated gaze locations of our eye tracking system [18]. 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 [17].

Acceptable levels of accuracy and precision depend on the nature of the eye-tracked application. In our context, we can tolerate larger inaccuracies compared to an eye tracker that is based on specialised hardware, especially in the periphery of vision. Due to the inherent limitations of tracking without IR light, we propose design guidelines for the eye-tracked GUI elements (Section 5.1.6). Given the nature of our setup (smartphone and mobile VR 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 actually fixating to the points as instructed can only *lower* our prediction accuracy measurements and never improve them.

5.1.1 Apparatus and Stimuli. We used an Apple iPhone 6S both for displaying the stimuli and tracking the eye. We used the Shinecon VR headset. We developed an eye tracking experiment that presents 12 red (RGB: 255,0,0) 20×20 pixel target points on a 3×4 grid on black background, one at a time. The luminance of the screen was at its highest level (500 cd/m^2). The target points were 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., not 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 about the experimental procedure during which they fixated at the set of 12 target points displayed in a specific order one by one. Each target point is presented for $t = 3$ s, including a $t = 1$ s data recording pause, so that users have enough time to adjust their gaze. Target locations can be seen as blue asterisks in Figure 7. Each target point is displayed three times per user for validity. As the users fixated 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 and Results. Accuracy and precision data across all users are plotted in Figures 7 and 8. We calculated the average accuracy and precision for each of the 12 targets and the mean accuracy averaged 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 0.06 cm on screen). In the y-axis the mean accuracy across all targets was 1.86° (12.5 pixels or 0.1 cm on screen). Regarding individual targets, average target 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). We also show mean, minimum and maximum gaze estimation error in degrees of visual angle per subject (Table 3). A slightly different amount of total fixations were recorded for each user. To calculate

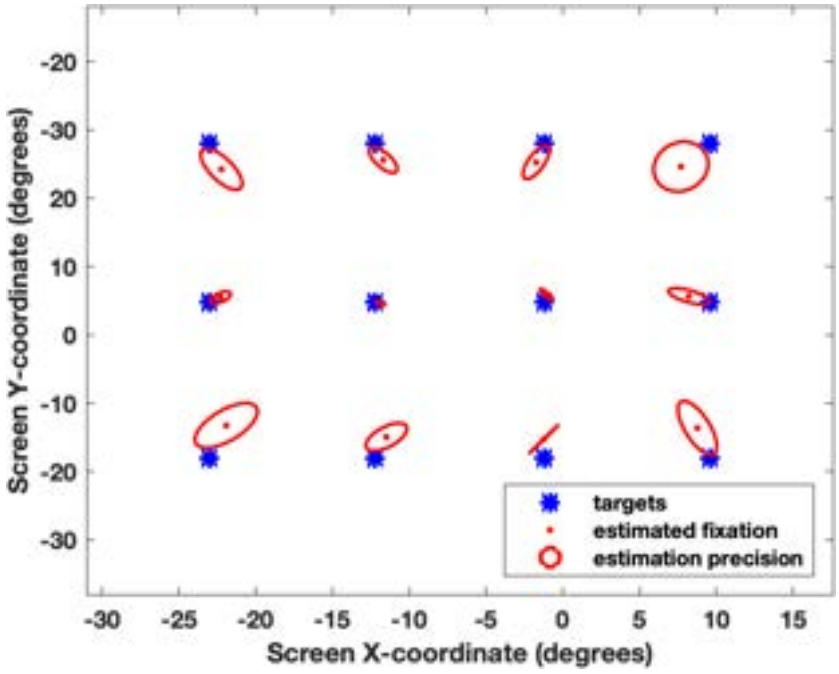

Fig. 7. Accuracy and precision of the estimated gaze points in degrees of visual angle. Blue asterisks indicate gaze targets. Red dots denote the mean estimated position. Red ellipses denote root mean square error for the estimated gaze positions.

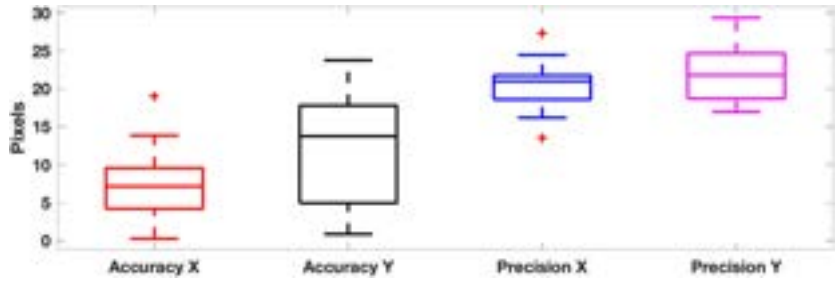

Fig. 8. Accuracy and precision of the estimated gaze points in pixels. On each box, the central mark indicates the median. Bottom and top edges indicate the 25th and 75th percentiles. Higher is worse. The whiskers extend to the most extreme data points not considered outliers. Outliers are plotted using the "+" symbol.

Table 2. Mean, Minimum, Maximum Gaze Estimation Error (Averaged across All Targets and Subjects) in Pixels, Degrees of Visual Angle, and cm on Screen

	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 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

visual angles we first estimated the iPhone's pixel size in meters as 0.0254/326, since the iPhone 6S display has a resolution of 326 ppi. Then used the standard formula:

$$visualAngle = 2 * arctan\frac{\#pixels * pixelSize}{2 * distanceToScreen} = 2 * arctan\frac{\#pixels * 7.79 * 10^{-5}}{0.06} = 2 * arctan(\#pixels * 0.0012986). \quad (4)$$

Sample averaging did not have a significant effect on precision. In our 30-Hz 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, then 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 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 [36, 37]. 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 7). 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° [33]. 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 (2 × 2.5 GHz + 4 × 1.6 GHz)) in comparison to the iPhone 6s employed (dual-core 1.84 GHz). Image acquisition performance is marginally improved by 2.73 ms (12.87 ms vs. 15.6 ms). Overall end-to-end latency is 5 ms lower on the iPhone Xs. These measurements were acquired with the same settings for both phones and 360 × 480 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 and Content Design Guidelines. Tracking accuracy and precision directly relate to the acceptable 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 expected fixations to fall inside a target region [9]. 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 99 px (≈0.8 cm at 326 ppi) and 125 px (≈1 cm at 326 ppi) respectively in the x and y-axes, according to [9]

$$S = 2 \times (lowestAccuracy + 2 \times precision), \tag{5}$$

$$S_{width} = 2 \times (23.7 + 2 \times 19.3) \approx 125, \tag{6}$$

$$S_{height} = 2 \times (10.9 + 2 \times 19.3) = 99. \tag{7}$$

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.

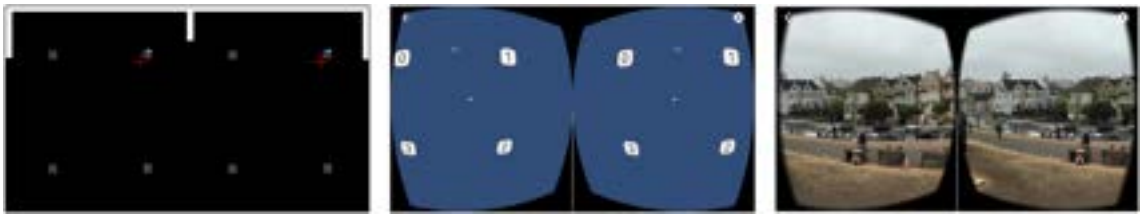

Fig. 9. Experiment 2–eye tracking (left): VR application employing our eye tracking system. Red crosses are estimated gaze points. Thin bezel of light on upper edge of screen. Experiment 2–head tracking (middle): Users position the "reticle" visible as a white dot on GUI elements by rotating/tilting their head. Use case study (right): Eye-tracked 360 VR panorama.

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 9.

The users were informed that they would be wearing a HMD and were presented with a set of numbered buttons that they should memorize. Each button contained a randomly generated number, present for $t = 2$ s. 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 1,000 ms in both conditions ($t \geq 1$ s) to eliminate false selections, also known as the "Midas touch problem" [21].

5.2.2 Participants. Thirteen 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 Experiment 1 (Section 5.1.2).

5.2.3 Results and 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 = 6.9 s, SD = 2.82 s) and head-tracked (M = 6.88, SD = 0.56 s) 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 and 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. Three users with light eye color (two green-eyed and one blue-eyed) yielded lower eye-tracking accuracy than users with dark eye color. One 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-axis 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 [23].

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. However, in cases when the displayed content becomes very bright (see Figure 10), 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 not include large areas of very high luminance. To keep obtrusive reflections to a minimum, avoiding extremely 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. We could then initiate a swift re-calibration procedure to maintain eye tracking accuracy. Additional future work could evaluate the potential benefits of dynamically

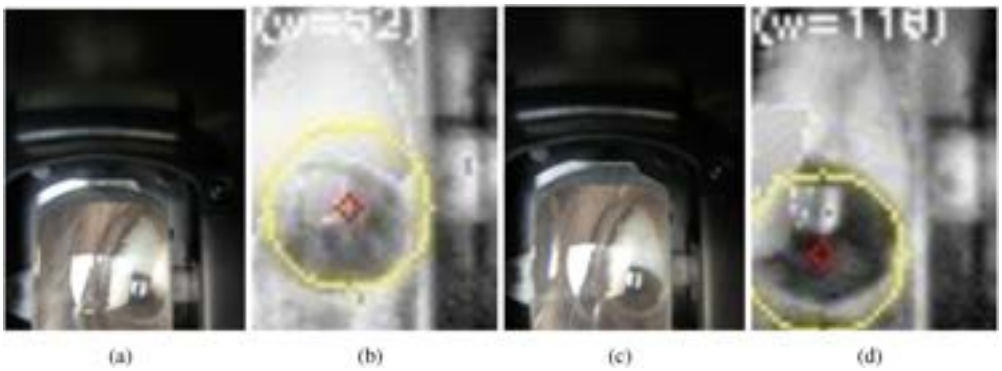

Fig. 10. Two examples of sub-optimal content that may decrease accuracy, such as images with high intensity colors and high contrast shapes used as panoramas (see Section 5.3). Such content inevitably produces obtrusive reflections in the raw eye images ((a) and (c)), reducing eye saliency and thus hampering accurate gaze estimation ((b) and (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 ($\omega = 52$) ((a) and (b)).

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, then T can be increased so that sub-optimal fits are rejected and only exceptionally accurate fits are kept (thus further improving overall accuracy). However, 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), then 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 dependent not only on eye morphology but also on the displayed content.

REFERENCES

[1] [n.d.]. OpenCV Library. Retrieved May 1, 2020 from https://opencv.org/.
[2] [n.d.]. XCode. Retrieved September 9, 2019 from https://developer.apple.com/xcode/.
[3] Karan Ahuja, Rahul Islam, Varun Parashar, Kuntal Dey, Chris Harrison, and Mayank Goel. 2018. EyeSpyVR: Interactive eye sensing using off-the-shelf, smartphone-based VR headsets. *Proc. ACM Interact. Mobile Wearable Ubiq. Technol.* 2, 2 (2018), 57.
[4] Robert W. Baloh, Andrew W. Sills, Warren E. Kumley, and Vicente Honrubia. 1975. Quantitative measurement of saccade amplitude, duration, and velocity. *Neurology* 25, 11 (1975), 1065–1065.
[5] Gary Bradski and Adrian Kaehler. 2008. *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media, Inc.
[6] Juan J. Cerrolaza, Arantxa Villanueva, and Rafael Cabeza. 2012. Study of polynomial mapping functions in video-oculography eye trackers. *ACM Trans. Comput.-Hum. Interact.* 19, 2 (2012), 10.
[7] Panagiotis Drakopoulos, George Alex Koulieris, and Katerina Mania. 2020. Front camera eye tracking for mobile VR. In *Proceedings of the 2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW'20).* IEEE, 643–644.
[8] Andrew T. Duchowski. 2018. Gaze-based interaction: A 30 year retrospective. *Comput. Graph.* 73 (2018), 59–69.
[9] Anna Maria Feit, Shane Williams, Arturo Toledo, Ann Paradiso, Harish Kulkarni, Shaun Kane, and Meredith Ringel Morris. 2017. Toward everyday gaze input: Accuracy and precision of eye tracking and implications for design. In *Proceedings of the 2017 CHI Conference Human Factors in Computing Systems.* ACM, 1118–1130.
[10] Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: A paradigm for model fitting with applications to image analysis & automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.
[11] Andrew Fitzgibbon, Maurizio Pilu, and Robert B. Fisher. 1999. Direct least square fitting of ellipses. *IEEE Trans. Pattern Anal. Mach. Intell.* 21, 5 (1999), 476–480.
[12] Wolfgang Fuhl, Thomas Kübler, Katrin Sippel, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2015. Excuse: Robust pupil detection in real-world scenarios. In *Proceedings of the International Conference on Computer Analysis of Images and Patterns.* Springer, 39–51.
[13] Wolfgang Fuhl, Thiago C. Santini, Thomas Kübler, and Enkelejda Kasneci. 2016. Else: Ellipse selection for robust pupil detection in real-world environments. In *Proceedings of the 9th Biennial ACM Symposium on Eye Tracking Research & Applications.* ACM, 123–130.

[14] Wenshuo Gao, Xiaoguang Zhang, Lei Yang, and Huizhong Liu. 2010. An improved Sobel edge detection. In *Proceedings of the 2010 3rd International Conference on Computer Science and Information Technology*, Vol. 5. IEEE, 67–71.
[15] Scott W. Greenwald, Luke Loreti, Markus Funk, Ronen Zilberman, and Pattie Maes. 2016. Eye gaze tracking with google cardboard using purkinje images. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*. ACM, 19–22.
[16] Hiroyuki Hakoda, Wataru Yamada, and Hiroyuki Manabe. 2017. Eye tracking using built-in camera for smartphone-based HMD. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST'17)*. Association for Computing Machinery, Inc, 15–16.
[17] Kenneth Holmqvist, Marcus Nyström, Richard Andersson, Richard Dewhurst, Halszka Jarodzka, and Joost Van de Weijer. 2011. *Eye Tracking: A Comprehensive Guide to Methods and Measures*. Oxford University Press, Oxford.
[18] Anthony J. Hornof and Tim Halverson. 2002. Cleaning up systematic error in eye-tracking data by using required fixation locations. *Behav. Res. Methods Instrum. Comput.* 34, 4 (2002), 592–604.
[19] Michael Xuelin Huang, Jiajia Li, Grace Ngai, and Hong Va Leong. 2017. Screenglint: Practical, in-situ gaze estimation on smartphones. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 2546–2557.
[20] John Illingworth and Josef Kittler. 1988. A survey of the Hough transform. *Comput. Vis. Graph. Image Process.* 44, 1 (1988), 87–116.
[21] R. Jakob. 1998. The use of eye movements in human-computer interaction techniques: What you look at is what you get. In *Readings in Intelligent User Interfaces* (1998), 65–83.
[22] Amir-Homayoun Javadi, Zahra Hakimi, Morteza Barati, Vincent Walsh, and Lili Tcheang. 2015. SET: A pupil detection method using sinusoidal approximation. *Front. Neuroeng.* 8 (2015), 4.
[23] Brendan John, Sanjeev Koppal, and Eakta Jain. 2019. EyeVEIL: Degrading iris authentication in eye tracking headsets. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*. 1–5.
[24] Anuradha Kar and Peter Corcoran. 2017. A review and analysis of eye-gaze estimation systems, algorithms and performance evaluation methods in consumer platforms. *IEEE Access* 5 (2017), 16495–16519.
[25] Pawel Kasprowski, Katarzyna Harężlak, and Mateusz Stasch. 2014. Guidelines for the eye tracker calibration using points of regard. In *Information Technologies in Biomedicine*, Vol. 4. Springer, 225–236.
[26] Moritz Kassner, William Patera, and Andreas Bulling. 2014. Pupil: An open source platform for pervasive eye tracking and mobile gaze-based interaction. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM, 1151–1160.
[27] Dmytro Katrychuk, Henry K. Griffith, and Oleg V. Komogortsev. 2019. Power-efficient and shift-robust eye-tracking sensor for portable VR headsets. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*. 1–8.
[28] George Alex Koulieris, Kaan Akşit, Michael Stengel, R. K. Mantiuk, Katerina Mania, and Christian Richardt. 2019. Near-eye display and tracking technologies for virtual and augmented reality. In *Computer Graphics Forum*, Vol. 38. 493–519.
[29] George Alex Koulieris, George Drettakis, Douglas Cunningham, and Katerina Mania. 2016. Gaze prediction using machine learning for dynamic stereo manipulation in games. In *Proceedings of the 2016 IEEE Virtual Reality (VR'16)*. IEEE, 113–120.
[30] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, and Antonio Torralba. 2016. Eye tracking for everyone. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2176–2184.
[31] Dongheng Li, David Winfield, and Derrick J. Parkhurst. 2005. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *Proceedings of the 2005 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition Workshops (CVPR'05)*. 79–79.
[32] Tianxing Li, Qiang Liu, and Xia Zhou. 2017. Ultra-low power gaze tracking for virtual reality. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. 1–14.
[33] Denis Pelisson and Alain Guillaume. 2009. *Eye-Head Coordination*. Springer, Berlin, 1545–1548. DOI: https://doi.org/10.1007/978-3-540-29678-2_3257
[34] Thiago Santini, Wolfgang Fuhl, and Enkelejda Kasneci. 2018. PuRe: Robust pupil detection for real-time pervasive eye tracking. *Comput. Vis. Image Understand.* 170 (2018), 40–50.
[35] Nikolaos Sidorakis, George Alex Koulieris, and Katerina Mania. 2015. Binocular eye-tracking for the control of a 3D immersive multimedia user interface. In *Proceedings of the 2015 IEEE 1st Workshop on Everyday Virtual Reality (WEVR'15)*. 15–18.
[36] Alexandra Sipatchin, Siegfried Wahl, and Katharina Rifai. 2020. Accuracy and precision of the HTC VIVE PRO eye tracking in head-restrained and head-free conditions. *Invest. Ophthalmol. Vis. Sci.* 61, 7 (2020), 5071–5071.
[37] Alexandra Sipatchin, Siegfried Wahl, and Katharina Rifai. 2020. Eye-tracking for low vision with virtual reality (VR): Testing status quo usability of the HTC Vive Pro Eye. *bioRxiv* (2020).
[38] Dave M. Stampe. 1993. Heuristic filtering and reliable calibration methods for video-based pupil-tracking systems. *Behav. Res. Methods Instrum. Comput.* 25, 2 (1993), 137–142.
[39] Lech Świrski, Andreas Bulling, and Neil Dodgson. 2012. Robust real-time pupil tracking in highly off-axis images. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 173–176.
[40] Tsuji and Matsumoto. 1978. Detection of ellipses by a modified Hough transformation. *IEEE Trans. Comput.* C-27, 8 (1978), 777–781.

[41] Erroll Wood and Andreas Bulling. 2014. Eyetab: Model-based gaze estimation on unmodified tablet computers. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. 207–210.
[42] Zhiwei Zhu and Qiang Ji. 2004. Eye and gaze tracking for interactive graphic display. *Mach. Vis. Appl.* 15, 3 (2004), 139–148.
[43] Zhiwei Zhu, Qiang Ji, and Kristin P Bennett. 2006. Nonlinear eye gaze mapping function estimation via support vector regression. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06)*, Vol. 1. IEEE, 1132–1135.
[44] Karel Zuiderveld. 1994. Contrast limited adaptive histogram equalization. In *Graphics Gems IV*. Academic Press Professional, 474–485.

Received July 2020; revised February 2021; accepted February 2021