An Embedded and Real-Time Pupil Detection Pipeline

Ankur Raj
Ubiquitous Computing
University of Siegen
Siegen, Germany

Diwas Bhattarai
Ubiquitous Computing
University of Siegen
Siegen, Germany

Kristof Van Laerhoven
Ubiquitous Computing
University of Siegen
Siegen, Germany

ABSTRACT

Wearable pupil detection systems often separate the analysis of the captured wearer's eye images for wirelessly-tethered back-end systems. We argue in this paper that investigating hardware-software co-designs would bring along opportunities to make such systems smaller and more efficient. We introduce an open-source embedded system for wearable, non-invasive pupil detection in real-time, on the wearable, embedded platform itself. Our system consists of a head-mounted eye tracker prototype, which combines two miniature camera systems with Raspberry Pi-based embedded system. Apart from the hardware design, we also contribute a pupil detection pipeline that operates using edge analysis, natively on the embedded system at 30fps and run-time of 54ms at 480x640 and 23ms at 240x320. Average cumulative error of 5.3368px is found on the LPW dataset for a detection rate of 51.9% with our detection pipeline. For evaluation on our hardware-specific camera frames, we also contribute a dataset of 35000 images, from 20 participants.

KEYWORDS

Pupil Detection, Embedded Systems, Edge Analysis, Near Eye Dataset

1 INTRODUCTION

Eye tracking applications have generated several commercial products [18, 37], initially for complementing user surveys or improving designs in shopping centres, products and advertisement placements [10]. In automobiles it has similarly been used to measure drivers' fatigue [14, 17]. Eye tracking is an especially important modality, as it has become ubiquitous in commercial products and serves many applications: it has been used to study users' visual attention [2], sleep detection and emotional state, and shows immense promise in Augmented Reality (AR) and Virtual Reality (VR) applications for foveated rendering [31], i.e., increasing the amount of detail in images based on the user's focus or fixation points. It includes behavioural information like visual-attention [4, 42], emotional state [23] as well as medical information. Eye tracking products also increasingly become smaller and more efficient, as they are integrated in mobile and wearable devices as an additional biometric, for instance for unlocking a smartphone [30]. Crucial to any eye tracking device is the pupil detection step, which is needed before performing other routines such as eye tracking, gaze estimation, and developing interfaces based on eye movement.

Pupil detection has been researched extensively, leading to many commercial eye trackers being available [18, 37] nowadays. These provide their end users with APIs (Application Programming Interface) for collection of data and performing eye and gaze tracking. Such eye trackers tend to be expensive and platform-specific, focusing strongly on high-end camera features to cater for all potential users. We argue in this paper that for many applications, the resulting high-speed and high-resolution cameras might prevent a

light-weight and energy-efficient pupil detection pipeline. For such cases, we propose an approach that uses a miniature and low-cost hardware design, which can be replicated by others with minimal effort. For the software design, we developed an embedded and real-time pupil detection pipeline based on edge analysis, and contribute a dataset of near eye images taken from this hardware for validation. We thus present a comprehensive hardware-software open-source framework that allows researchers to holistically design their own pupil tracker. The major contributions of our paper can thus be summarized as follows:

- We present an embedded hardware design for embedded eye tracking, based around commercial and widely available camera modules and a Raspberry Pi platform.
- We developed an open-source pupil detection pipeline that can run natively on the Raspberry Pi, in real-time.
- We created a benchmark dataset of near-eye images with our hardware design, from people of different ethnicity, gender, eye colour and age.
- We evaluate our pipeline on this dataset and a public one [38] for performance in detection speed and accuracy on an embedded platform.

Figure 1: The 3D-printed hardware setup connects two cameras through USB and CSI, along a LED power delivery using GPIO to the Raspberry Pi Model 4 B. The bottom-right image shows the front view of the hardware while worn.

The remainder of this paper is structured as follows: In section 2, the state-of-the-art in this field is described, where we focus particularly on existing detection methods and different hardware setups, as well as on state-of-the-art datasets used in eye tracking. We position our contributions among these. In the following section 3, the design methodology for the hardware and the detection method that we designed to work on the embedded system are presented, as well as our strategy for the creation of a near eye benchmark dataset. This is followed by the evaluation (section 4), which contains the validation of our detection pipeline on the proposed hardware, and also contains the performance of the pupil detection on an existing state-of-the-art public dataset. We finally conclude this paper in section 5 where we discuss the main results and give an outlook on this research.

2 RELATED WORK

Eye tracking and pupil detection depends on various components. In this section, we introduce the closest areas in eye tracking research, particularly the hardware elements that capture images of the eye, the detection pipeline to analyze those images, and available datasets that are used to measure the performances of the detection strategies. Several pupil detection and gaze estimation methods have been introduced over these past years and numerous researchers have used different methodologies, ranging from blob detection [15], edge analysis [6, 18, 34] to machine learning [7, 19, 20] based approaches. We focus here on pupil detection as a first step towards creating an eye tracking and gaze estimation setup. Prior research has focused on developing a fast and accurate pupil detection pipeline, that could also work in a variety of challenging scenarios.

2.1 Eye Tracking Hardware Designs

Any pupil detection strategies are highly dependent on the hardware being used for capturing high resolution images of the eye, especially in different lighting conditions, frame rate, and position of the camera. Due to these parameters, the final detection method heavily relies on the selection of the hardware setup. Such setups that are commonly used for pupil detection can be divided into two sub-categories; (1) *Remote Eye Setup* and (2) *Near Eye Setups* or *Head Mounted Camera* systems.

Remote Eye Hardware setups have the camera situated away from the participant. It can follow both a multiple camera [1, 28] or a single camera [12, 26] design. Free movement of the user's head is allowed within a fixed range. The camera images of the eye, head and posture of the whole body is captured with these systems. They provide high resolution cameras with high frame rates (>500 Hz), with some commercial eye tracking systems even reaching frame rates of more than 1000 Hz [36]. These high frame rates are not usually observed with the near eye setups. The remote setups are useful in making inferences about the behaviour of the user, like visual-attention [2], drowsiness, and fatigue [13, 32, 39], which could be essential in sending alerts if an automobile driver becomes drowsy [13]. Hardware designed by researchers includes custom remote eye tracking setups such as [11, 12, 26], mostly due to its own feature requirements and the expensive commercial alternatives. A recent open-source setup is RemoteEye [12], which

provides a high-speed eye tracker that could achieve frame rates of more than 500 *fps* [12] during pupil detection, and provides a gaze estimation accuracy of 0.98 degrees, on average [12]. It provides a low-cost alternative to commercial high speed remote camera systems.

Near Eye Hardware Setups or head mounted camera system uses an eye tracker headset with one (monocular) or two (binocular) eye cameras and/or a world camera. It can be enclosed e.g. VR setups or can be open e.g. Google Glass or Microsoft Holo lens. Additionally, it has numerous infrared LEDs to provide consistent lighting and contrast between pupil and iris. The setups also use USB or other wired/wireless connectivity. The near eye tracker is mobile and could capture higher detail images of the eye. Researches have made their own near eye trackers from commercial components. The authors [27] modified safety glasses to attach analogue cameras and IR LEDs to create an invasive eye tracking system, where the point of gaze is estimated offline. openEyes [22] is an open source eye tracking contribution that uses two IEEE-1394 web cameras (640x480 @ 30 Hz) for synchronised frames and Starburst detection algorithm. The authors [33] have extended the framework with the ability to seamlessly switch from watching the pupil/iris border in bright light to tracking the iris/sclera boundary (limbus) in low light. Researches have disassembled web camera like Microsoft LifeCam VX-1000 to create a low-cost DIY eye tracker [25]. The Newer eye trackers include frames that are 3D printed [3, 24] and use strobo-scopic lighting [3]. Furthermore, research has also been performed on a low-cost hardware to produce a smart eye tracking system [16], that connects with a wheelchair to assist elderly people in gaining mobility via smart commands. Another study implementing an embedded system (Raspberry Pi) in eye tracking is from the author(s) [5], where the open-source gaze tracking software (*openGazer*) [41] has been extended upon and tested for evaluation on an embedded device. Other well-known commercial products for near eye tracking are from Pupil Labs [7, 18] and Tobiipro [37].

As suggested by the authors of [12], one disadvantage of the near eye system is lower frame rates (30Hz - 120Hz), which hinders in the application of robust saccade and micro-saccade detection. So far, only limited research exist in using a head mounted, multi-camera eye tracking system incorporated within a fully embedded system, whereas this field is particularly interesting because it would grant more mobility to the end user, as the device would be portable enough that it can be carried along or be worn by the user.

2.2 Pupil detection methods

Historically, a majority of pupil detection methods employ a range of well-known computer vision techniques. They predominantly rely on edge detection methods, thresholding, and morphological filtering to locate the pupil in the current image. Recently, more end-to-end machine learning based approaches have gained in popularity and are also being used for pupil detection. Any of these methods have to face similar challenges while performing pupil detection, however, with pupil detection being mostly hampered by challenges that include:

- reflection in eye images due to external lighting
- blury images due saccade and micro-saccade
- poor contrast between iris and pupil

- obstruction/occlusion in pupil view due to eyebrows, eyelid and/or eye wear.

A well known computer vision based pupil detection pipeline is proposed by Lech Świrski et al. [35]. In this pipeline, the input image is first convolved with a HAAR-like feature to have a maximum response within the image. This convolution operation is repeated for different user-defined radii sizes. The radii with maximum response is used as a pupil region. The pupil is identified in this region by k-means clustering on the histogram, followed by edge detection using canny filter to finalize the boundary. Lastly, a Random Sample Consensus (RANSAC) ellipse fitting is used to detect the pupil. PupilLabs [18] follow an approach for pupil detection that is similar to the pipeline used in article [35]. After ROI estimation and edge detection, filtering of edges is performed based on histogram analysis and removal of any spectral reflection. The contours are then connected, and ellipse fitting is performed on the identified pupil.

In another study [15] a detector using threshold operation has been proposed. After the identification of pupil segments, a convex hull operation is used to enclose the segments and the enclosure is used to identify the centre of pupil. ExCuSe [6] is yet another well known state-of-the-art method for pupil detection in real world scenarios. It uses two different approaches to find the pupil, being edge detection and thresholding. If there is a peak in the bright region of the histogram analysis, detection can be performed using edge analysis [6, 8]. PuRe (or Pupil Reconstructor) is also an edge-based detector that uses edge segment selection and conditional segment combination schemes to perform pupil detection [34]. Additionally, it provides a confidence measure (ψ) for the selected ellipse.

Machine learning based methods are known to be used in the research of pupil detection and gaze estimation. The ML based estimators use features for training a neural network model to locate the pupil and determine the gaze. Convolutional Neural Network (CNN) and VGG are the commonly used ML models. A CNN based ML model is NVGaze [19], it provides accurate pupil detection and gaze estimation on synthetic as well as real near eye images. It ihas little modifications, depending on the application. The model includes 6 - 7 convolutional layers each with 2×2 stride, and dropout layers after each convolutional layer, and no padding or pooling. PupilNet is another well known CNN based detector [7]. It uses *dualCNN* pipeline, where one network coarse estimates the location of pupil and the other network fine positions on this estimate to have a final localisation of pupil. Estimation for occluded eye images due to eyelids, wearables or even reflections make the pupil detection a difficult task. EllSeg framework [20] also offers a method to segment the entire elliptical structure containing the iris and pupil, hence providing pupil detection on highly occluded images. The EllSeg framework can be used with any encoder-decoder architecture, as well.

2.3 Datasets: Pupil & Gaze Estimation

With the rise in augmented reality and virtual reality applications and machine learning (ML)-based estimators, there are now more publicly available datasets, containing images or videos of Near Infrared (NIR) [38], remote eye images [21] and simulated eye models

[19], with manual annotation of pupil position. Table 1 contains some of the most well-known state-of-the-art datasets for pupil detection and gaze estimation to provide an overview of their features.

LPW or Labelled pupil in the wild [38] contains 66 high quality videos from 22 participants, where the videos were recorded using a head mounted camera system [18]. The dataset is collected under varying lighting conditions, both indoors and outdoors, at $95 fps$. The pupil location is manually labelled inside the dataset. Gaze estimation is not part of the dataset, but it contains pupil centre location annotation for each frame.

NVGaze [19] contains near eye images in infrared lighting. It has both synthetically generated data (2M images at 1280×960) and real-world data (2.5M images at 640x480) collected from 30 subjects. Synthetic data was rendered from 3D models [40] under active illumination with 4 IR LEDs, with each image labelled with 3D eye location, 2D pupil location and 2D gaze vector. The synthetic dataset contains exact segmentation for sclera, skin, pupil and iris. The real-world data consists of binocular images captured from a 120Hz camera. The data is collected under two separate conditions, with users performing acuity tasks. One using VR hardware with IR illumination, and the other using a head mounted setup for AR with varying infrared intensity, emulating a real world scenario with varied lighting conditions.

GazeCapture is a dataset that is used for gaze estimation [21]. It is a crowd-sourced project where users capture their images using mobile devices like phones and tablets. The subjects use the mobile application to capture images of their face while they look at markers on the screen. The images are captured in a variety of lighting conditions, making it a diverse dataset. There are more than 2.5M images in the dataset. Since the images are taken in different lighting conditions and from a remote device (mobile phone or tablet), near eye images of the pupil are not part of this dataset. The dataset also contains motion data from the mobile device (accelerometer, gyroscope, manometer) captured at 60Hz.

Open EDS [9] contains near eye infrared images in a controlled environment. The images were captured using a VR headset that has been modified to capture images in 200 Hz. The dataset contains binocular images with a subset of images that have pixelated annotation of eye regions (eyelid, pupil, iris). These annotations were made using ellipse selector as well as manual annotators. Since this dataset contains not only the centre of pupil but also annotation for each region of eye on real-world participants, it becomes a novel dataset for training ML and non-ML based detectors.

3 METHODOLOGY

The eye tracking hardware is the central component of this paper, which provides near eye infrared images in high quality that would be used for our pupil detection pipeline. It is important to have a low-cost and open-source hardware platform that makes it accessible for any researchers to incorporate and eye tracking in their projects. We have created a custom monocular hardware design that has an eye camera and a world camera. The work from the pupil labs [18] and previous eye trackers [22, 27] served as an inspiration for this hardware design.

Table 1: State-of-the-art publicly available datasets, the lighting conditions within these datasets can be varied i.e. indoors and outdoors. For article [9] a controlled environment is created to capture the images inside a VR headset.

Dataset	Number of images	Lighting conditions	Hardware used	YOP	Gaze Direction
GazeCaptue [21]	2.5M	Indoors and outdoors	Mobile/tablet camera	2016	yes
NVGaze [19]	2.5M real and 2M synthetic	Indoors with varied lighting	VR headset and a head mounted camera 640x480 @ 30 Hz and 1280x960	2019	yes
Labelled Pupils in the wild [38]	130,856	Indoors and outdoors	Head mounted camera; 480x640 @ 95 Hz	2016	no
OpenEDS [9]	356,649	Indoors, in a controlled environment	VR headset; 400x640 @ 200 Hz	2019	no

The eye tracker can be divided into 3 major components, the frame, the camera systems & their connection, and the compute-device. Except the compute device, all the components are attached to the eye tracker frame. The frame of the eye tracker draws inspiration from safety eyewear and DIY eye tracker from Pupil Labs, and can be worn as an accessory. The eye tracker headset/frame is designed in a 3D modelling software and is 3D printed using PLA as a print material. Individual camera support holders are printed separately and joined together via snap connectors. The eye tracker is monocular and the frame incorporates a holder for the world camera (fig 1).

The camera system can be further subdivided into the eye camera and world camera. The eye camera is a Raspberry Pi Zero camera with an OmniVision OV5647 camera sensor from Sparkfun. It is a fixed focus 5 MP camera that supports VGA(640x480) @ 90Hz and 720 p @ 60 Hz. The focus of the camera is manually modified to have it capture near images of the eye. The eye camera (Pi Zero NoIR) supports taking images in infrared lighting, as it is build without the infrared blocking filter and an infrared LED array is created with SFH 4050-Z from OSRAM Opto Semiconductors. The world camera (Logitech HD-C615) takes images of the subject's Field of View (FOV), this commercial camera needs to be disassembled first to be attached with our headset. It supports 1080p/30fps and 720p/30fps resolutions.

The central processing module is a Raspberry Pi model 4B, a miniature and low-cost single-board computer made by from Raspberry Pi foundation that runs on a Debian based Linux distribution (Pi OS bullseye). This board is used to perform pupil detection locally, on the . It has an ARM Cortex A72 64-bit SoC @ 1.5GHz and 4 GB of RAM. The compute-device provides the necessary interface to our cameras and LED array. The eye camera is connected to a camera adapter to be later connected to the compute device with a CSI cable, the world camera is connected with a USB cable to the compute-device, and the IR-emitter is controlled with GPIO connections to the Raspberry Pi.

The hardware is used to acquire the near eye images in IR lighting, with the initial constraint of having a detection method to be able to run on our embedded device. We wanted our detection strategy to have as few components as possible, such that it would not be computationally expensive and would not stall the processor of the Raspberry Pi. This motivated us to create a custom ellipse fitting method, which is not computationally expensive and also provides detection of pupil in a variety of edge cases. The ellipse fitting methods use edge detection to find contours in the image, and

from these contours the contour representing the pupil is identified and segmented. Figure 2 shows our pipeline components, which is similar to one presented by the authors of [6] and relies on edge features of the images to find the pupil in the initial image. The pupil is assumed to be dark and have a distinguishing boundary as compared to the iris when illuminated by the infrared LED lighting.

Preprocessing. The initial image is captured in infrared lighting of resolution 480x640 or 240x320 px. This image is cropped for the Region of Interest (ROI), and contains the region of eye including the eyebrows and eyelashes. The ROI cropped image is converted to greyscale and is blurred using a median blur filter with a fixed kernel size (K_{blur}). Canny edge detection with intensity threshold (T_{canny}) is performed to obtain the edges of the image. A morphological open operation is used to reduce the intensity of the eyelashes [35].

It is important to note at this poit that the choice of the canny edge detector intensity (T_{canny}) and median blurring (K_{blur}) could lead to images with various goodness in edge response. It is therefore crucial to find a set of T_{canny} & K_{blur} values that would produce the best identifiable edges for pupil. This is investigated in section 4. Once we have an image with clearly identifiable features for pupil, morphological operations are performed to find which edges could correspond to pupil.

Filtering contours for pupil detection.

(1) After the determination of initial edges in the frame with preprocessing, the contours inside the frame are identified and filtered to correspond to the pupil. For this, an estimation of the closed area of the contour is made using convex hull operation. This ensures that, only close contours are considered as a prospective pupil, hence, straight line edges are discarded.

(2) The area of the closed contour is used as a filtering strategy, since the pupil size in a frame is usually found to be constant or lie in a range (*MAX_PUPIL_SIZE* & *MIN_PUPIL_SIZE*). Based on this, we obtain a list of closed contours that could correspond to pupil.

(3) Filtering is then performed based on circularity of the closed contour. The eye-camera is in front of the subject's eye, leading to the pupil appearing circular. Therefore, we can filter the contours which do not lie above a circularity value ($T_{circularity}$). The circularity is calculated by formula 1.

$$circularity\_hull = \frac{(4 * \pi * area\_hull)}{circumference\_hull^2} \quad (1)$$

Table 2: List of components in the Eye Tracker with all the component being available commercially, where the headset is created using a custom design, that could accommodate the camera system. The 3D model design of the headset can be found in this repository : https://github.com/anonymized.edu/pupiltracker

Components	Product Name	Specifications
Eye Camera	Pi Zero Camera NoIR	5MP VGA(640x480) @ 90Hz and 720 p @ 60 H
World Camera	Logitech HD-C615	1080p/30fps and 720p/30fp UVC compliant
Compute Embedded Device	Raspberry Pi model 4 B	ARM Cortex A72 64-bit SoC @ 1.5GHz, 4 GB RAM
Headset	Custom 3D printed design	
Emitter IR-LEDs	OSRAM SFH 4050-Z	

Figure 2: Our pupil detection pipeline, which is implemented in Python3 and makes extensive use of the OpenCV library [29], is made available open-source. It uses ellipse fitting, where the input image is capture in infrared lighting, and confidence on the parameter selection for canny (T_{canny}) and blurring (K_{blur}) is performed using the parameter estimation routine (sec 4.2).

With these operations a closed contour is identified, that lies within the estimated area of pupil, and is above a circularity threshold value. Now an ellipse can be fitted onto the identified closed contour. Estimation of the centre of the pupil can be made on the selected contour.

The third component of a detection system is the dataset of images, which can be used for analysis of different detection methods and can be tested with different computer architectures. It is common in the research area of eye tracking to create a dataset specific to the hardware configuration [19, 38]. Since our DIY hardware is made from off-the-shelf components, there is no publicly available dataset that is specific to our camera selection. The images are obtained in two different resolutions, i.e. 480x640 and 240x320 px. The lower resolution images are useful when we would like to have a faster detection system, as it reduces the IO operation time. The higher resolution images are useful when we would want to have more details on our images, i.e. edges are more prominent. All the images for the dataset are taken in indoor conditions with an open window and/or incandescent lighting in the room. The display/monitor is also a source of light during this experiment. The setup represents a normal indoor condition where there could be light coming from different light sources, instead of a controlled environment.

Two different approaches are taken for data collection. First, a *fixed marker location* where positional markers are added to a screen, the participants are asked to look at fixed positional markers on the display. A constant distance of 60 cm from the screen is maintained while performing the experiment. Second, *free movement* where the participants are asked to perform a clockwise/counterclockwise movement of the eye, maintain gaze towards different objects. The participants maintained a fixed head pose while they performed both the experiments.

In Dataset Marker Location, fixed positional markers are displayed on a monitor of resolution 1920x1080 px. The participants are asked to look at these marker locations sequentially (figure 3). For each marker location, 30 images of the participant's eye are captured. Since the participants are told to look at the marker locations, there are only a few blinks while performing the experiment. And the participant maintained a constant gaze. In total 150 images are captured per session of the data collection. The participants performed one experiment for the higher resolution (480x640 px), and then repeated the same procedure for a lower resolution (240x320 px).

Dataset Free Movement is the second data collection procedure, the participants are not asked to look at fixed markers, but instead, they are encouraged to look in different directions while maintaining a fixed head position. The participants rolled their eyes in clockwise and counterclockwise, moved their eye randomly, and sometimes even fixated on different objects in the room. The aim of this method is to get more edge case data, where the participants are not bound to a fixed location marker or even keep their eye open, during the entire length of the experiment. The data collection is also performed for two different resolutions (640x480 and 240x320 px), and 300 images of the eye are taken in each session. The *free_movement* procedure provided images where there is blinking, occluded eyes and blurry frames. These images are helpful in performance evaluation of the pupil detection pipeline.

4 EVALUATION

The ellipse detection pipeline is evaluated with the LPW dataset [38], it provides near eye infrared images of 22 different participants. The image repository of each participant contains near eye images (2000 frames) in different lighting conditions, which are called parts in the dataset. We use the term "use case" to represent the

Figure 3: Study participants perform the experiments, while these run completely on the embedded system itself. The Raspberry Pi (left) acquires the images, while in the fixed marker location experiment each participant would start from the top left marker location, and would go to top right marker, followed by central marker, then bottom left marker, and finally bottom right marker. Displayed here are all markers and the real-time eye camera feed, which is not displayed to the participant while performing the experiment.
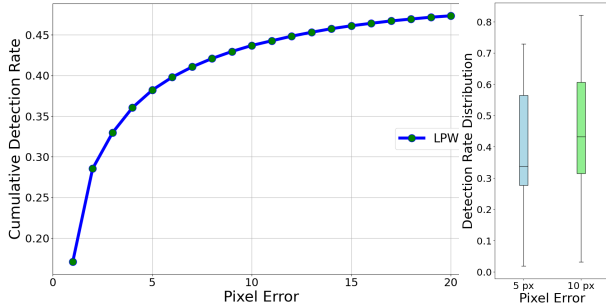

Figure 4: Cumulative error distribution for our ellipse detection method on the LPW dataset [38] up-to 20px error. The detection rate distribution at 5px and 10px is shown in the right.

parts in a participant repository, for example, use case *LPW/P1/P9* represents Participant 1 Part 9 of the dataset. Since the LPW dataset has a lot of challenging images e.g. outdoors, participants with prescription glasses, reflections and shadows, it is therefore selected as a benchmark for testing our pupil detection pipeline.

4.1 Model Evaluation

The Euclidean distance or L2 norm is used as a method for testing the accuracy of the model. The error between the evaluated centre of pupil from our detector and the actual centre of pupil is calculated for every participant in the dataset. The initial frame from each participant (480x640 px) is cropped for region of interest (ROI) and the default processing is used. Changes are only made to the T_{canny}, K_{blur} and the size of pupil to perform detection.

Our ellipse detection pipeline performed at a cumulative accuracy of 38.21% with less than 5px error and 43.6% with less than

10px error and an overall average accuracy of 51.9% at 5.3368 px. The detection rate is comparable to *Pupil Labs*, *ExCuSe* and *Świrski* of 30% detection rate with lower than 5px error on LPW dataset [38]. Our detection method stagnates after 20px error at close to 50% accuracy on the LPW dataset. However, the current ellipse detector is able to perform near accurate detection on complicated dataset without using any extensive feature extraction method e.g. HAAR or ML based operation. The LPW dataset contains challenging images that have high reflections, strong shadows and eyelid occlusions [38]. Images in outdoor scenarios can be challenging, especially in case the participant is wearing prescriptive glasses, resulting in strong reflections. This is one of the reason for the lower detection distribution of less than 10% in these extreme cases (fig 4). One example of this case is with *LPW/P5/P6*, where the pupil is obstructed with eye wear, another example is *LPW/P20/P7* where the pupil is not in focus resulting in weak edges for pupil. For some use cases, as suggested in *PuRe* [34], detection accuracy can be improved by readjustment of the position of eye camera.

4.2 Parameter Estimation

As discussed before, the embedded edge detection pipeline, works on pre-determined set of parameters for Canny threshold (T_{canny}) and Median blur kernel size (K_{blur}). Since there are multiple combinations of T_{canny} & K_{blur} that could be used for pupil detection, it is important to find the best parameter set that could provide detection for most cases. To achieve this goal, a parameter estimation routine is created that uses a variety of near eye images to produce the best estimate on these parameters (T_{canny} & K_{blur}).

4.2.1 Procedure for parameter estimation. All near eye infrared images of each participant are stored in a separate directory. Only images that have a visible pupil are stored in the working directory. Edge cases that have less than 50% of the visible pupil, closed eye and/or high reflections are omitted, since they could affect the parameter estimation result, and there might be no pupil detection. The objective of the procedure is to have images where the pupil detector is bound to work, and if there is no definitive pupil in the image, there would not be any detection, which could add a penalty to the loss parameter. Next, using a list of threshold and kernel size values, the routine takes a combination of T_i & K_i values and uses it to perform pupil detection on all the images in the working directory. The program then calculates a loss for each image or frame in the directory, this gives an estimate on how good the pupil detection was for this combination. The loss parameter L is evaluated as :-

$$L = N_c + A_{min} + C \qquad (2)$$

The loss formula is a confidence measure for the ellipse detection strategy, and gives an estimation on the goodness of the selected parameters. The components of loss criteria include:

N_c : Number of initial contours, is high in case there are more edges in the input image. This happens due to a lower T_{canny} value (see fig 5), due to which the calculated loss (L) is high. Therefore, it is computationally more expensive for our embedded system, and thus it is penalized. For a higher median blur kernel size (K_{blur}), N_c is lower in comparison to a low K_{blur} value.

C : Constant for no contour determination. If no contours/edges are identified in the initial image i.e. $N_c = 0$, there is no possibility to look for a contour that could correspond to pupil. This occurs due to a high T_{canny} value, therefore, the calculated loss value (L) is a constant (C) penality of 1000. This penality is empirically determined for the resolution 480x640 px.

A_{min} : Area difference between convex hull and fitted ellipse, is a measure of goodness in the fitted ellipse and convex hull. If this measure is high, there is a significant area difference between them, therefore the choice of T_{canny} & K_{blur} is not effective. If the area enclosed by the convex hull and the area of the fitted ellipse are close to each other, (A_{min}), becomes minimum. In case of evaluation of A_{min}, there are contours present in the initial image, i.e. C is evaluated as zero.

As can be seen in figure 5, the value $T_i = 14$ and $K_i = 15$ are the best parameters for estimating pupil in *subject 04*, image resolution (480x640 px). Using this method, we can visualize the selection of parameters that could perform well in the pupil detection pipeline. It is recommended to find the best parameters (T_{canny} & K_{blur}) for each subject before performing pupil detection.
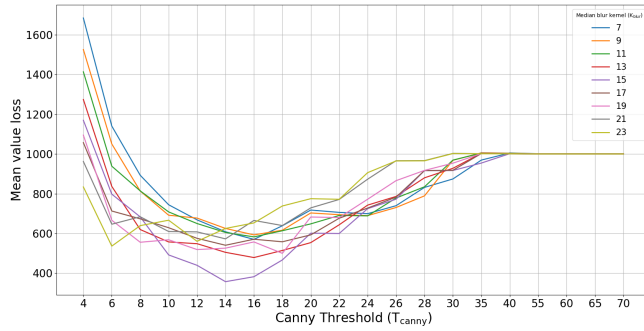

Figure 5: Confidence on parameter selection (T_{canny} & K_{blur}) for Subject 04 from our dataset based on the loss formula (eq 2). The parameters $T_{canny} = 14$ & $K_{blur} = 15$ display an overall better detection rate.

4.3 Runtime & Memory Consumption

Embedded devices are limited in terms of memory and compute resources. It is therefore important to make sure that our programs are not stalling the device by consuming a lot of resources. Our embedded compute device i.e. Raspberry Pi 4B @ 1.5 GHz, 4GB RAM, has enough compute resources to perform our low weight detection method. But performing any graphic intensive processes, could lead to stalling or even crashing of some programs due to low memory. The pipeline for detection method is made using Python and the experimentation was performed on a GUI based operating system. The pupil detection pipeline (*python3* kernel) uses around 100MB of memory and 30 - 35% of the CPU resources to perform detection on 480x640 px initial image, and 90MB memory with 28 - 30% CPU resources for 240x320 px. Both of the detections were performed on 30 *fps*. This can be further improved as the pupil camera supports 90 *fps*.

The time required for each process in the detection pipeline is measured using software timestamps. Python's standard library was used for calculating process timing using a monotonic CPU process timer. The time measurement calculation is performed for 100 frames, where the participant wore the hardware while the detection method is running. The time estimation is made for both image resolutions, 240x320 and 480x640 px. The parameters used for both the resolutions are mentioned in table 4.

The edge based pupil detection pipeline (fig 2), consists of various components like pre-processing, finding of contours, filtering of contours, and post-processing. Figure 6 shows the breakdown of time required by each component in pupil detection in a box whisker plot on a linear scale. It also shows a comparison of time required to perform pupil detection by different resolutions. The left image is for a higher resolution and the right image is for a lower resolution. As can be seen from figure 6, there is a strong difference in finding pupil for both the resolutions while performing the experiment on the hardware. For the lower resolution, the complete operation took 22 ms (median value) for one frame, while it took 50 ms/frame for the higher resolution. The experiment was carried out without any overclocking or acceleration on the Raspberry Pi embedded hardware. For both the experiments, the blurring operation took a significant portion of the total time required for the complete detection process.

Our results have also shown that for the lower resolution of 240x320 pixels, the blurring operation has only a minor impact. The detection rate per 100 images for both with and without blurring is the same, i.e., 92 out of 100 frames. This is due to the in-camera downsampling of the image, which leads to an inherent addition of blurring for the lower resolution setting, whereas for the higher resolution of 480x640 pixels, the blurring operation contributes a more significant part of the detection process: Without the blurring operation, the pupil detection performance is lower, with 25-30 frames out of 100 frames, while with the blurring process added in the pipeline, the detection percentage increases to approximately 60-70 frames out of 100.

4.4 Dataset

To study the effect of images taken with our proposed embedded platform, we have collected **36153** near eye images with our hardware from **20** participants. We have collected images from people of different ethnicity and age groups and made it as generic as possible to test different pupil detection pipelines. Table 5 contains some key features about the collected data. The procedure of data collection contains two different strategies that allow us to have a diverse set of images, which have been motivated in section 3. The dataset from *fixed marker location* can be used as a calibration dataset, containing images of different participants looking at the screen from a fixed distance, or can be used for parameter estimation. The *free movement* images provided enough variations to our dataset and gave us edge cases with blurry images, reflections and close/occluded eye.

Figure 7 displays several example images from the fixed marker experiment of the participants, where the participants' gaze is towards the set markers. Due to this setup, there are almost no images where the participant is either blinking or rolling their eye, thereby providing images that can be used as a calibration method for the

Table 3: Average resource consumption during real time ellipse detection at 30fps. The memory consumed, and CPU usage varied depending on the type of operation being performed e.g. writing pupil centre locations to a file or displaying a GUI image for pupil detection. These calculations were made with no GUI display and file operations.

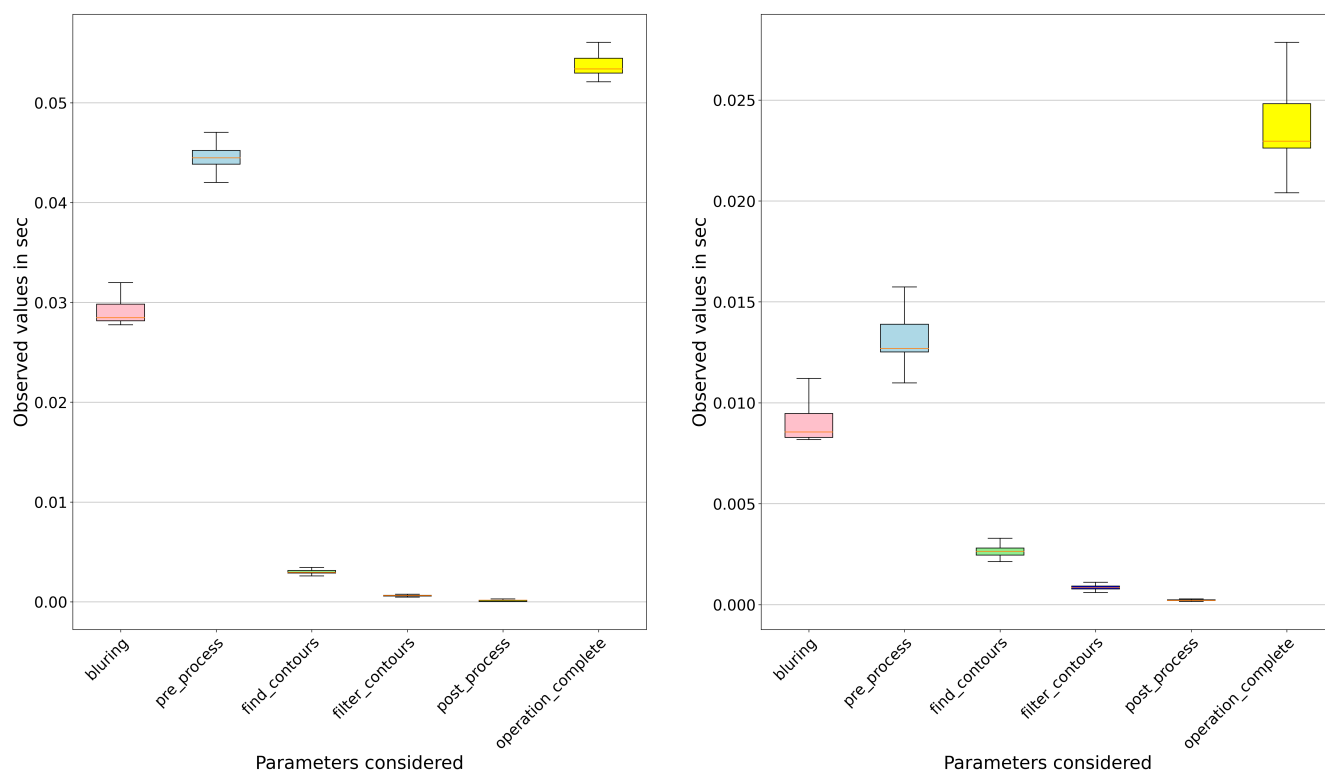| Experiment	Size of initial image (px)	Memory Consumed (*python3* kernel)	CPU
Ellipse detection for 1000 frames	240x320	80-90 MB	28 - 30%
Ellipse detection for 1000 frames	480x640	100 MB	30 - 35%

Figure 6: The time required for pupil detection, in linear scale for both resolution settings of 480x640 px (left) and 240x320 px (right). The time measurement is performed on the Raspberry Pi embedded device. Blurring (pre-process) constitutes a significant amount of time, compared to the other steps in the pipeline, accounting for around 50% in the high and 35% in the low resolution settings.

Table 4: Parameter selection for computing the time requirement for pupil detection. The table below contains the selection of parameters for both resolution settings of our dataset, 480x640 px. and 240x320 px.

Parameter	480x640 px	240x320 px
Canny Threshold (T_{canny})	24	30
Median Blur Kernel Size (K_{blur})	23	7
Max size pupil	2000 px	300 px
Min size pupil	1000 px	100 px

detector. The marker location dataset contains images in two different resolutions, 480x640 px and 240x320 px. Additionally, these

Table 5: Key features to illustrate the diversity of our collected dataset of near eye infrared images, taken with the embedded platform as described in Section 3.

Parameter	Value	Parameter	Value
Participants	20	Images	36153
Participant - Male	12	Nationalities	9
Participant - Female	8	Average Age	26

images were also used to identify the parameters for pupil detection, since the pupil is always clearly visible in the frames. On the other hand, figure 8 contains some example images from the free movement experiment. Figures 8a - 8d show images of participants

Figure 7: Dataset images from the fixed marker location strategy with resolution 480x640 px. The participants are fixating on different marker locations, due to which there are no blinking/closed eye images.
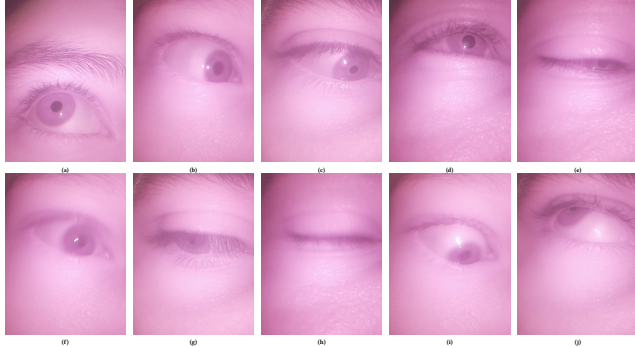

Figure 8: Free movement experiment @ 480x640 px, it contains edge cases as blurry images (fig 8e, 8i), occluded eyes (fig 8e, 8j), closed eyes and gaze fixation in various directions (fig 8a, 8c, 8g).

looking in random directions. There are multiple more difficult edge case images within the dataset which would make the detection process challenging. Figures 8e - 8j show particular examples of some these edge cases. These include blurry images in figure 8f, occluded images in figures 8e and 8h) as well as reflections in figure 8i. The dataset contains free movement images in both resolutions of 640x480 px and 320x240 px.

5 CONCLUSIONS

In this paper, we first presented a real-time and open-source eye tracker setup that allows reproducible pupil detection studies. The hardware design centres around a custom 3D head mount design, which supports a miniature infrared-sensitive eye (Pi Zero) camera, infrared illumination, and a (Logitech C615) world camera. These components are controlled by a single (Raspberry Pi Model 4B) embedded system. Using this platform, we contribute with an open-source data pipeline for pupil detection for the eye camera. Based on ellipse fitting and edge analysis, we present a method that can perform real-time pupil detection on the embedded system. Furthermore, we have contributed a complementary dataset of near eye images of more than 35000 frames, from 20 participants at $30 fps$ for our particular hardware system. In subsequent experiments, we show that the pupil detection pipeline has a cumulative accuracy of 38.21% with less than 5px error, 43.6% with less than 10px error, and an overall average accuracy of 51.9% at 5.3368 px on the Labelled Pupils in the Wild (LPW) dataset [38].

The hardware design files, the detection pipeline source code, and the complementary dataset presented in this paper have been made publicly available to support furthering this research by others and replication of our results. These can be downloaded at: https://github.com/ankurrajw/Pi-Pupil-Detection.

REFERENCES

[1] D. Beymer and M. Flickner. 2003. Eye gaze tracking using an active stereo head. *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.* 2 (2003), II–451. https://doi.org/10.1109/cvpr.2003.1211502

[2] Ali Borji and Laurent Itti. 2013. State-of-the-Art in Visual Attention Modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (2013), 185–207.

[3] Frank H. Borsato, Federal University of Technology, Carlos H. Morimoto, and University of São Paulo. 2019. Towards a low cost and High Speed Mobile Eye Tracker: Proceedings of the 11th ACM symposium on Eye Tracking Research; Applications. https://dl.acm.org/doi/pdf/10.1145/3314111.3319841

[4] Magdalena Borys and Małgorzata Plechawska-Wójcik. 2017. Eye-tracking metrics in perception and visual attention research. In *European Journal of Medical Technologies (EJMT)*. ISASDMT, Lublin, Szczerbowskiego, Poland, 11–23.

[5] Onur Ferhat, Fernando Vilarino, and Francisco Javier Sanchez. 2014. A cheap portable eye-tracker solution for common setups. *Journal of Eye Movement Research* 7, 3 (2014). https://doi.org/10.16910/jemr.7.3.2

[6] Wolfgang Fuhl, Thomas Kübler, Katrin Sippel, Wolfgang Rosenstiel, and Enkelejda Kasneci. 2015. ExCuSe: Robust Pupil Detection in Real-World Scenarios. In *CAIP 2015: Computer Analysis of Images and Patterns*, Vol. 9256. Springer International Publishing, Cham, 39–51. https://doi.org/10.1007/978-3-319-23192-1_4

[7] Wolfgang Fuhl, Thiago Santini, Gjergji Kasneci, and Enkelejda Kasneci. 2016. PupilNet: Convolutional Neural Networks for Robust Pupil Detection. https://doi.org/10.48550/ARXIV.1601.04902

[8] Wolfgang Fuhl, Marc Tonsen, Andreas Bulling, and Enkelejda Kasneci. 2016. Pupil detection for head-mounted eye tracking in The wild: An evaluation of the state of the art. *Machine Vision and Applications* 27, 8 (2016), 1275–1288. https://doi.org/10.1007/s00138-016-0776-4

[9] Stephan J. Garbin, Yiru Shen, Immo Schuetz, Robert Cavin, Gregory Hughes, and Sachin S. Talathi. 2019. OpenEDS: Open Eye Dataset. https://doi.org/10.48550/ARXIV.1905.03702

[10] Kerstin Gidlöf, Annika Wallin, Richard Dewhurst, and Kenneth Holmqvist. 2013. Using eye tracking to trace a cognitive process: Gaze behaviour during decision making in a natural environment. *Journal of Eye Movement Research* 6, 1 (2013), 1–14. https://doi.org/10.16910/jemr.6.1.3

[11] Craig Hennessey, Borna Noureddin, and Peter Lawrence. 2006. A Single Camera Eye-Gaze Tracking System with Free Head Motion. In *Proceedings of the 2006 Symposium on Eye Tracking Research &; Applications* (San Diego, California) *(ETRA '06)*. Association for Computing Machinery, New York, NY, USA, 87–94. https://doi.org/10.1145/1117309.1117349

[12] Benedikt Hosp, Shahram Eivazi, Maximilian Maurer, Wolfgang Fuhl, David Geisler, and Enkelejda Kasneci. 2020. RemoteEye: An open-source high-speed remote eye tracker. *Behavior Research Methods* 52, 3 (2020), 1387–1401. https://doi.org/10.3758/s13428-019-01305-2

[13] Xinyun Hu and Gabriel Lodewijks. 2021. Exploration of the effects of task-related fatigue on eye-motion features and its value in improving driver fatigue-related technology. *Transportation Research Part F: Traffic Psychology and Behaviour* 80 (2021), 150–171. https://doi.org/10.1016/j.trf.2021.03.014

[14] Tariq Jamil, Iftaquaruddin Mohammed, and Medhat H. Awadalla. 2016. Design and implementation of an eye blinking detector system for automobile accident prevention. In *SoutheastCon 2016*. IEEE, NY, USA, 1–3. https://doi.org/10.1109/SECON.2016.7506734

[15] Amir-Homayoun Javadi, Zahra Hakimi, Morteza Barati, Vincent Walsh, and Lili Tcheang. 2015. SET: a pupil detection method using sinusoidal approximation. *Frontiers in Neuroengineering* 8 (2015), 1–10. https://doi.org/10.3389/fneng.2015.00004

[16] Aniwat Juhong, T. Treebupachatsakul, and C. Pintavirooj. 2018. Smart eye-tracking system. In *2018 International Workshop on Advanced Image Technology (IWAIT)*. IEEE, NY, USA, 1–4. https://doi.org/10.1109/IWAIT.2018.8369701

[17] Bronisław Kapitaniak, Marta Walczak, Marcin Kosobudzki, Zbigniew Jóźwiak, and Alicja Bortkiewicz. 2015. Application of eye-tracking in drivers testing: A review of research. *International Journal of Occupational Medicine and Environmental Health* 28, 6 (2015), 941–954. https://doi.org/10.13075/ijomeh.1896.00317

[18] Moritz Kassner, William Patera, and Andreas Bulling. 2014. Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-based Interaction. https://doi.org/10.48550/ARXIV.1405.0006

[19] Joohwan Kim, Michael Stengel, Alexander Majercik, Shalini De Mello, David Dunn, Samuli Laine, Morgan McGuire, and David Luebke. 2019. NVGaze: An Anatomically-Informed Dataset for Low-Latency, Near-Eye Gaze Estimation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300780

[20] Rakshit S. Kothari, Aayush K. Chaudhary, Reynold J. Bailey, Jeff B. Pelz, and Gabriel J. Diaz. 2021. EllSeg: An Ellipse Segmentation Framework for Robust Gaze Tracking. *IEEE Transactions on Visualization and Computer Graphics* 27, 5 (2021), 2757–2767. https://doi.org/10.1109/TVCG.2021.3067765

[21] Kyle Krafka, Aditya Khosla, Petr Kellnhofer, Harini Kannan, Suchendra Bhandarkar, Wojciech Matusik, and Antonio Torralba. 2016. Eye Tracking for Everyone. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, New York City, USA, 2176–2184.

[22] Dongheng Li, Jason Babcock, and Derrick J. Parkhurst. 2006. OpenEyes: A Low-Cost Head-Mounted Eye-Tracking Solution. In *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications* (San Diego, California) *(ETRA '06)*. Association for Computing Machinery, New York, NY, USA, 95–100. https://doi.org/10.1145/1117309.1117350

[23] Jia Zheng Lim, James Mountstephens, and Jason Teo. 2020. Emotion recognition using eye-tracking: Taxonomy, review and current challenges. *Sensors* 20, 8 (2020), 2384. https://doi.org/10.3390/s20082384

[24] Kristian Lukander, Sharman Jagadeesan, Huageng Chi, and Kiti Müller. 2013. OMG! A New Robust, Wearable and Affordable Open Source Mobile Gaze Tracker. In *Proceedings of the 15th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Munich, Germany) *(MobileHCI '13)*. Association for Computing Machinery, New York, NY, USA, 408–411. https://doi.org/10.1145/2493190.2493214

[25] Radosław Mantiuk, Michał Kowalik, Adam Nowosielski, and Bartosz Bazyluk. 2012. Do-It-Yourself Eye Tracker: Low-Cost Pupil-Based Eye Tracker for Computer Graphics Applications. In *Proceedings of the 18th International Conference on Advances in Multimedia Modeling* (Klagenfurt, Austria) *(MMM'12)*. Springer-Verlag, Berlin, Heidelberg, 115–125. https://doi.org/10.1007/978-3-642-27355-1_13

[26] André Meyer, Martin Böhme, Thomas Martinetz, and Erhardt Barth. 2006. A Single-Camera Remote Eye Tracker. In *Proceedings of the 2006 International Tutorial and Research Conference on Perception and Interactive Technologies* (Kloster Irsee, Germany) *(PIT'06)*. Springer-Verlag, Berlin, Heidelberg, 208–211. https://doi.org/10.1007/11768029_25

[27] Jason S. Babcock Rochester Institute of Technology, Jason S. Babcock, Rochester Institute of Technology, Rochester Institute of TechnologyView Profile, Jeff B. Pelz, Clemson University, and Queens University. 2004. Building a lightweight eyetracking headgear: Proceedings of the 2004 symposium on eye tracking research; applications. https://dl.acm.org/doi/pdf/10.1145/968363.968386

[28] Takehiko Ohno and Naoki Mukawa. 2004. A Free-Head, Simple Calibration, Gaze Tracking System That Enables Gaze-Based Interaction. In *Proceedings of the 2004 Symposium on Eye Tracking Research &; Applications* (San Antonio, Texas) *(ETRA '04)*. Association for Computing Machinery, New York, NY, USA, 115–122. https://doi.org/10.1145/968363.968387

[29] OpenCV. 2015. Open Source Computer Vision Library. https://opencv.org/

[30] Keyurkumar Patel, Hu Han, and Anil K. Jain. 2016. Secure face unlock: Spoof detection on smartphones. *IEEE Transactions on Information Forensics and Security* 11, 10 (2016), 2268–2283. https://doi.org/10.1109/tifs.2016.2578288

[31] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016. Towards Foveated Rendering for Gaze-Tracked Virtual Reality. *ACM Trans. Graph.* 35, 6, Article 179 (2016), 12 pages. https://doi.org/10.1145/2980179.2980246

[32] Mathieu Rodrigue, Jungah Son, Barry Giesbrecht, Matthew Turk, and Tobias Höllerer. 2015. Spatio-Temporal Detection of Divided Attention in Reading Applications Using EEG and Eye Tracking. In *Proceedings of the 20th International Conference on Intelligent User Interfaces* (Atlanta, Georgia, USA) *(IUI '15)*. Association for Computing Machinery, New York, NY, USA, 121–125. https://doi.org/10.1145/2678025.2701382

[33] Wayne J. Ryan, Andrew T. Duchowski, and Stan T. Birchfield. 2008. Limbus/Pupil Switching for Wearable Eye Tracking under Variable Lighting Conditions. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications* (Savannah, Georgia) *(ETRA '08)*. Association for Computing Machinery, New York, NY, USA, 61–64. https://doi.org/10.1145/1344471.1344487

[34] Thiago Santini, Wolfgang Fuhl, and Enkelejda Kasneci. 2018. Pure: Robust pupil detection for real-time pervasive eye tracking. *Computer Vision and Image Understanding* 170 (2018), 40–50. https://doi.org/10.1016/j.cviu.2018.02.002

[35] 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 (ETRA '12)*. Association for Computing Machinery, New York, NY, USA, 173–176. https://doi.org/10.1145/2168556.2168585

[36] Tobii. 2022. Most Advanced Eye Tracking System - TOBII Pro Spectrum. Retrieved October 10, 2022 from https://www.tobii.com/products/eye-trackers/screen-based/tobii-pro-spectrum

[37] Tobii Pro AB. 2014. Tobii Pro Lab. Computer software. http://www.tobiipro.com/

[38] Marc Tonsen, Xucong Zhang, Yusuke Sugano, and Andreas Bulling. 2016. Labelled Pupils in the Wild: A Dataset for Studying Pupil Detection in Unconstrained Environments. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research &; Applications* (Charleston, South Carolina) *(ETRA '16)*. Association for Computing Machinery, New York, NY, USA, 139–142. https://doi.org/10.1145/2857491.2857520

[39] Yan Wang, Guangtao Zhai, Shaoqian Zhou, Sichao Chen, Xiongkuo Min, Zhongpai Gao, and Menghan Hu. 2018. Eye Fatigue Assessment Using Unobtrusive Eye Tracker. *IEEE Access* 6 (2018), 55948–55962. https://doi.org/10.1109/ACCESS.2018.2869624

[40] Erroll Wood, Tadas Baltrusaitis, Xucong Zhang, Yusuke Sugano, Peter Robinson, and Andreas Bulling. 2015. Rendering of Eyes for Eye-Shape Registration and Gaze Estimation. In *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Santiago, Chile, 3756–3764. https://doi.org/10.1109/ICCV.2015.428

[41] Piotr Zieliński. 2013. Opengazer: open-source gaze tracker for ordinary webcams (software). http://www.inference.org.uk/opengazer/

[42] Gal Ziv. 2016. Gaze behavior and visual attention: A review of Eye Tracking Studies in aviation. *The International Journal of Aviation Psychology* 26, 3-4 (2016), 75–104. https://doi.org/10.1080/10508414.2017.1313096