DEVELOPMENT OF A HEAD-MOUNTED EYE TRACKER FOR HUMAN-COMPUTER INTERACTION

ONG LI KANG

UNIVERSITI TUNKU ABDUL RAHMAN

DEVELOPMENT OF A HEAD-MOUNTED EYE TRACKER FOR HUMAN-COMPUTER INTERACTION

ONG LI KANG

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2024

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	[illegible]
Name	:	ONG LI KANG
ID No.	:	19UEB02463
Date	:	20 / 04 / 2024

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"DEVELOPMENT OF A HEAD-MOUNTED EYE TRACKER FOR HUMAN-COMPUTER INTERACTION"** was prepared by **ONG LI KANG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Mechatronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Dr. Tan Lee Fan
Date	:	29th April 2024

Signature	:	[illegible]
Co-Supervisor	:	Ir. Dr. Danny Ng Wee Kiat
Date	:	29th April 2024

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Tan Lee Fan for her invaluable advice, guidance and her enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped me in the collection of data and given me enormous patience when testing on the prototype.

ABSTRACT

This paper outlines the design, creation, and integration of an affordable head-mounted eye tracker system specifically designed for applications involving human-computer interaction (HCI). The technology employs easily accessible webcams and 3D-printed parts to create a lightweight and cost-effective eye-tracking gadget. The gaze detection software framework utilizes the Pupil Capture program, which has been developed by Pupil Labs. This ensures precise and dependable eye tracking.

An important contribution of this research is the adaptation and integration of a cursor control plugin, which improves the usability of the eye tracker for human-computer interaction tasks. By including the cursor control plugin, users are able to engage with graphical user interfaces and manipulate the mouse cursor exclusively by using eye movements. This enables a seamless and instinctive engagement with computers.

In addition, a series of comprehensive data-gathering experiments were carried out to assess the performance and efficacy of the eye tracker system that was built. The experiments included a range of scenarios and tasks that are commonly found in HCI applications. Eye-tracking capabilities in terms of its accuracy, precision, and robustness were evaluated by the application dart board graphical user interface. Subsequently, the gazed-controlled mouse cursor could be used for typing in a virtual keyboard.

Keywords: Eye-tracking, Human-computer interaction, Head-mounted, Gaze detection, Cursor control, Pupil Labs, Virtual Keyboard.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

mW/sr	Radiant Intensity
Hz	Frequency
R	Resistance
VS	Supply voltage
VF	LED forward voltage
IF	LED forward current
θ	Visual angle per pixel
S	Screen size
D	Viewing distance
Re	Resolution
θo	Angular offset
xV2	x coordinate of fixation location
yV2	y coordinate of fixation location
xV1	x coordinate of fixation target
yV1	y coordinate of fixation target
σ	Root mean square of the angular distance
n	Number of samples

ALS	Amyotrophic Lateral Sclerosis
API	Application Programming Interface
CAD	Computer Aided Design
CSV	Comma-separated Values
DIY	Do-it-yourself
EOG	Electro-oculography
fps	Frame per second
HCI	Human-computer Interaction
HD	High-definition
IMU	Inertial Measurement Units
IOG	Infrared Oculograhy
IR	Infrared

LED	Light-emitting Diode
Mpx	Megapixels
PCB	Printed Circuit Board
px	Pixels
RC	Radio-controlled
RMS	Root Mean Square
SMI	Senso Motoric Instruments
VOG	Video Oculography (VOG)
VR	Virtual Reality

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Eye tracking is a technology and approach that involves keeping track of a person's eye activities and positions, often to acquire insights regarding where a person is gazing and for how long. As defined by Klaib et al. (2021), the process of monitoring where one is gazing (point of gaze) or the shifting position of an eye relative to the head is known as eye tracking. In other words, eye tracking involves monitoring and tracing the movements of a person's eyes to precisely determine both the direction of their gaze and the duration of their focus. Eye tracking systems are designed to capture the eye's position, movement, and pupil size at any given moment, enabling the detection of regions that pique the user's interest. The accurate tracking of eye movement, including gaze direction, points of fixation (where the eyes halt to focus), and the length of those fixations, is made possible by this technology using a variety of methods, such as cameras or sensors.

Eye-tracking technology is increasingly used in fields like psychology, market research, usability testing, and human-computer interaction to understand visual attention, user behaviour, and cognitive processes. As cited in the research paper by Klaib et al. (2021), eye-tracking technology is applied to many research fields as shown in Table 1.1.

Table 1.1: Research Fields and Applications of Eye-tracking Technology

Research Fields	Explanation	Citations
Visual Systems	Understanding how individuals interact with visual stimuli	(Hristozova, Ozimek, & Siebert, 2018) (Moreno-Esteva, White, Wood, & Black, 2018) (Ulutas, Özkan, & Michalski, 2020)
Neuroscience & Psychology	Studying how people respond to various stimuli and the aspects of human behaviour and mental processes.	(Maria, et al., 2019)
Psycholinguistics & Healthcare	Development of assistive devices for people with communication disabilities	(Park, Subramaniyam, Hong, Kim, & Yu, 2017) (Chen, Fu, Lo, & Chi, 2018)
User Engagement & Interaction	Offering valuable insights for interface design and usability testing.	(Lukander, 2016)
Consumer Research & Marketing	Analyse customer behaviour, paying special attention to what grabs people's attention and has an impact on purchases.	(Hang, Yi, & Xianglan, 2018)

Table 1.1: Research fields and applications of eye-tracking technology

Research Fields	Explanation	Citations
Economy, Clinical Research & Education	Decision-making in the economy, medical studies for illnesses like autism, and educational uses like evaluating pupils' reading abilities	(Colliot & Jamet, 2018)
Sports Performance & Research, Product Design & Software Engineering	Enhancing software user interfaces, product usability, and sports performance	(Obaidellah, Al Haek, & Cheng, 2018) (Zohreh Sharafi, 2015);
Transportation	Improve driver safety, examine driving-related eye movements, and optimize vehicle design.	(Noland, Weiner, Gao, Cook, & Nelessen, 2017);
Virtual Reality (VR)	Enabling more realistic and immersive simulations	(Clay, König, & König, 2019)

This study underscores the utilization of eye-tracking technology in the context of human-computer interaction (HCI), with a particular emphasis on its applications. One prevalent application involves using eye movements and fixations to communicate with and control HCI systems. A noteworthy illustration of this is the control of a computer's mouse cursor, whereby eye movements and fixations are harnessed to execute functions such as clicking, moving, and typing. There are multiple reasons for using eye trackers as an input device to control HCI. First and foremost, it demands a reduced level of exertion on the part of the user. As noted by Glenstrup and Engell-Nielsen (1995), the process of eye-gaze is regarded as an integral aspect of human physiology. Consequently, the eye-gaze system necessitates no unnatural motions or prior training (Alhamad et al., 2022). Secondly, eye-tracking devices serve as swift input mechanisms. When employed as the method of activation, which measures the time needed to select an option, they lead to a reduction in selection time. This theory was examined and validated by Sibert and Jacob (2000), as cited by Alhamad et al., 2022 through their experimental research, which revealed that eye-gaze activation demonstrated an 85% reduction in selection time compared to traditional mouse-click activation.

1.2 Importance of the Study

The development of a head-mounted eye-tracking system for HCI communication is the main topic of this study. It establishes the foundation for prospective future research projects targeted at fusing this technology with a self-driving wheelchair that could be operated by eye motions. Those with severe motor limitations are the main gainers from such developments. According to Collaguazo, Córdova and Gordón (2018), common motor disabilities include ailments like Multiple Sclerosis, which affects the brain and spinal cord and causes problems with limb movement, sensation, and balance, Cerebral Palsy, which typically manifests during infancy or early childhood and causes impaired motor function and Amyotrophic Lateral Sclerosis (ALS), which predominantly affects respiratory and voluntary control over limbs, but has minimal influence on cognitive or sensory capacities. Hence, ALS patients can potentially restore their communication capabilities by utilizing various eye-gaze systems. These systems encompass eye typing, enabling patients to compose text through eye movements, and the application of eye-gaze for wheelchair navigation (Alhamad et al., 2022).

1.3 Problem Statement

Commercial eye tracking systems are known for their durability but come with a significant price tag. As highlighted by Chan (2021), the majority of these eye trackers are priced at RM 10,000 or higher. Furthermore, for seamless communication between the eye tracker and a computer, it is imperative that the eye tracker mechanism demonstrates exceptional accuracy and precision in the detection and extraction of pupil and gaze-related information. Several factors can contribute to inaccuracies and data loss in this process, including the potential for the eye tracker frame to slip, head movements, and limitations in the resolution, frames per second (fps), and field of view of the camera modules (Niehorster et al., 2020).

Consequently, the primary challenge lies in the development and enhancement of cost-effectiveness, resilience, adaptability, as well as the accuracy and precision of current eye tracker designs, thereby facilitating improved execution of HCI applications.

1.4 Aim and Objectives

The aim of the present study is to develop a head-mounted eye tracker for communicating with or controlling an HCI.

The specific objectives are:

1. To design a wearable head-mounted eye tracker which optimised in the consideration of cost efficiency, robustness, and accuracy.
2. To integrate the hardware with a pupil detection software to obtain the pupil and gaze information.
3. To develop an application for human-computer interaction (HCI).

1.5 Scope and Limitation of the Study

The main scope of this project is to investigate, design, and implement a wearable head-mounted eye-tracking system tailored for facilitating HCI. This project requires a thorough analysis of all the flaws in the current prototype. Additionally, it involves a meticulous exploration of the specifications of the camera modules employed in the project. To allow HCI features, the construction of the hardware must be followed by the crucial task of seamlessly connecting the hardware with the software. These capabilities cover activities like controlling cursor movement via pupil and gaze detection. To achieve the best eye-tracking system performance, the project also requires a thorough investigation of alternative algorithms and image processing techniques.

However, it is crucial to acknowledge certain limitations within this study. First off, choosing a camera module imposes a big restriction. The open-source platform Pupil Labs advises using at least a relatively high-end camera module to get optimum performance. Regrettably, other solutions must be taken into account due to financial limitations and the scarcity of suitable camera modules on the market. The gaze detection program used in this study is yet another drawback. Creating gaze detection software from scratch is judged time-consuming and outside the scope of this project due to the complexity and knowledge required. As a result, the study will rely on the use of open-source software that has already been developed for gaze detection.

1.6 Contribution of the Study

The study provides a cost-effective hardware solution for eye tracking by leveraging widely available webcams and 3D-printed parts. This approach significantly reduces the barrier to entry for researchers and developers interested in implementing eye-tracking technology for HCI applications.

The modification and implementation of a cursor control plugin enhance the functionality of the eye tracker for HCI tasks. This plugin allows users to control the mouse cursor solely through eye movements, thereby enabling intuitive and efficient interaction with graphical user interfaces.

The study conducts thorough data collection experiments to assess the performance and effectiveness of the developed eye tracker system. By evaluating the system under various HCI scenarios and tasks, the study provides insights into its accuracy, precision, and robustness in real-world applications.

1.7 Outline of the Report

The report is organized into multiple discrete sections. The text begins with an introduction that provides a broad overview of eye-tracking technology, setting the stage for the following discussion. The literature review follows the introduction and involves citing relevant journal papers on the issue and critically analysing their conclusions.

The methodology section plays a crucial part in the study, explaining the complexities of both hardware and software development. This section provides a detailed explanation of the stages involved in creating the prototype, including the assembly of the hardware components and the complexities of integrating the software.

Following that, the results and discussion section offers a thorough examination of the created prototype by means of rigorous testing. The results obtained from these tests are carefully examined and placed in the wider context of the study's goals and previous research, which helps to develop a more profound comprehension of the effectiveness of the prototype and its possible consequences.

Finally, the study presents its conclusions, summarizing the important insights gained from the investigation. In addition, suggestions for future progress are provided, offering essential assistance for expanding the field of eye-tracking technology and its applications in human-computer interaction.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this part, the general concept of eye-tracking system and eye trackers from different research papers will be discussed. Apart from that, the overall comparison among existing applications of eye-tracking system will be discussed. A comparison of different camera modules used in the eye-tracking system will be highlighted. Furthermore, different algorithms and techniques of eye tracking will be furthered studied and discussed. Lastly, various human-computer interaction (HCI) applications will be compared.

2.2 Eye Tracking System

Eye-tracking system is the mechanism that utilise software to detect and locate the pupil, record and process the pupil and gaze information for other functions and applications. According to Klaib *et al.* (2021), software algorithms including the implementation of libraries or packages such as OpenCV and NumPy are required for pupil recognition, image processing, data classification, and capture of eye movement using a fixation point, fixation time, and saccades.

2.2.1 Taxonomy of Eye Movements

Humans perceive the world through their eyes via a process involving light reflection. As elucidated by the National Eye Institute (2022), when light reflects off an object and enters the eyes, it undergoes a sequence of optical events. Initially, the incoming light ray traverses the dome-shaped cornea and undergoes refraction. Subsequently, it proceeds through the small aperture within the iris, known as the pupil, as illustrated in Figure 2.1. The iris serves the crucial function of regulating the amount of light permitted to pass through the pupil. The light then continues its journey through the lens positioned behind the iris, where it undergoes further refraction and ultimately converges on the retina, forming an upside-down image, as depicted in Figure 2.2 (Daly, n.d.). The retina, comprising numerous photoreceptor cells, plays a vital role by the incoming light into electrical signals, which are subsequently relayed to the brain for intricate processing and interpretation.

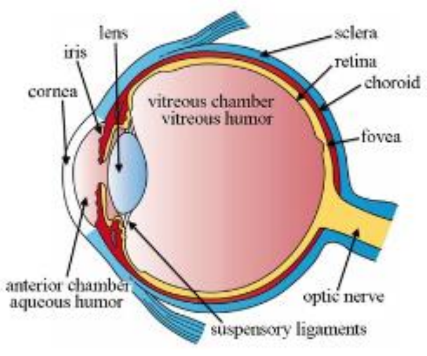


Figure 2.1: Anatomy of Eye (George, 2019)

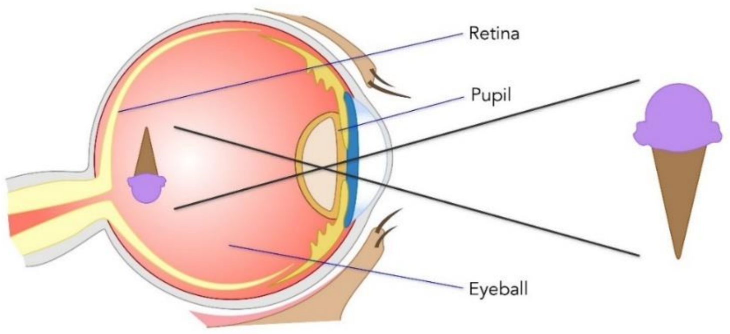


Figure 2.2: Formation of an Image on the Retina (Daly, n.d.)

There are generally four primary types of eye movements, as delineated by Purves *et al.* (2001): saccades, smooth pursuit movements, vergence movements, and vestibulo-ocular movements. Saccades are rapid, ballistic eye movements that quickly shift the gaze from one fixation point to another, bringing a portion of the visual scene onto the fovea. Examples of saccades range from subtle eye movements during reading to scanning a large area. Even when a person appears to be staring at a single point, known as fixation, it involves a series of small, accumulated saccades (Farnsworth, 2019).

Smooth pursuit movements, on the other hand, are slower eye movements aimed at maintaining focus on a moving stimulus, ensuring that it remains centered on the fovea (Purves, 2001). However, when the stimulus exceeds a speed of 30°/s, the smooth pursuit is no longer effective, and saccades take over to track the target (Land & Tatler, 2012 as cited in Miseviciute, n.d.).

Vergence movements occur when the eyes shift focus between objects at varying distances in three-dimensional space, tracking objects moving closer or farther away (Farnsworth, 2019). Convergence occurs when the eyes converge to focus on a nearby object, while divergence happens when they diverge to focus on a distant object. Vestibulo-ocular movements are reflexive actions that stabilize vision when the head is in motion.

In addition to these fundamental eye movements, Farnsworth (2019) introduces a newer type known as optokinetic response movements, which combine saccadic and smooth pursuit motions to track a stimulus. An example provided is the eyes performing a corrective horizontal movement to return to an object's previous location as it moves out of the field of vision while looking out from a moving vehicle. Eye fixations refer to the moments when the eyes stably position an object on the fovea, enabling a thorough intake of visual information. The development of eye-tracking technology is rooted in the principles and characteristics of these various eye movements.

2.2.2 Existing Eye Trackers

Two primary categories of eye trackers exist, remote or screen-based eye trackers and head-mounted eye trackers as shown in Figure 2.3. Screen-based eye trackers, as described by George (2019), monitor an observer's eye movements from a distance and are typically used in controlled desktop environments. Observers are required to sit in front of the device or screen while engaging with stimuli for eye tracking to occur. This type of eye tracker is designed to maintain detection accuracy even when the observer moves their head while gazing at the screen, but it is constrained by the desktop environment. In contrast, head-mounted eye trackers, also detailed by George (2019), incorporate at least two cameras, one for detecting pupil movements and the other for capturing the observer's field of view from a real-world perspective. Head-mounted eye trackers record pupil movements at a closer proximity compared to screen-based counterparts. They excel in collecting data in real-life environments, allowing observers to move freely while wearing them. However, a potential limitation lies in the possibility of slippage or misalignment of the eye tracker when the user is in motion (Farnsworth, 2023).

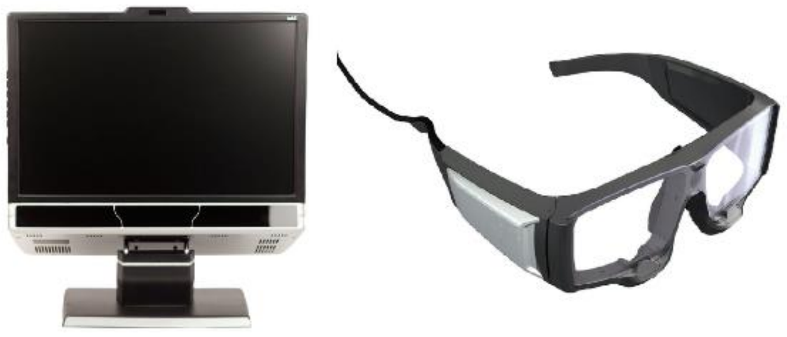


Figure 2.3: Types of Eye Trackers: (a) Screen-based Eye Tracker and (b) Head-mounted Eye Tracker (George, 2019)

Several eye-tracking systems are available in the market, including well-known brands like Pupil Labs, Tobii, Argus Science, and Senso Motoric Instruments (SMI). A study conducted by Niehorster et al. (2020) aimed to compare the performance of four existing eye tracker models: Tobii Pro Glasses 2 in 50 Hz mode, SMI Eye Tracking Glasses 2.0 at 60 Hz, Pupil Labs' Pupil with Pupil Capture software in 3D mode, and Pupil Labs' Pupil with the Grip gaze estimation algorithm implemented in the EyeRecToo software, referred to as Grip. Users were tasked with speaking, displaying facial expressions, making slight random movements with the eye trackers, and readjusting the eye trackers to their original calibrated orientation. The comparison of the eye trackers' performance in terms of accuracy, precision, and data loss, and the results are summarized in Table 2.1.

In short, when compared to SMI and Pupil Labs, Tobii and Grip eye-tracking setups performed better in terms of gaze position deviation during slippage and data loss. However, they exhibit reduced precision in gaze position information. SMI had higher precision but more gaze position variations and data loss during slippage. During slippage, Pupil-Labs had larger deviations and data loss, as well as reduced precision, highlighting possible issues with retaining accuracy during mobility.

Table 2.1: Comparison between Four Eye Trackers (Niehorster *et al.*, 2020)

Criteria	**Tobii and Grip**	**Pupil Labs**	**SMI**
Gaze Position Deviation Due to Slippage	Minimal average median deviations during slippage (≤ 0.4° increase over baseline in facial movement and ≤ 0.8° increase in eye-tracker movement conditions)	Larger deviations (0.8-3.1° in facial movement conditions and an increase of 5.9-25° in eye-tracker movement conditions)	Larger gaze position deviations during slippage (5.9-25° increase over baseline in eye-tracker movement conditions)
Data Loss During Slippage	Low data loss during slippage (≤ 5.4%)	Average 9.1 - 47% data loss and even more during facial- and eye-tracker movement conditions compared to the baseline	Significant amount of data loss during slippage (average 9.1 - 47%)
Validation After Slippage	Gaze deviations remained similar during validation conditions	Increase in deviation between the two validation moments	Increase in deviation, and data loss was higher at the second validation moment
Precision	Lower precision in gaze position signals	Lower precision during validation at the start of the recording but showed an increase in precision over the session	Higher precision during validation at the start of the recording

In another study conducted by Macinnes et al. (2018), the angular accuracy and precision of three widely used versions of wearable eye trackers were evaluated. These models include the Pupil Labs 120 Hz Binocular (Pupil Core), SMI ETG 2.6, and Tobii Pro Glasses 2. This experiment aims to accurately estimate the calibration performance by assessing a target stimulus at three distinct distances (1m, 2m, and 3m) and three different gaze angles at each distance (target grids positioned at -10°, 0°, and +10° horizontal visual angle relative to the centre of the stimulus). Participants were directed to remain seated and motionless throughout the task for each condition. The test comprised of 9 trials in which participants were instructed to focus their gaze on a distinct spot on the same target grid. Participants direct their attention to each area for a duration of 3 seconds. The 9 places were arranged in a random sequence for each condition. The angular offset and polar angle relative to the target point are calculated for each gaze location occurring between 500-2500ms, as depicted in Figure 2.4.

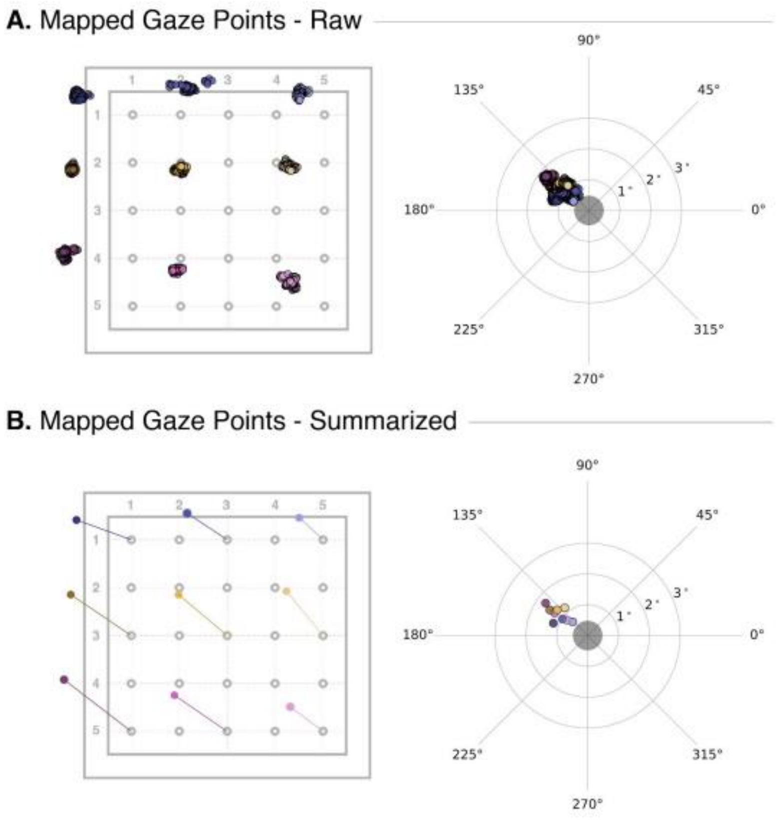


Figure 2.4: Data Collected: (a) Raw Gaze Points and (b) Mapped Gaze Points (MacInnes et al., 2018)

The average total angular accuracy and precision are shown in Table 2.2. Pupil Labs has the offsets of 0.84° and 0.16°, showing the highest angular accuracy and angular precision as the offsets are lowest among all the models. The overall accuracy and precision are graphed as shown in Figure 2.5.

Table 2.2: Overall Accuracy and Precision across Wearable Eye Tracker Models

Eye-Tracker Model	Mean Overall Accuracy	Mean Overall Precision
Pupil Labs	0.84° (0.42)	0.16° (0.07)
SMI	1.21° (0.66)	0.19° (0.08)
Tobii	1.42° (0.58)	0.34° (0.16)

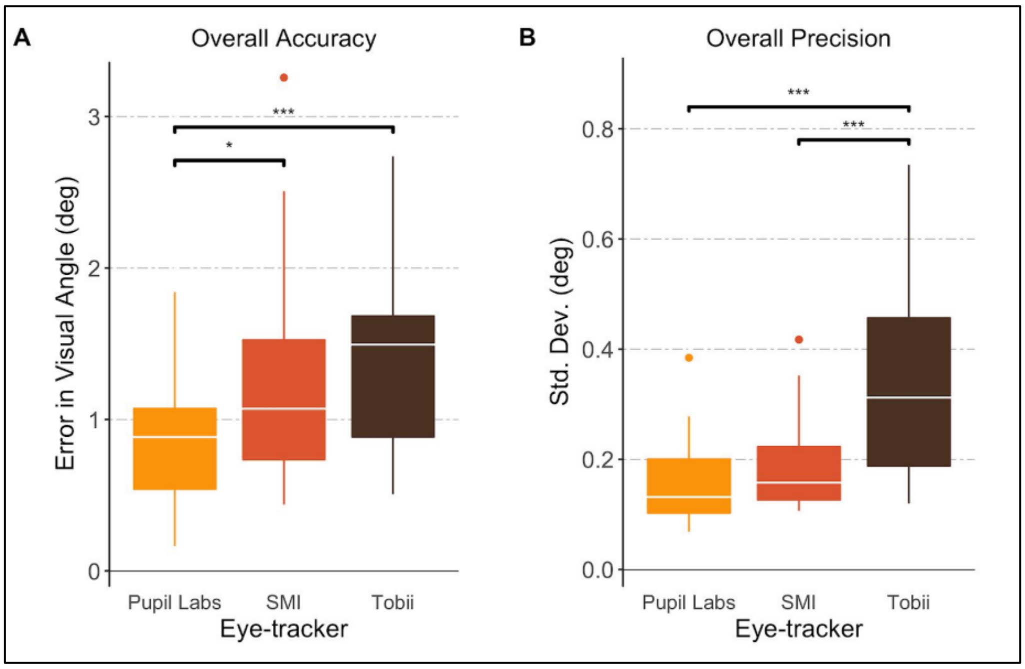


Figure 2.5: Overall Accuracy and Precision: (a) Overall Accuracy and (b) Overall Precision

2.2.3 Eye Tracking Working Principles

There are four major techniques that play essential roles in eye tracking, which are scleral search coil technique, infrared oculography (IOG), electrooculography (EOG), and video oculography (VOG).

The scleral search coil technique was originally devised by Robinson in 1963 (Klaib et al., 2021). It involves the placement of a contact lens equipped with mirrors directly onto the cornea and sclera (Thankachan, 2018). This contact lens is affixed with a coil, and the eye's position is determined by the induced potential difference when it resides within a magnetic field (George, 2019). To monitor horizontal eye movements, a pair of coils is positioned on opposite sides of the head, creating a field. Likewise, for tracking orthogonal eye movements in the vertical direction, another set of coils is placed in a relatively vertical orientation to the first set (Alhamad et al., 2022). Eye movements can be captured by eye-tracking devices by detecting the infrared wavelength range reflected by a mirror and then assuring the exact alignment of a picture with the eye's motion (Klaib et al., 2021). This technique is shown in Figure 2.6.

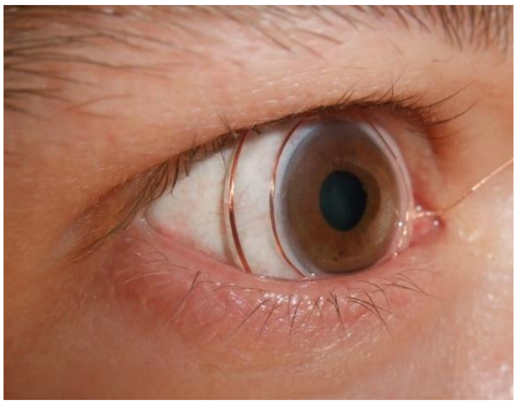


Figure 2.6: Scleral Search Coil Technique (Thankachan, 2018)

The infrared oculograhy (IOG) technique evaluates the intensity of infrared light reflected from the sclera, which provides information about eye location (Klaib et al., 2021). To address the issue of susceptibility to head movements, Pupil Center Corneal Reflection (PCCR) is introduced. According to Farnsworth (n.d.) from iMotions, in PCCR, near-infrared light is aimed at the central area of the eye, specifically the pupil, resulting in visible reflections in both the pupil and the cornea, which are captured by an infrared camera. The technique involves monitoring the position of the pupil centre in relation to the corneal reflection. By measuring the distance between these two points, it becomes possible to determine the gaze direction accurately.

Electro-oculography (EOG) utilises potential differences across electrodes which surrounds the eyes region. This technique records a range of 5 to 200 microvolts with a sensitivity of 20 microvolts per degree of eye movement (Alhamad et al., 2022). However, this method relies on eye movement relative to head position, hence it does not suit the estimation of point of regard (George, 2019). The EOG technique is illustrated in Figure 2.7.

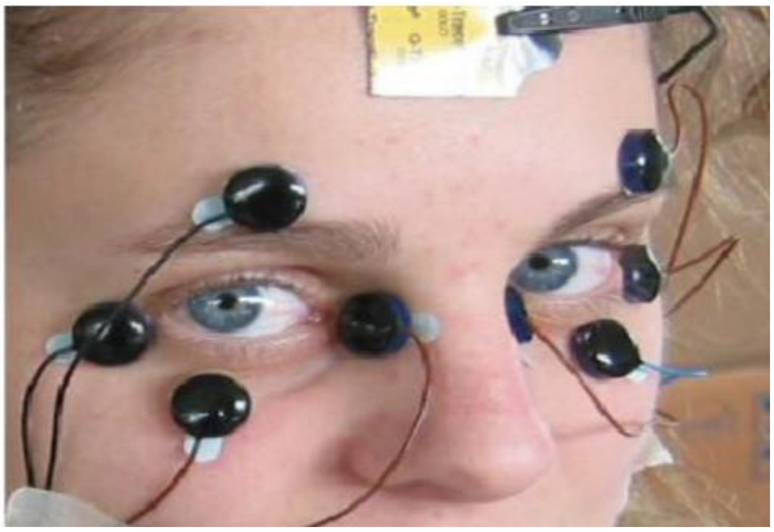


Figure 2.7: EOG Technique (Thankachan, 2018)

The Video Oculography (VOG) device can utilize either visible or infrared lights and functions in a non-invasive manner, enabling eye monitoring from a distance. Video eye tracking can be implemented using either a single camera or numerous cameras. In this technique, to accurately measure the point of regard, either the head must remain stationary so that the eye's location is relative to the head, synchronizing with the point of regard, or multiple ocular characteristics, such as the corneal reflection of an infrared light and the pupil centre, must be measured to distinguish head movement from eye rotation. A head-mounted VOG is shown in Figure 2.8.

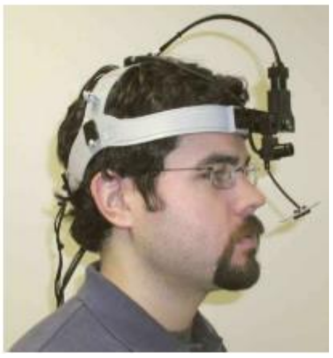
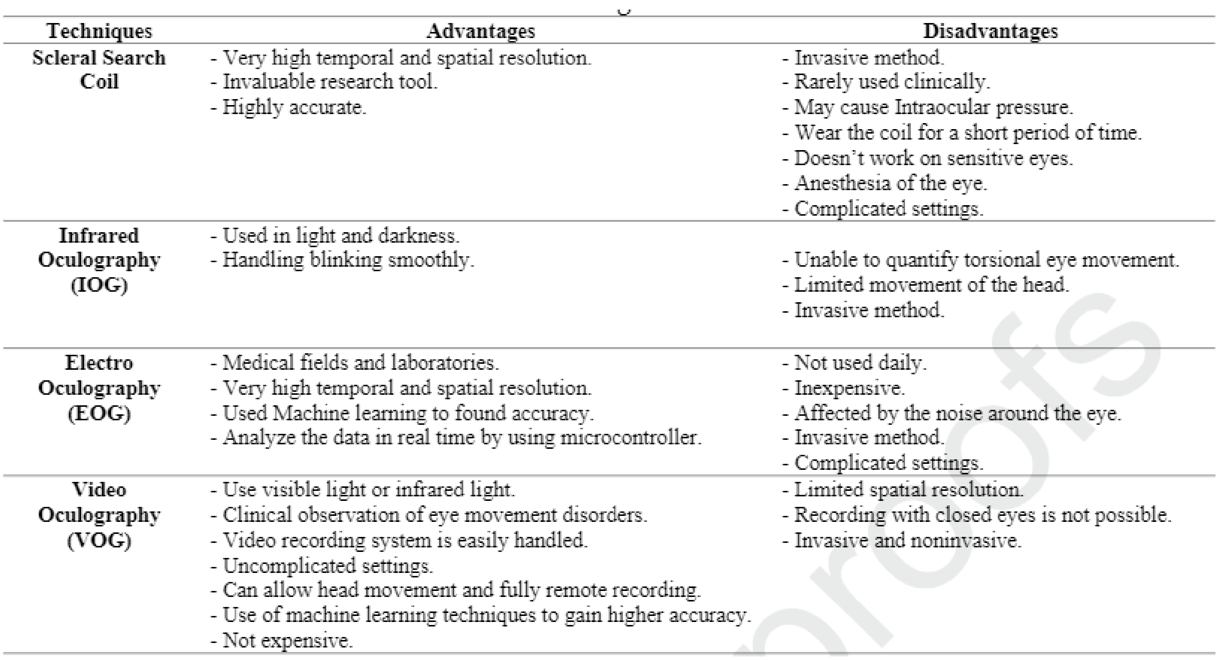


Figure 2.8: Video-based Head-mounted Eye Tracker (Alhamad *et al.*, 2022)

The advantages and disadvantages of these four types of eye tracking techniques are summarized in Figure 2.9 below (Klaib et al., 2021).



Techniques	Advantages	Disadvantages
Scleral Search Coil	- Very high temporal and spatial resolution. - Invaluable research tool. - Highly accurate.	- Invasive method. - Rarely used clinically. - May cause Intraocular pressure. - Wear the coil for a short period of time. - Doesn't work on sensitive eyes. - Anesthesia of the eye. - Complicated settings.
Infrared Oculography (IOG)	- Used in light and darkness. - Handling blinking smoothly.	- Unable to quantify torsional eye movement. - Limited movement of the head. - Invasive method.
Electro Oculography (EOG)	- Medical fields and laboratories. - Very high temporal and spatial resolution. - Used Machine learning to found accuracy. - Analyze the data in real time by using microcontroller.	- Not used daily. - Inexpensive. - Affected by the noise around the eye. - Invasive method. - Complicated settings.
Video Oculography (VOG)	- Use visible light or infrared light. - Clinical observation of eye movement disorders. - Video recording system is easily handled. - Uncomplicated settings. - Can allow head movement and fully remote recording. - Use of machine learning techniques to gain higher accuracy. - Not expensive.	- Limited spatial resolution. - Recording with closed eyes is not possible. - Invasive and noninvasive.

Figure 2.9: Summary of Advantages and Disadvantages of Different Eye Tracking Techniques (Klaib *et al.*, 2021)

Currently, commercially available eye trackers operate on the VOG system as noted on Tobii website. IR light is employed to illuminate the eyes and generates corneal reflections. Subsequently, cameras capture these reflections by means of video capturing. This enables the capability of tracking eye movements in real time.

2.3 Hardware

Eye trackers usually consists of several main components, including frame design, camera modules, filters, LEDs, and computing devices. In this part of review, different models will be discussed.

2.3.1 Frame Design

Numerous frame designs for head-mounted eye trackers are fashioned from a variety of materials, such as glasses, goggles, headbands, and masks, which are adapted to serve as eye-tracking devices. To optimize the performance, durability, and overall user experience, specific design considerations come to the fore.

First and foremost, the frame's design should prioritize flexibility and user comfort. It is essential that the frame is adjustable and ergonomically designed, accommodating variations in users' head shapes and sizes. For instance, adjustments should be made possible for camera mounting and temple components, a concept demonstrated in the work of Hausamann et al. (2020) who enhanced the Pupil Core frame with nose pad cushions for comfort and IR-reflective films to mitigate unwanted IR reflections.

In addition to ergonomic considerations, the frame should be engineered to support camera modules and essential components at its front. Consequently, the design should incorporate counterweight features to ensure the eye tracker's stability during use. Furthermore, the frame's weight is a crucial factor to be addressed. In the context of research aiming to develop assistive technologies, such as the visual typewriter for individuals with disabilities studied by Alhamad et al. (2022), customizations often involve the addition of IR LEDs, webcams, and strategically placed aluminium supports. Similarly, in the project by Mazhar et al. (2015) focused on controlling an RC car via an eye-tracking system, a headband crafted from curved acrylic sheeting and an elastic band was customized for the eye tracker. In both cases, camera modules were dehoused, retaining only the PCB to minimize the eye tracker's overall weight.

Furthermore, open-source eye tracking platforms like Pupil Labs provide guidelines for do-it-yourself (DIY) Pupil Core eye trackers that employ 3D printing technology. While the frame CAD design may not be open source, downloadable CAD designs for camera mounts are readily available. One of the most cost-effective, durable, and practical approaches is the use of a 3D-printed frame constructed from polylactic acid (PLA), primarily due to PLA's exceptional mechanical properties, as highlighted by Szarlej et al. (2021).

2.3.2 Selection of Camera Modules

Camera modules are widely regarded as the most critical components of an eye tracker, with specifications such as resolution, frame rate, refresh rate, and additional features like autofocus directly influencing the eye tracker's performance. Hausamann, Sinnott and MacNeilage (2020) introduced the Intel RealSense T265, equipped with visual-inertial simultaneous localization and mapping (VI-SLAM), as a world camera. VI-SLAM leverages a combination of cameras and Inertial Measurement Units (IMU) for navigation. In another study by Alhamad *et al.* (2022), the Mercury ViewCam Zoom10, designed monocularly, was implemented as the eye camera. Sabab et al. (2022) proposed the use of the Logitech C920 for detecting the user's gaze. According to Pupil Labs' Github page, for the customization of a Pupil Core eye tracker, it is recommended to use Logitech C525 or C615 as the world camera and Microsoft Lifecam HD-6000 as the eye camera. All of these camera modules have been compared with respect to their resolution, frame rate, refresh rate, and other features, and the results are summarized in Table 2.3.

Table 2.3: Comparison of Proposed Model of Camera Used in Eye Tracking

	Intel RealSense T265	**Mercury ViewCam Zoom10**	**Logitech C920**	**Logitech C615**	**Logitech C525**	**Microsoft Lifecam HD-6000**
Max. Video Resolution	848 × 800 px	1280 × 960 px	1920 × 1080 px	1920 × 720 px	1280 × 720 px	1280 × 720 px
Max. Frame Rate	30 fps	NA	30 fps	30 fps	30 fps	30 fps
Megapixel	8 Mpx	10 Mpx	15 Mpx	8 Mpx	8 Mpx	1 Mpx
Built-in Microphone	No	NA	Yes	Yes	Yes	Yes
Auto Focus	No	NA	Yes	Yes	Yes	Yes
IR Illumination	NA	Yes	No	NA	No	NA
Additional Features	VI-SLAM	Night Vision	HD Auto Light Correction	Motion Sensor	Motion Sensor	True-color Technology
Price Range	RM 2,680.00	NA	RM 240 – RM 500	RM 200 – RM 400	RM 200 – RM 300	RM 500 – RM 700
Market Availability	Yes	No	Yes	Yes	Yes	Yes (not in Malaysia)

2.3.3 Infrared (IR) LED

To achieve the desired dark pupil effect for eye tracking, it is recommended to position near-infrared LED lights alongside the eye camera, as proposed by Alhamad *et al.* (2022). Detailed tutorials illustrating the steps to de-solder the visible light LED and replace it with an infrared (IR) LED can be found on Pupil Labs' GitHub page. Among the suggested IR LED models is the SFH 4050-Z infrared (IR) Emitter (850 nm), which provides a radiant intensity of 4 mW/sr. Alternatively, HT-170IRPJ and VSMY1850CT-ND are viable alternatives, offering a radiant intensity of 5 mW/sr.

2.4 Software

A software application is required to be integrated with hardware to perform pupil and gaze detection. In this part, softwares used for eye tracking system will be discussed.

2.4.1 Gaze Detection Software

Various gaze-detection software options are available for eye-tracking applications. In addition to Pupil Labs, an open-source platform that provides the Pupil Capture software for gaze detection, there is another open-source software known as EyeRecToo. EyeRecToo is compatible with Pupil Eye Trackers and Pupil DIY setups and serves as a second-generation software for head-mounted eye trackers. However, it's important to note that the information and resources available for EyeRecToo are not as extensive as those for Pupil Labs, and it has not achieved the same level of popularity. Pupil Labs stands out by offering a wide range of plugins and Application Programming Interfaces (APIs) for incorporating custom features into the application. Furthermore, Pupil Labs provides robust technical support and training opportunities, allowing users to consult with experts and seek assistance when encountering challenges.

2.5 Summary

The literature review encompasses several articles related to the eye-tracking system. It initiates by discussing the overall system theory and fundamental working principles. Subsequently, it explores the hardware and software utilized by other researchers in the field. This comprehensive examination provides a lucid perspective on the project in terms of its concept and design. In short, an effective eye tracker design should incorporate elements such as an ergonomic frame which is not easy to slip, camera modules with appropriate specifications, and compatible software.

CHAPTER 3

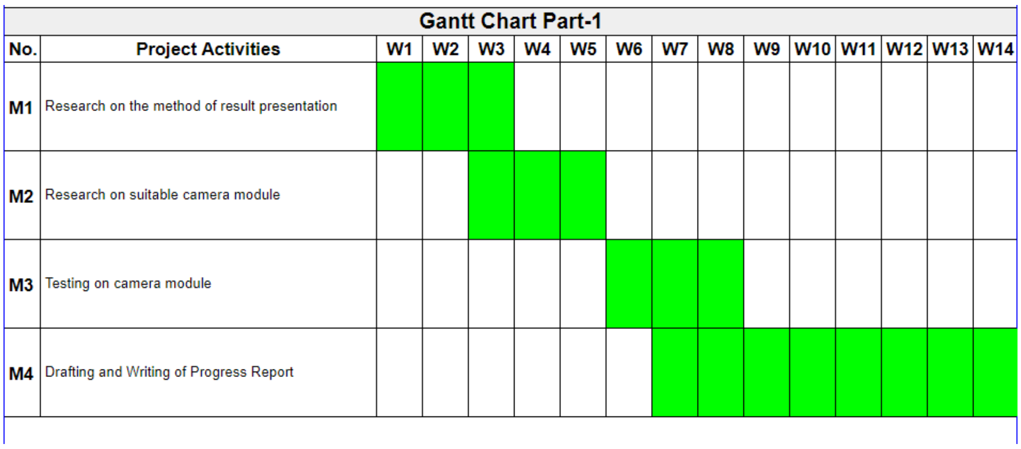
METHODOLOGY AND WORK PLAN

3.1 Introduction

In this chapter, the project planning and milestones, an overview of the head-mounted eye tracker, hardware development and software development will be discussed in details.

3.2 Project Planning and Milestones

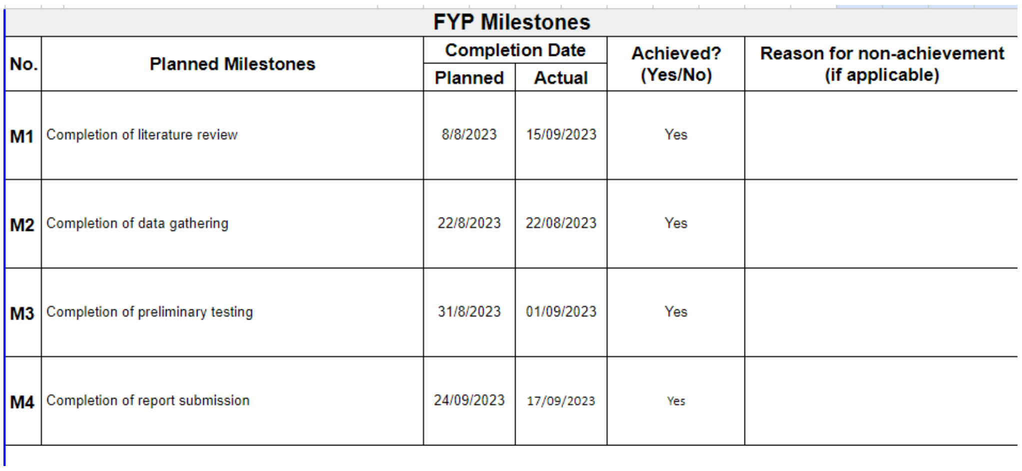
The project is divided into two major phases. The first phase highlights the research on the method of result presentation, research on suitable camera modules, testing on camera modules, and drafting and writing of progress report. Figure 3.1 shows the Gantt Chart of FYP's first phase. Figure 3.2 shows the milestone of FYP's first phase. All milestones are achieved.



Gantt Chart Part-1

No.	Project Activities	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14
M1	Research on the method of result presentation	🟩	🟩	🟩											
M2	Research on suitable camera module			🟩	🟩	🟩									
M3	Testing on camera module						🟩	🟩	🟩						
M4	Drafting and Writing of Progress Report							🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

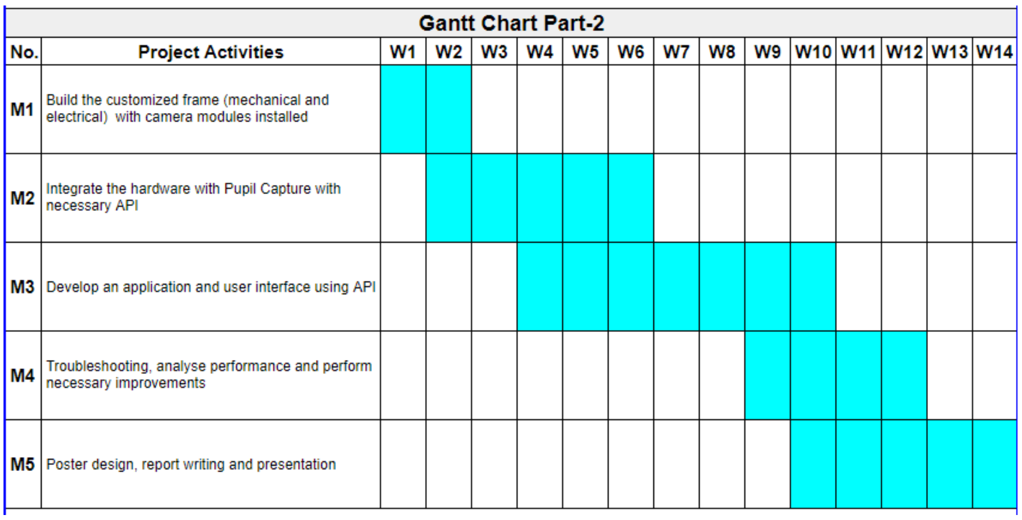
Figure 3.1: Gantt Chart of FYP Phase 1



FYP Milestones					
No.	**Planned Milestones**	**Completion Date**		**Achieved? (Yes/No)**	**Reason for non-achievement (if applicable)**
		Planned	**Actual**		
M1	Completion of literature review	8/8/2023	15/09/2023	Yes	
M2	Completion of data gathering	22/8/2023	22/08/2023	Yes	
M3	Completion of preliminary testing	31/8/2023	01/09/2023	Yes	
M4	Completion of report submission	24/09/2023	17/09/2023	Yes	

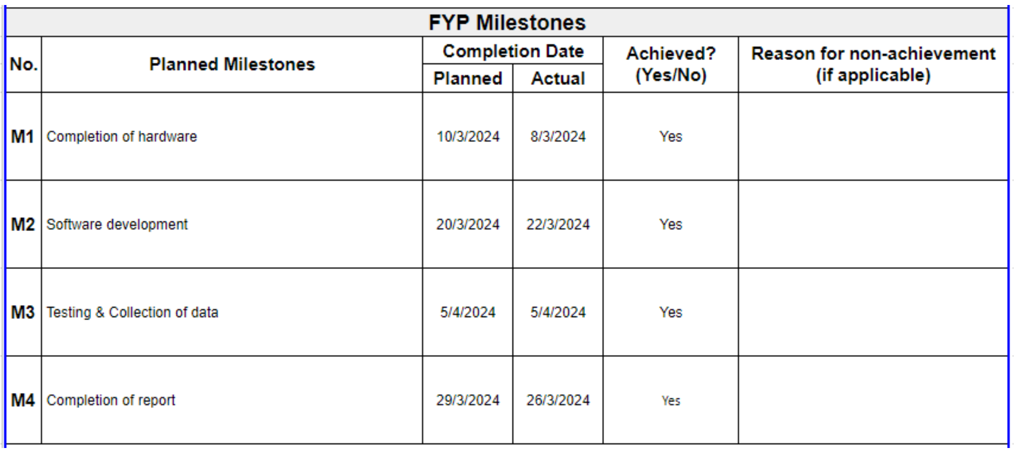
Figure 3.2: Milestones of FYP Phase 1

The second stage of the project is dedicated to the advancement of the head-mounted eye tracker. The development process is divided into two distinct sub-phases: hardware development and software development. The development process is accompanied by ongoing enhancements. Once the development is finished, the testing, and data collection are also carried out. Finally, the process of creating and composing the report has been completed. Figure 3.3 shows the Gantt Chart of FYP's second phase. Figure 3.4 shows the milestones of FYP's second phase. All milestones are successfully met within the planned timeframe.



Gantt Chart Part-2															
No.	**Project Activities**	**W1**	**W2**	**W3**	**W4**	**W5**	**W6**	**W7**	**W8**	**W9**	**W10**	**W11**	**W12**	**W13**	**W14**
M1	Build the customized frame (mechanical and electrical) with camera modules installed	🟦	🟦												
M2	Integrate the hardware with Pupil Capture with necessary API		🟦	🟦	🟦	🟦	🟦								
M3	Develop an application and user interface using API				🟦	🟦	🟦	🟦	🟦	🟦	🟦				
M4	Troubleshooting, analyse performance and perform necessary improvements									🟦	🟦	🟦	🟦		
M5	Poster design, report writing and presentation										🟦	🟦	🟦	🟦	🟦

Figure 3.3: Gantt Chart of FYP Phase 2



FYP Milestones					
No.	Planned Milestones	Completion Date		Achieved? (Yes/No)	Reason for non-achievement (if applicable)
		Planned	Actual		
M1	Completion of hardware	10/3/2024	8/3/2024	Yes	
M2	Software development	20/3/2024	22/3/2024	Yes	
M3	Testing & Collection of data	5/4/2024	5/4/2024	Yes	
M4	Completion of report	29/3/2024	26/3/2024	Yes	

Figure 3.4: Milestones of FYP Phase 2

3.3 Overview of Head-mounted Eye Tracker

The primary goal is to create a head-mounted eye tracker that can be seamlessly included in a HCI application. This project involves integrating a head-mounted eye tracker with the Pupil Capture v3.5.1 program by Pupil Labs to provide a gaze-controlled mouse cursor. This allows the user to move the cursor and make clicks on the computer screen using their eyes. Regarding the hardware, the customized frame will have two camera modules: a world camera and an eye camera. The eyes will be illuminated by unique IR LED and then captured by the eye camera. Once the hardware is installed, the Pupil Capture v3.5.1 software is initiated to calibrate and record real-time pupil and gaze data. The implementation of the cursor control in this project requires the use of an additional cursor plugin API to carry out specified functions. Once the plugin is activated, a user interface (UI) is created to gather data in order to measure the angular accuracy and angular precision. Figure 3.5 depicts the entire system.

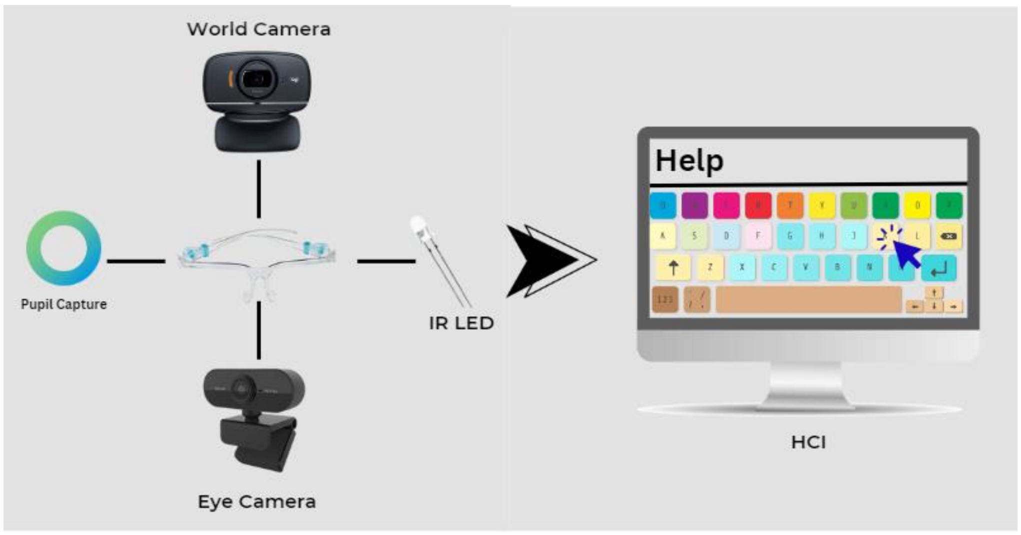


Figure 3.5: Overview of Head-mounted Eye Tracker Integrated with HCI

3.4 Hardware Development

Prior to the commencement of hardware development, the necessary supplies are procured via online platforms such as Shopee and Carousell. Once all the necessary supplies have been collected, the hardware development process commences by removing the outer case of the webcams. The original casings are removed, leaving only the PCBs. Next, the eye camera requires certain alterations, such as removing the IR filter and constructing the circuit for the IR LED onto the eye camera PCB. Finally, the mounting brackets that secure both the world and eye camera are created and manufactured utilizing 3D printing technology. The flowchart of the hardware development is shown in Figure 3.6.

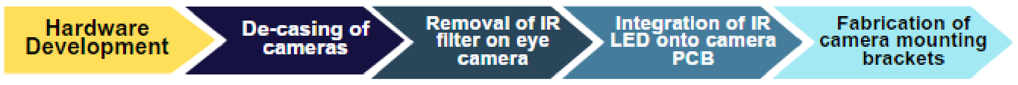

Hardware Development
De-casing of cameras
Removal of IR filter on eye camera
Integration of IR LED onto camera PCB
Fabrication of camera mounting brackets

Figure 3.6: Flowchart of Hardware Development

3.4.1 Frame

The eye tracker frame must meet specified parameters, including: 1) ergonomic adaptation, 2) lightweight durability, and 3) stability to prevent slippage caused by movement. This frame has been adapted to accommodate a monocular design, which means it has a single-eye camera instead of the more common binocular setup. Pupil Labs offers detailed instructions for building DIY Pupil Core eye trackers using 3D printing technology. However, it is important to understand that the design of the frame is proprietary, and using open-source camera mounts can be difficult due to size and printability issues.

Therefore, based on current prototypes and the Pupil Core as standards, necessary adjustments are necessary to meet the specified criteria. The utilization of a commercially available face shield frame arises as a practical and cost-effective alternative, capitalizing on its robust structure and reasonable price. As depicted in Figure 3.7, the selected face shield frame acts as the fundamental framework for the eye tracker. In addition, to improve stability and reduce slipping, a set of nose pads, shown in Figure 3.8, are attached to the nose supports of the face shield frame.

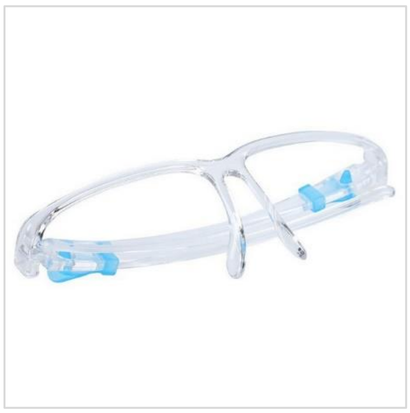


Figure 3.7: Face Shield Frame

Figure 3.8: Adhesive Nose Pads

3.4.2 Camera Modules

The eye tracker consists of two cameras: the eye camera and the world camera. It is advisable to utilize the camera modules suggested by Pupil Labs when taking a do-it-yourself approach. This is due to the necessity of making alterations to the camera modules, such as removing the shell and performing soldering tasks. The recommended global camera module options are the Logitech C525 or the Logitech C615. The Logitech C525 is selected due to its comparable specifications with the C615, with the exception of the resolution. Nevertheless, the 1280 × 720 px resolution fulfills the minimum requirements, and given the cost of the camera module, the Logitech C525, seen in Figure 3.9, is the favored option. The Logitech C525 is acquired through a transaction on the online marketplace, Carousell. Upon arrival, the device is disassembled, and its printed circuit board (PCB) is revealed, as depicted in Figure 3.10. The specifications of Logitech C525 are tabulated in Table 3.1.

Figure 3.9: Logitech C525

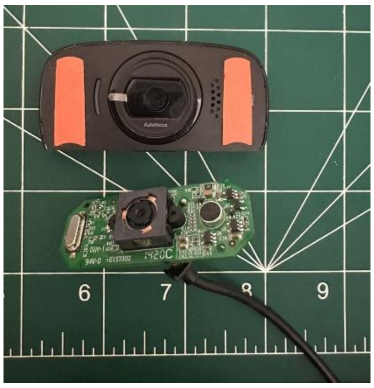


Figure 3.10: World Camera PCB

Table 3.1: Specifications of Logitech C525

Particulars	**Specifications**
Sensor Type	CMOS
Frame Rate	30 fps
Drive Type	Plug and Play
Resolution	1280 × 720 p
Built-in Microphone	Yes
Focus Range	48.94 mm
Net Weight	88 g

As for the eye camera, other researchers have used the Intel RealSense T265 and the Mercury Viewcam Zoom10. The former is considered too expensive, while the latter is no longer available in the market. Therefore, the Microsoft Lifecam HD-6000, suggested by Pupil Labs, is considered for implementation as the eye camera. Figure 3.11 shows the Microsoft Lifecam HD-6000. However, this model is not available in Malaysia and can only be purchased through eBay or Amazon, incurring higher costs and long lead time. Upon arrival, the camera module is disassembled. Unfortunately, the eye camera sustained damage while being disassembled. Originally, the plan was to remove the little wires from the PCB by melting the solder before putting them back together after removing the shell. However, it seems that the use of extremely high temperatures from the soldering gun may have caused the deterioration of the protective mask on the PCB. As a result, the eye camera malfunctions when connected to the laptop.

Figure 3.11: Microsoft Lifecam HD-6000

Due to the possibility of prolonged shipping periods causing delays and affecting the execution of the project, an alternative solution has been found and obtained. This alternate camera, A03 PC1082, obtained from Shopee and represented in Figure 3.12, offers a cost-efficient choice. Although the camera is affordable, it possesses qualities that are considered appropriate for the eye tracker's intended use, namely in terms of resolution and frame rate. The camera's particular details are shown in Table 3.2.

Figure 3.12: A03 PC1082 Camera

Table 3.2: Specifications of A03 PC1082 Camera

Particulars	Specifications
Sensor Type	CMOS
Frame Rate	30 fps
Drive Type	Plug and Play
Resolution	1920 × 1080 px
Built-in Microphone	Yes
Focus Range	20 mm
Net Weight	62.8g
Cost	RM 30.40 (After Discount)

After the A03 PC1082 camera module arrived, it is taken apart and separated into its PCB, as shown in Figure 3.13. The camera PCB has a square configuration, resembling that of the Microsoft Lifecam HD-6000. Although it is smaller in size, making it acceptable for use as an eye camera, it needed to be removed from its case for the initial inspection. Afterwards, the camera lens is removed from the PCB by rotating it counterclockwise. Subsequently, the thin glass pane that constituted the infrared (IR) filter is eliminated. After successfully removing the IR filter, as shown in Figure 3.14, the camera lens is then reinstalled and adjusted until achieving the best focus on the laptop, as displayed in Figure 3.15.

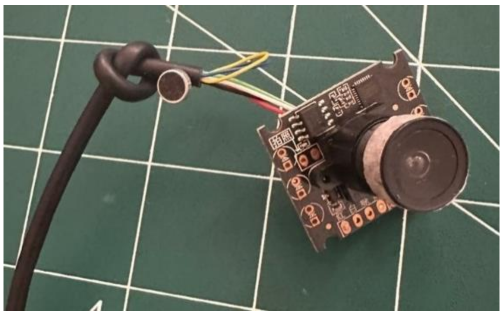


Figure 3.13: Eye Camera PCB

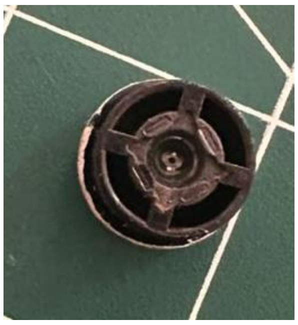


Figure 3.14: Camera Lens with Removed IR Filter

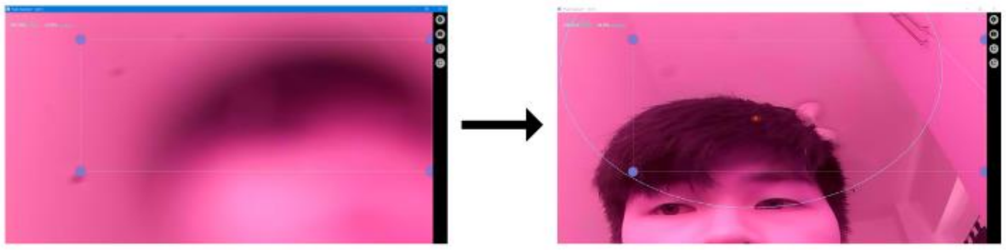


Figure 3.15: Focus Adjustment during Reinstallation of Camera Lens

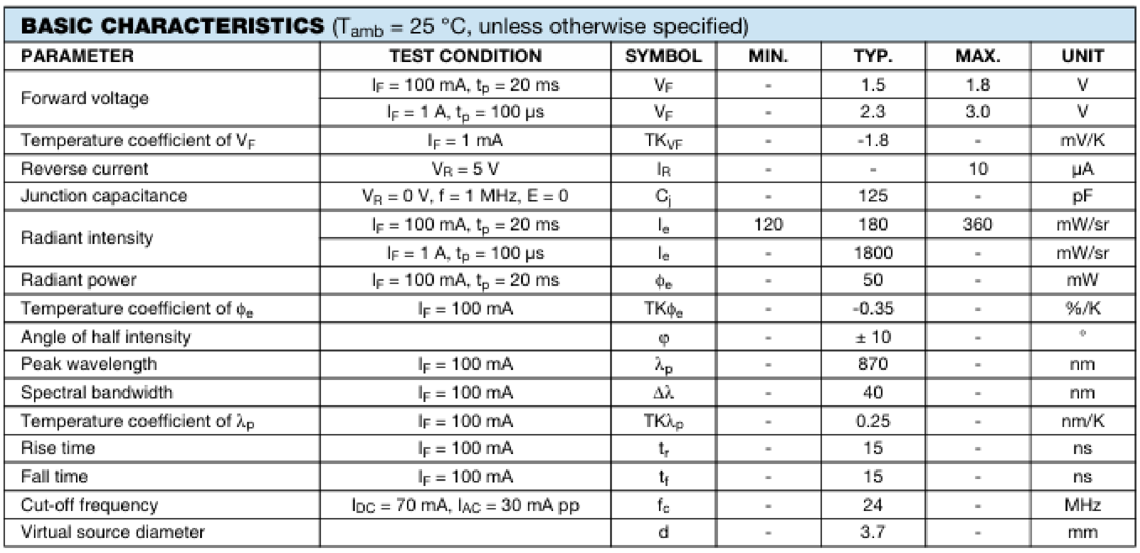
3.4.3 Infrared (IR) LED

In order to obtain the intended corneal reflection for the purpose of eye tracking, a near-infrared LED light is strategically placed next to the camera that captures images of the eye. After considering several alternatives discussed in the literature review, the SFH 4050-Z infrared (IR) Emitter (850nm) is identified as the best candidate for integrating onto the PCB of the eye camera. It is important to mention that the SFH 4050-Z IR emitter is a surface-mount device (SMD), which means it must be directly mounted onto the PCB surface.

When the Microsoft Lifecam HD-6000 is replaced with the A03 PC1082, the IR LED is replaced with the TSFF5210, as seen in Figure 3.16. The TSFF5210 has a peak wavelength that is 20 nm higher than its predecessor, measuring at 870 nm. Additionally, it has a higher radiant intensity of 15 mW/Sr. Although there is an 11 mW/Sr increase compared to its counterpart, the radiant intensity of 15 mW/Sr is still significantly below the acceptable limit of 4 W/Sr, especially when measured at a distance of 0.2 m. The specifications are tabulated in Table 3.3.

Figure 3.16: TSFF5210 IR LED

Table 3.3: Specifications of TSFF5210



BASIC CHARACTERISTICS (T_{amb} = 25 °C, unless otherwise specified)

PARAMETER	TEST CONDITION	SYMBOL	MIN.	TYP.	MAX.	UNIT
Forward voltage	I_F = 100 mA, t_p = 20 ms	V_F	-	1.5	1.8	V
	I_F = 1 A, t_p = 100 μs	V_F	-	2.3	3.0	V
Temperature coefficient of V_F	I_F = 1 mA	TK_{VF}	-	-1.8	-	mV/K
Reverse current	V_R = 5 V	I_R	-	-	10	μA
Junction capacitance	V_R = 0 V, f = 1 MHz, E = 0	C_j	-	125	-	pF
Radiant intensity	I_F = 100 mA, t_p = 20 ms	I_e	120	180	360	mW/sr
	I_F = 1 A, t_p = 100 μs	I_e	-	1800	-	mW/sr
Radiant power	I_F = 100 mA, t_p = 20 ms	ϕ_e	-	50	-	mW
Temperature coefficient of ϕ_e	I_F = 100 mA	$TK\phi_e$	-	-0.35	-	%/K
Angle of half intensity		φ	-	± 10	-	°
Peak wavelength	I_F = 100 mA	λ_p	-	870	-	nm
Spectral bandwidth	I_F = 100 mA	$\Delta\lambda$	-	40	-	nm
Temperature coefficient of λ_p	I_F = 100 mA	$TK\lambda_p$	-	0.25	-	nm/K
Rise time	I_F = 100 mA	t_r	-	15	-	ns
Fall time	I_F = 100 mA	t_f	-	15	-	ns
Cut-off frequency	I_{DC} = 70 mA, I_{AC} = 30 mA pp	f_c	-	24	-	MHz
Virtual source diameter		d	-	3.7	-	mm

The initial stage in commencing the development of the IR circuit on the eye camera PCB entails assessing the existing vias on the PCB. A multimeter is used to verify the continuity of the vias, thus determining the presence of a "ground" connection. After identifying the "ground" terminal, the multimeter is used to test the remaining non-grounded vias, measuring the potential difference between the positive and ground terminals. Therefore, the measured potential difference is 4.70 V, as shown in Figure 3.17. Afterwards, the desired resistance is determined using Equation 3.1.

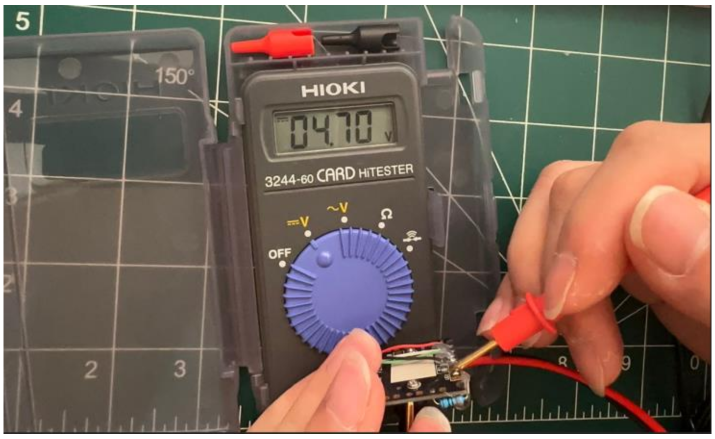


Figure 3.17: Measurement of Supply Voltage using Multimeter

$$R = \frac{V_S - V_F}{I_F} \tag{3.1}$$

where

R = Resistance, **Ω**

V_S = Supply voltage, V

V_F = LED forward voltage, V

I_F = LED forward current, A

According to the Vishay datasheet, the highest voltage that can be applied to the LED is 1.5 V, and the highest current that can flow through it is 100 mA. In order to guarantee proper operation within acceptable limits, the required resistance is calculated using a forward voltage of 1.4 V and a forward current of 10 mA. Therefore, a resistor with a value of 330 ohms is determined. Using this specific resistor value, the circuit is attached to the PCB, as shown in Figure 3.18. Afterwards, the webcam is linked to the laptop and examined for the functionality of the IR LED. The outcomes are depicted in Figure 3.19. The modifications on the eye camera PCB are completed after the implementation of IR circuit.

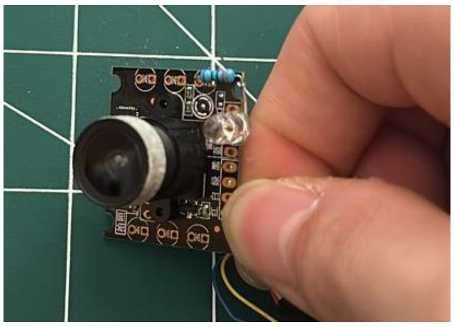


Figure 3.18: Eye Camera PCB with IR LED Circuit Implementation

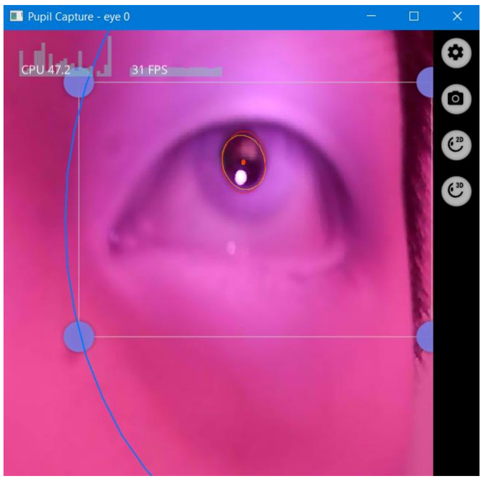


Figure 3.19: Illumination of IR on Pupil

3.4.4 Customized Mounting Brackets and Casings

The mounting brackets are crucial elements in the eye tracker system, providing vital support for securely fastening the camera PCBs in position. The absence of these brackets would impair the functionality of the eye tracker. For this project, Autodesk Fusion 360 is employed to create the mounting brackets, which are then produced using a 3D printer. Every camera module requires its own distinct mounting bracket and housing.

The world camera bracket and housing consist of four carefully designed sub-components that enable both modularity and effectiveness in securely holding the globe camera. The generation of a modular design serves two primary purposes: to streamline the fabrication process, as 3D printing is contingent upon printing orientation, and to facilitate the effortless removal of the world camera module for storage when the eye tracker is not in use. In order to make the design process more efficient, it is initiated by creating a rough drawing and model of the main structure of the globe camera PCB, as shown in Figure 3.20.

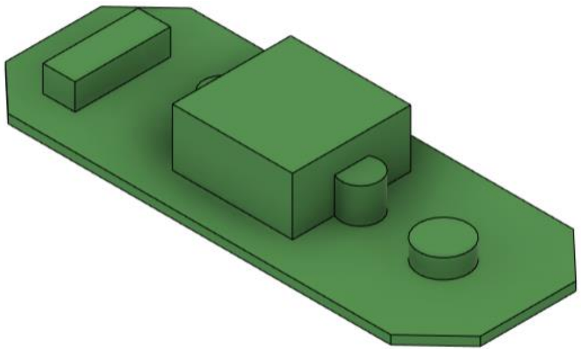
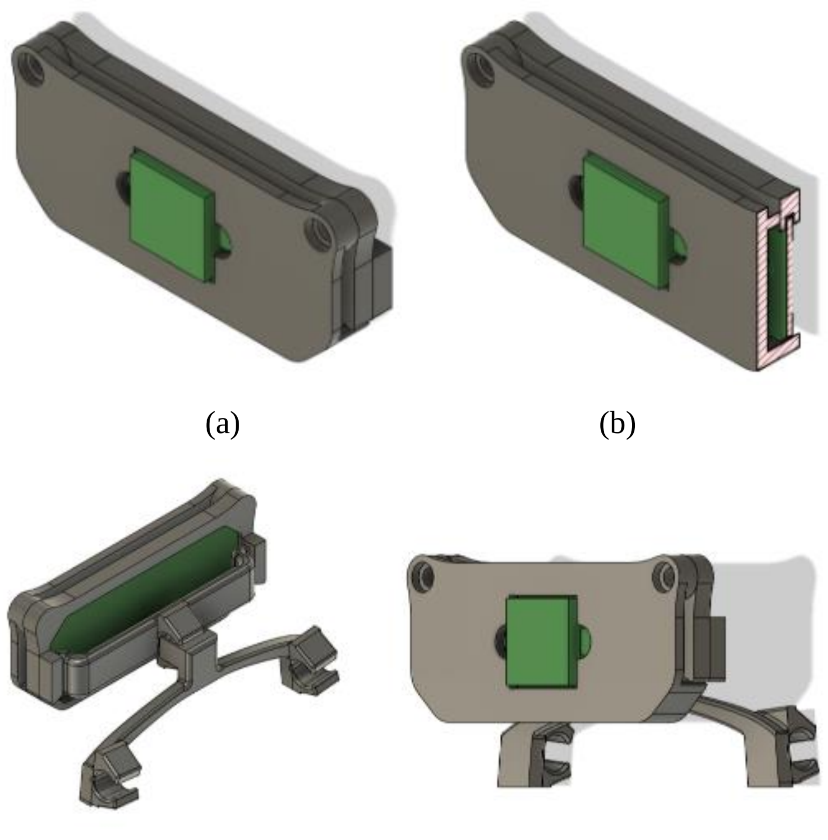


Figure 3.20: Sketch of World Camera PCB

The casing for the global camera PCB is intricately built with a force-fit slot design, as depicted in Figure 3.21(a) and (b). The objective of this slot design is to provide a tight and secure placement, effectively gripping the slender edge of the PCB within the enclosure. In addition, the design includes two screw holes to enhance the longevity of the casing.

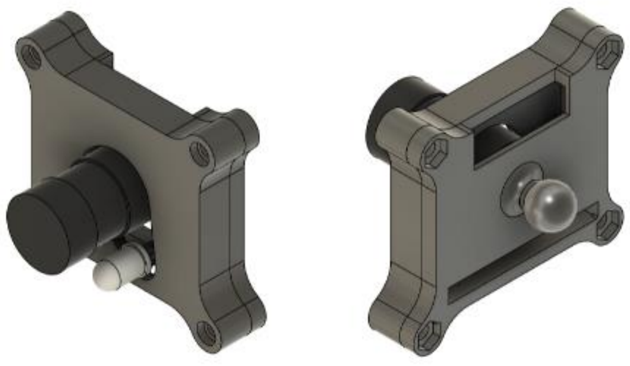
Afterwards, the mounting brackets are fabricated utilizing the C-clip and slot design. The casing holder is intended to be inserted into the rear of the casing, with sufficient clearance for the webcam cable to pass through. The C-clip can be secured to the curved mounting bracket, as shown in Figure 3.21(c) and (d). Ultimately, the C-clip of the curved bracket is affixed onto the frame of the face shield, so concluding the procedure of assembling.


(a) (b)

(c) (d)

Figure 3.21: Overall Design of World Camera Mounting Bracket: (a) Casing Design, (b) Section Analysis, (c) Rear View and (d) Front View

For the eye camera, the design of the casing and mounting bracket are more complicated than that of the world camera as the world camera is fixed at its position while the eye camera should be dynamic and adjustable. This is aimed to focus the camera on the pupil and ensure the pupil captured is within the field of view (FOV) of the eye camera. As the PCB is square-shaped, the casing is designed to have a front and back body with 4 screw holes for tightening. There is a ball at the back of the rear body for the implementation of a ball joint. The middle part of the front body is removed as the camera lens on the PCB is protruding. There are some clearances for the cable to pass through. The front and back view of the eye camera casing are shown in Figure 3.22.



(a) (b)

Figure 3.22: Casing for Eye Camera PCB: (a) Front View and (b) Rear View

Subsequently, the ball joint is constructed with two identical components, as seen in Figure 3.23. These components are designed to enable the ball to be securely held by the curved surfaces of the holes on both sides, while also allowing for rotational flexibility. Furthermore, the design incorporates a screw hole that allows for the adjustment of the tension of the ball joint, providing the most effective functionality.

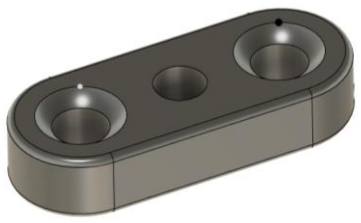


Figure 3.23: Joint Part

Following this, a male extender is designed with a spherical object at one end and a C-shaped clip positioned at the centre, enabling the eye camera wire to be firmly fastened. Figure 3.24 displays this design. The overall ball joint mechanism is illustrated in Figure 3.25.

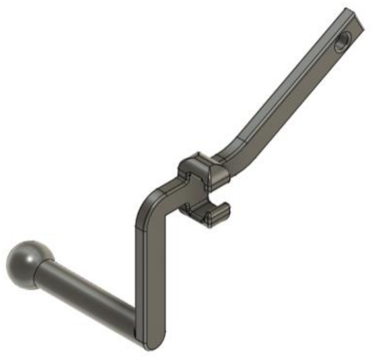


Figure 3.24: Male Extender with Ball Joint

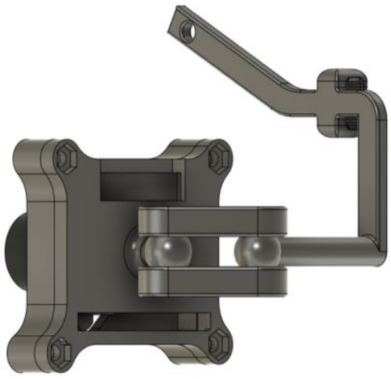


Figure 3.25: Overall Ball Joint Mechanism

The final component of the mounting bracket design is the female extender, which serves as the link between the male extender and the eye tracker frame. The initial design, depicted in Figure 3.26, utilized only C-clips for mounting onto the frame. However, relying solely on C-clips proved insufficient in providing the necessary durability to support the weight of the entire eye camera system. Consequently, the female extender tended to become loose over time, causing it to slant downward.

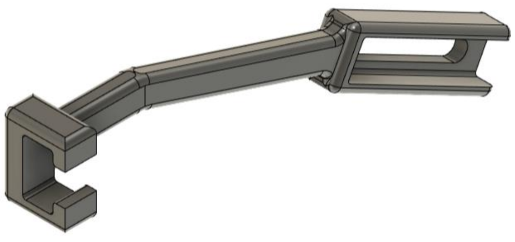


Figure 3.26: Initial Design of Female Extender

Figure 3.27 illustrates the comparison between the initial position and the slanted position before and after the placement of the eye camera, respectively. This issue underscores the importance of ensuring robust support for the eye camera assembly to maintain stability and functionality throughout operation.

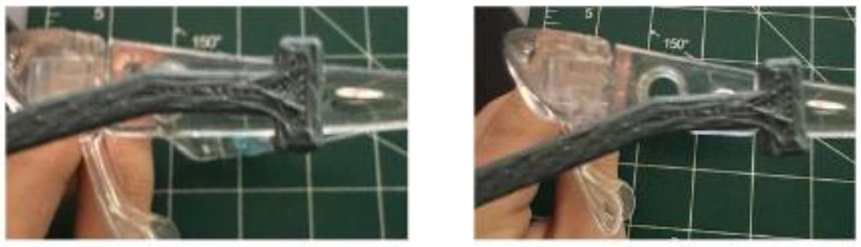


Figure 3.27: Placement of Female Extender: a) Initial Position Before Placement and b) Position After Placement

Consequently, a novel design has been adopted to tackle these concerns. After examining a circular opening on the frame, as shown in Figure 3.27b, the updated design includes an extruded cylinder that fits tightly into this opening. The addition, in conjunction with the C-clip, efficiently prevents the female extension from becoming misaligned following the placement of the eye camera. Figure 3.28 shows the final female extender design.

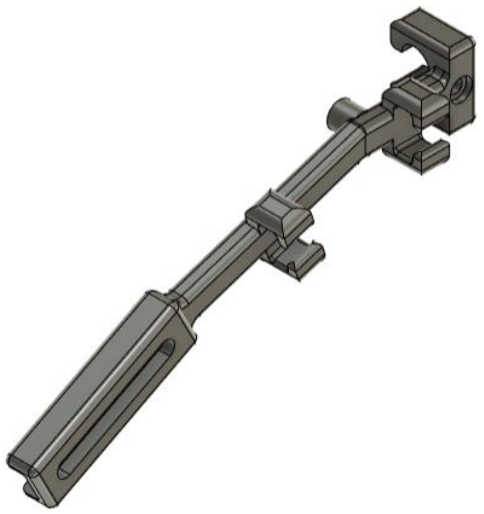


Figure 3.28: New Female Extender Design

In addition, the updated design incorporates C-clips that run along the whole body of the mounting bracket, making it easier for both the eye and world camera wires to flow through in an organized manner. This guarantees that the eye tracker retains a neat and organized appearance, free from any dangling wires. Figure 3.29 depicts the completed design of the female extension.

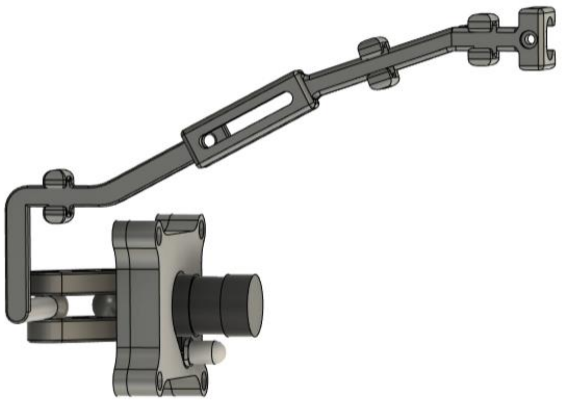


Figure 3.29: Overall Design of Eye Camera Mounting Bracket

3.5 Software Development

The software development process begins with the installation of Pupil Capture v3.5.1 software. Afterwards, the calibration setup is carried out, followed by the modification and implementation of the cursor control plugin API. After integrating the plugin, a user-friendly interface is created to facilitate testing and data gathering. Ultimately, the process of analysing and presenting data is conducted. Figure 3.30 depicts the flowchart representing the process of software development.

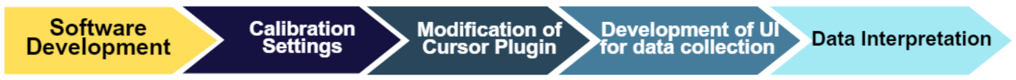

Software Development
Calibration Settings
Modification of Cursor Plugin
Development of UI for data collection
Data Interpretation

Figure 3.30: Flowchart of Software Development

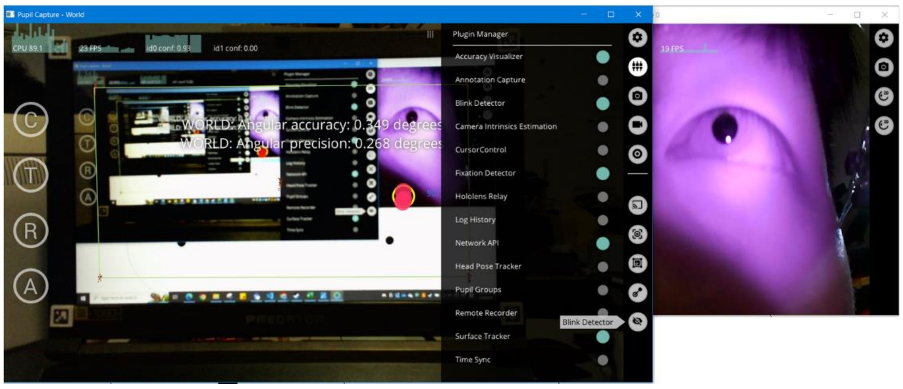
3.5.1 Gaze Detection Software Setup

For precise identification, monitoring, and logging of pupil and gaze information, it is crucial to seamlessly combine hardware and software applications. The primary eye-tracking software chosen for this project is Pupil

Capture v3.5.1 software. Pupil Capture, created by Pupil Labs, provides a wide range of features, such as calibration and recording capabilities. The system is capable of working with both single and dual-eye camera setups. However, for this project, a monocular eye tracker configuration is implemented.

Pupil Capture v3.5.1 features a variety of pre-installed plugin APIs, including a fixation detector, an accuracy visualizer, and a blink detector. These plugins allow for the creation of real-time data, making it easier to carry out a range of application tasks. However, even though the Pupil Player program, which allows for the conversion of recordings into CSV data, is available, it is not being used in this project. The decision is made taking into account the application's real-time nature for controlling the mouse cursor.

Pupil Capture v3.5.1 does not directly recognize regular webcams connected to a laptop because of compatibility problems with USB drivers. Through extensive online investigation and examination of the resources available on the Pupil Labs GitHub page, a solution offered by a user is discovered. To resolve this issue, the appropriate driver, which is the *libusbk 3.0.7.0* driver, is downloaded and installed. Afterwards, the *Zadig* application, specifically developed for Windows, iss used to change the driver from WinUSB to *libusbk 3.0.7.0*. Pupil Capture effectively identified and enabled the webcam by following the specified procedures, as illustrated in Figure 3.31.



WORLD: Angular accuracy: 0.349 degrees
WORLD: Angular precision: 0.268 degrees

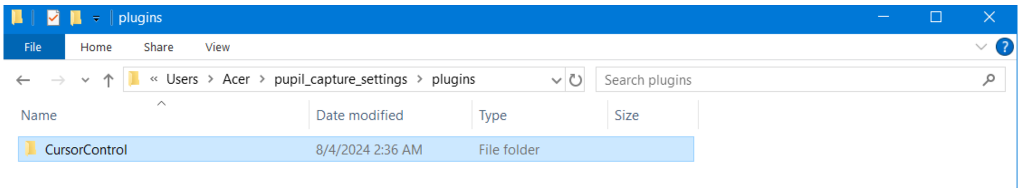
Figure 3.31: Successful Detection of World and Eye Cameras

3.5.2 Cursor Control Plugin

Integrating a plugin API with Pupil Capture v3.5.1 is necessary in order to offer cursor control for exploring the laptop screen and executing clicks. This project involves the examination and customization of a cursor control plugin API, which has been contributed by the community on Pupil Labs GitHub. The goal is to adapt the API to meet the specific needs of the cursor control application. The cursor control plugin API, created by Emendir in 2022, serves as the foundation of the cursor control system.

The cursor control application begins by installing physical *AprilTags* from the tag16h5 family into the corners of the screen. According to Olson (2011), *AprilTag* is a visual fiducial system that has several applications such as augmented reality, robotics, and camera calibration. Visual fiducial systems, in contrast to typical 2D barcode systems, not only capture the barcode in an image but also offer accurate details about its location and orientation in relation to the camera. Fiducial systems are specifically designed to detect and classify several markers within a single image.

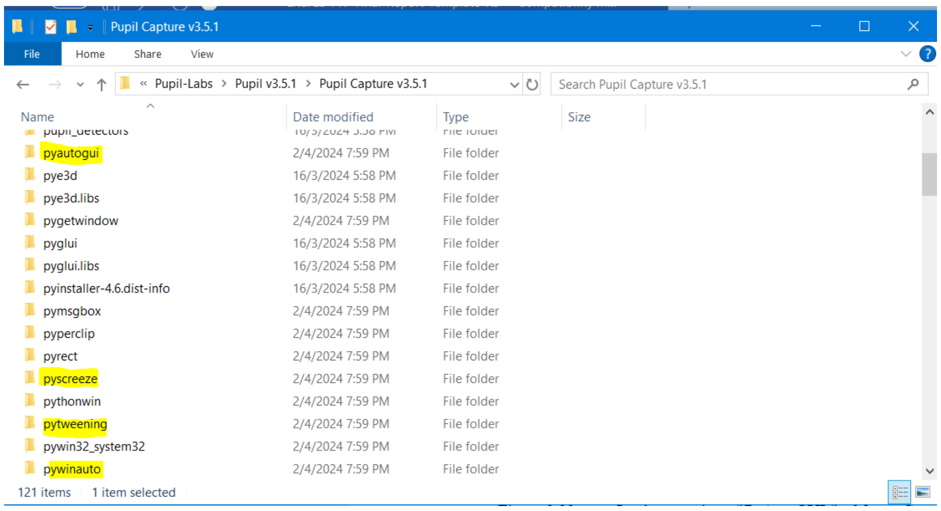
To add the cursor control plugin to the Pupil Capture v3.5.1, the python files are moved to the plugin folder under the directory of "pupil_capture_settings", as shown in Figure 3.32. When the cursor control plugin is activated, a calibration screen appears with four virtual *AprilTags* belonging to the tag36h11 family. After detecting all *AprilTags* with the world camera using *OpenCV*, the plugin then proceeds to estimate the corners and edges of the screen. This ultimately leads to the creation of the screen border using the "draw_polyline" module from the *pyglui* package. Afterwards, the plugin employs the gaze locations identified by Pupil Capture v3.5.1 after calibration to forecast the gaze location in (x, y) pixel coordinates. In addition, the plugin has error handling to handle gaze locations that exceed the screen edge within specified tolerances. This error-handling technique ensures that if the gaze locations stay within the tolerance limit, the mouse cursor will be directed to the edge of the screen. Ultimately, the mean of the gaze positions is calculated in order to establish the ultimate cursor location.



Name	Date modified	Type	Size
CursorControl	8/4/2024 2:36 AM	File folder	

Figure 3.32: Addition of Cursor Control Plugin to Pupil Capture

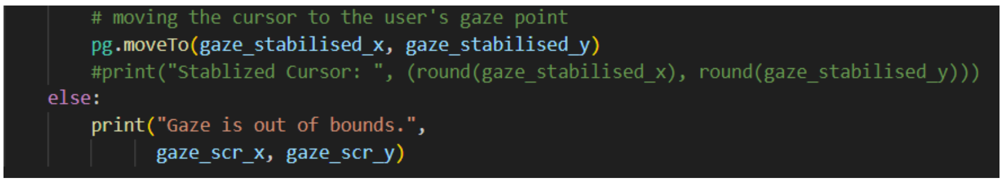
Nevertheless, specific alterations have been implemented to the plugin. The *PyAutoGUI* library is integrated into these improvements. When running Pupil Capture v3.5.1 in Python, the software is only able to recognize external libraries if they are located within the Pupil Capture package directory. Put simply, the traditional approach of installing Python libraries using pip does not instantly activate their integration within the software. The solution involves identifying the directory where the site packages are located, installing the necessary packages using pip, and then moving the installed packages into the directory as shown in Figure 3.33. After the directory transfer, *PyAutoGUI* is effectively incorporated into the plugin to enable cursor movement, as seen in Figure 3.34.


Pupil Capture v3.5.1
File
Home
Share
View
« Pupil-Labs › Pupil v3.5.1 › Pupil Capture v3.5.1
Search Pupil Capture v3.5.1

Name	Date modified	Type	Size
pupil_detectors	16/3/2024 5:58 PM	File folder	
pyautogui	2/4/2024 7:59 PM	File folder	
pye3d	16/3/2024 5:58 PM	File folder	
pye3d.libs	16/3/2024 5:58 PM	File folder	
pygetwindow	2/4/2024 7:59 PM	File folder	
pyglui	16/3/2024 5:58 PM	File folder	
pyglui.libs	16/3/2024 5:58 PM	File folder	
pyinstaller-4.6.dist-info	16/3/2024 5:58 PM	File folder	
pymsgbox	2/4/2024 7:59 PM	File folder	
pyperclip	2/4/2024 7:59 PM	File folder	
pyrect	2/4/2024 7:59 PM	File folder	
pyscreeze	2/4/2024 7:59 PM	File folder	
pythonwin	2/4/2024 7:59 PM	File folder	
pytweening	2/4/2024 7:59 PM	File folder	
pywin32_system32	2/4/2024 7:59 PM	File folder	
pywinauto	2/4/2024 7:59 PM	File folder	

121 items 1 item selected

Figure 3.33: Package Transfer to Pupil Capture Directory

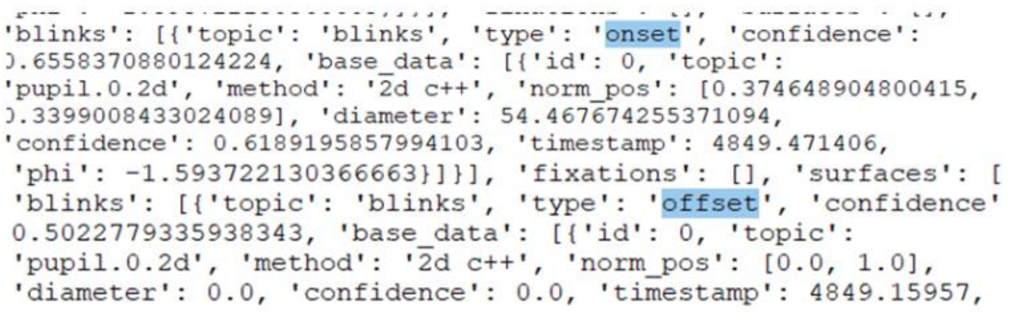


```
        # moving the cursor to the user's gaze point
        pg.moveTo(gaze_stabilised_x, gaze_stabilised_y)
        #print("Stablized Cursor: ", (round(gaze_stabilised_x), round(gaze_stabilised_y)))
    else:
        print("Gaze is out of bounds.",
              gaze_scr_x, gaze_scr_y)
```

Figure 3.34: Implementation of *PyAutoGUI* for Mouse Cursor Movement

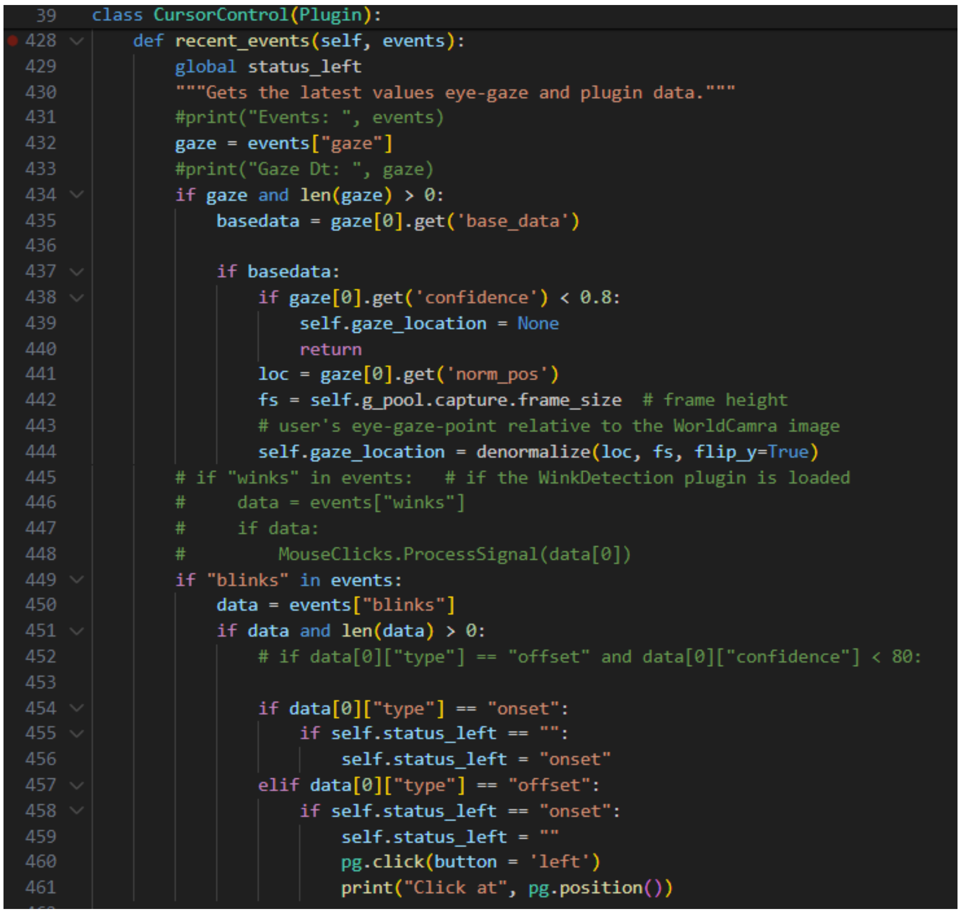
Another significant alteration relates to the clicking functionality within the plugin. Initially, the plugin employed another plugin called the Wink Detection plugin, which functioned with both eyes. Unfortunately, the Wink Detection plugin cannot be used in this particular situation. As a result, the code is modified to allow the click function by utilizing blinking and making use of the *PyAutoGUI* package. This revised method employs the "events" captured by Pupil Capture v3.5.1 in real-time.

When the pre-installed blink detection plugin is activated, the "blink" event includes values that indicate the occurrence of the user's eye blinks. Figure 3.35 provides an illustration of a recorded blink event. Using this blink event data, the code is modified to incorporate the new functionality. The revised code snippet is depicted in Figure 3.36.



```
'blinks': [{'topic': 'blinks', 'type': 'onset', 'confidence':
).6558370880124224, 'base_data': [{'id': 0, 'topic':
'pupil.0.2d', 'method': '2d c++', 'norm_pos': [0.374648904800415,
).3399008433024089], 'diameter': 54.467674255371094,
'confidence': 0.6189195857994103, 'timestamp': 4849.471406,
 'phi': -1.593722130366663}]}], 'fixations': [], 'surfaces': [
 'blinks': [{'topic': 'blinks', 'type': 'offset', 'confidence'
 0.5022779335938343, 'base_data': [{'id': 0, 'topic':
 'pupil.0.2d', 'method': '2d c++', 'norm_pos': [0.0, 1.0],
 'diameter': 0.0, 'confidence': 0.0, 'timestamp': 4849.15957,
```

Figure 3.35: Blink Events Captured by Pupil Capture



```python
class CursorControl(Plugin):
    def recent_events(self, events):
        global status_left
        """Gets the latest values eye-gaze and plugin data."""
        #print("Events: ", events)
        gaze = events["gaze"]
        #print("Gaze Dt: ", gaze)
        if gaze and len(gaze) > 0:
            basedata = gaze[0].get('base_data')

            if basedata:
                if gaze[0].get('confidence') < 0.8:
                    self.gaze_location = None
                    return
                loc = gaze[0].get('norm_pos')
                fs = self.g_pool.capture.frame_size  # frame height
                # user's eye-gaze-point relative to the WorldCamra image
                self.gaze_location = denormalize(loc, fs, flip_y=True)
        # if "winks" in events:   # if the WinkDetection plugin is loaded
        #     data = events["winks"]
        #     if data:
        #         MouseClicks.ProcessSignal(data[0])
        if "blinks" in events:
            data = events["blinks"]
            if data and len(data) > 0:
                # if data[0]["type"] == "offset" and data[0]["confidence"] < 80:

                if data[0]["type"] == "onset":
                    if self.status_left == "":
                        self.status_left = "onset"
                elif data[0]["type"] == "offset":
                    if self.status_left == "onset":
                        self.status_left = ""
                        pg.click(button = 'left')
                        print("Click at", pg.position())
```

Figure 3.36: Modified Code Snippet

In short, with the implementation of *PyAutoGui* library, the cursor control plugin is working as it could move the cursor based on the gaze locations and perform clicks using eye blinks.

3.5.3 Development of Dart Board User Interface

To evaluate the effectiveness of the cursor control plugin outlined earlier, it is necessary to examine accuracy and precision, which are the key concepts in eye-tracking world. Accuracy and precision, in this context, specifically pertain to the level of angular accuracy and angular precision. These concepts will be further explained in the following chapter.

The user interface consists of four equidistant circular dart boards displayed across the entire screen. Each dart board is decorated with a red dot at its centre, which serves as the target point for fixation. These dart boards, which function as circular buttons, can be clicked. When the user clicks a button, a symbol in the shape of a "X" should be displayed on the button to indicate that the click is successful. Areas beyond the bounds of the button are not able to be clicked. After clicking all the buttons in a specific order, the coordinates of the clicked locations in pixels are saved, and the angular accuracy and precision are calculated and shown.

The user interface development utilizes the pre-existing Python module called *tkinter* and is divided into two classes: the “ResultWindow” class and the “TestUI” class. The user interface is initially specified as “self.master”, with fullscreen settings, no title, and a shortcut key for dismissing the interface. Afterwards, a black canvas is made to conceal the window. The deliberate selection of a black background is based on the fact that the eye tracker's camera catches darker images when the screen is white, which is caused by contrast problems.

The “create_button” function is called to build buttons that are evenly positioned in the four quadrants of the screen. This function manages settings pertaining to the sizes, positions, and visual aspects of buttons and dots. Furthermore, it is associated with the “button_clicked” function to carry out tasks when buttons are clicked. The “button_clicked” function logs the positions that were clicked, adds a "X" mark at the clicked position, and then invokes the “calculate_accuracy” function.

The “calculate_accuracy” function determines the angular accuracy and precision, and then transfers them to the “ResultWindow” class for presentation. The computational logic of the “calculate_accuracy” function will be explained in detail in the following chapter. The entire code is provided in Appendix A. Figure 3.37 displays the graphical user interface (GUI) of dart board system for data collection.

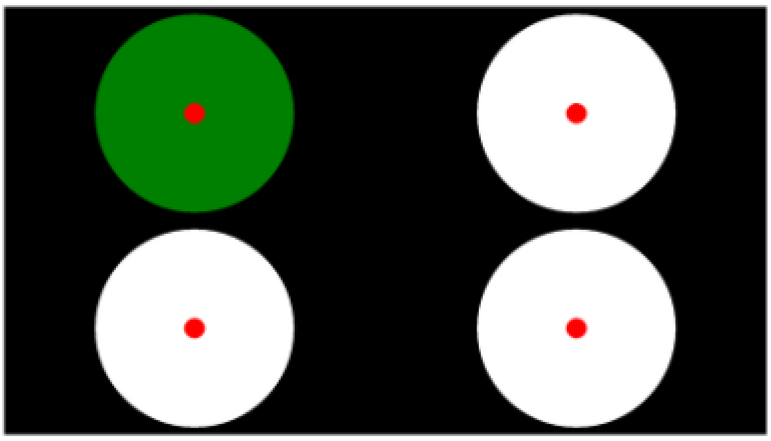


Figure 3.37: Graphical User Interface

3.5.4 Development of Virtual Keyboard

This project involves the development of a rudimentary virtual keyboard using the Python programming language. The keyboard allows users to input text by clicking on the mouse. Although virtual keyboards are widely used on cellphones, they are not as widespread on computers. Nevertheless, its implementation acts as a fundamental precursor for prospective future advancements. This virtual keyboard functions as an interactive graphical user interface by including gaze-controlled mouse cursor capability.

The virtual keyboard interface, as seen in Figure 3.38, is developed using the *tkinter* and *customtkinter modules*. The keyboard incorporates keys for numerals, letters of the alphabet, often employed punctuation symbols, and emojis, so aiding typing endeavours. In addition, functional keys like backspace, clear, arrow keys for typing cursor movements, and an escape button have been incorporated to improve the user experience.

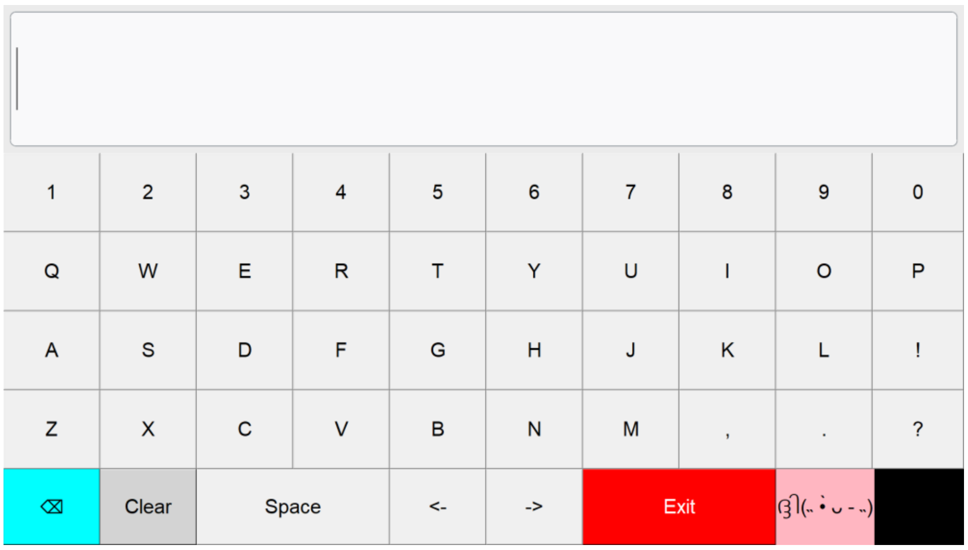


Figure 3.38: Virtual Keyboard Interface

The development process began by creating the “VirtualKeyboard” class. Once the class created, the main window is then personalized by adjusting the window settings such as deleting the title, activating fullscreen mode, and defining the backdrop colour. Afterwards, the area for typing is established, and a blinking cursor is activated to show where text can be entered. The keys are arranged in an array to facilitate the development of buttons, and distinct functions are assigned to the functional keys. As an illustration, a user-defined function called "def space(self)" places a blank space at the current position of the typing cursor. This function is triggered when the space button is clicked. In addition, a button with a dropdown menu of emojis appears when selected is created, enabling users to include emoticons in their content. The "xview_moveto()" predefined function is used to automatically scroll the display when entered words exceed the visible area. The keyboard controls this circumstance. Once these procedures have been accomplished, a basic virtual keyboard is now prepared for usage The completed code is shown in Appendix B.

3.6 Performance Evaluation

Prior to initiating the data gathering procedure, the installation of *AprilTags* on the laptop screen is required. Afterwards, the process of collecting data begins by wearing the head-mounted eye tracker and connecting the camera modules as shown in Figure 3.39. Pupil Capture v3.5.1 is launched, and the user aligns the eye camera to verify that the pupil is within the camera's field of view (FOV). Next, the user places their chin on a chin rest and verifies that the laptop screen is within the field of view (FOV) of the world camera.

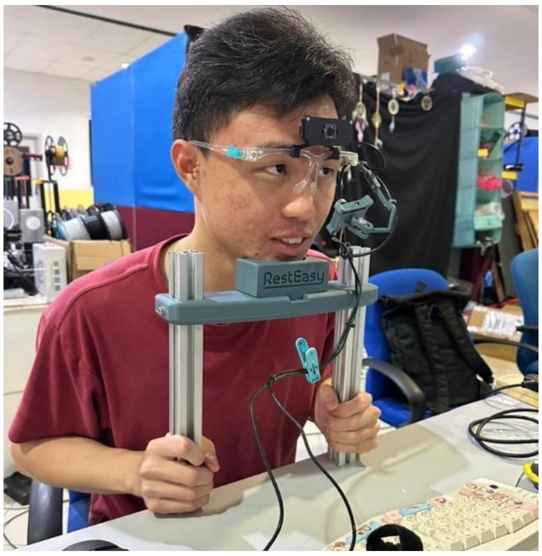


Figure 3.39: Wearing Eye Tracker

Subsequently, the calibration settings are adjusted for 2D gaze mapping, and the calibration process is started by pressing the "C" key. Screen markers are displayed in a manner that corresponds to their intended purpose, and the user's gaze should align with these markers as shown in Figure 3.40. After the calibration process is finished, the angular accuracy and precision are documented.

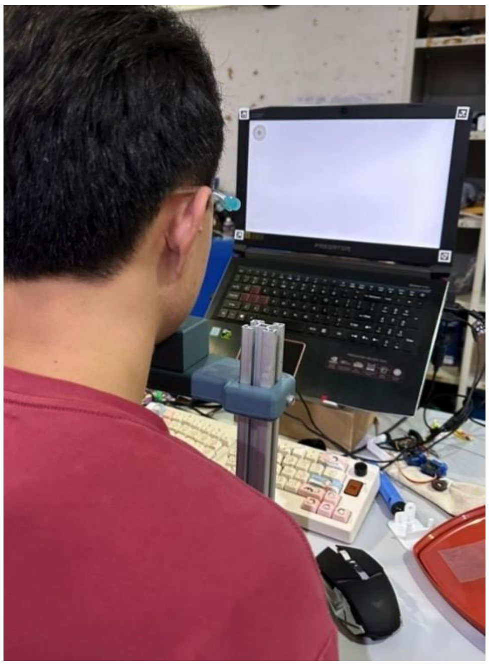


Figure 3.40: Calibration Process

The cursor control plugin is enabled by choosing it in the Plugin Manager of the Pupil Capture software. Upon detecting all the *AprilTags*, the globe camera quickly displays and then removes a calibration screen. The cursor control application starts, causing the mouse pointer to begin moving, and then the user interface is activated.

The user is directed to focus on the red dot positioned at the upper-right corner, causing the mouse cursor to adjust its movement accordingly. To execute a left click, the user rapidly closes and opens his eyes, causing the button to change color to green and leaving a "X" symbol in yellow at the clicked location as shown in Figure 3.41. The user iterates through these stages in a clockwise manner, directing their attention to each red dot until all four buttons are pressed.

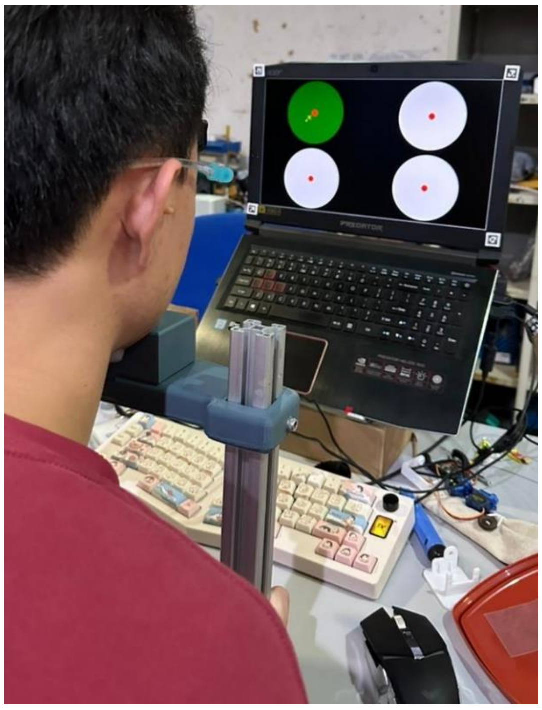
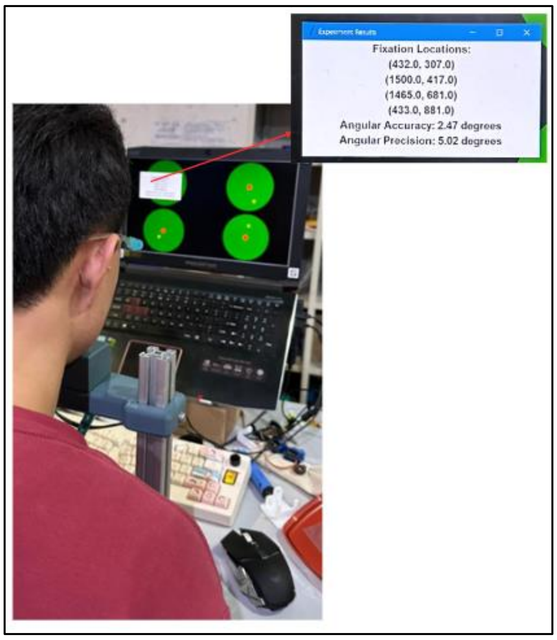


Figure 3.41: Interaction with User Interface (UI)

After finishing the task, a fresh window will emerge, presenting the recorded clicked coordinates in pixels, along with the angular accuracy and precision as shown in Figure 3.42. This represents a single endeavour. The user must repeat these procedures from the activation of UI three times for tasks related to immobility, facial expression, and speech. The immobility task involves the user performing cursor motions and clicks without any movement of his head. During the facial expression task, the user must manipulate the cursor while displaying a smile or other facial emotions. During the speech task, the user must talk or sing while directing the cursor.



```
Fixation Locations:
(432.0, 307.0)
(1500.0, 417.0)
(1465.0, 681.0)
(433.0, 881.0)
Angular Accuracy: 2.47 degrees
Angular Precision: 5.02 degrees
```

Figure 3.42: Data Display

3.6.1 Angular Accuracy and Angular Precision

For the result of this project, the main elements include angular accuracy and angular precision. According to the Pupil Core official GitHub page, it is noted that angular accuracy refers to the average angular offset (distance) (in degrees of visual angle) between fixation locations and the corresponding locations of the fixation targets. The average offset is measured in unit degree (°), which higher average offset indicates lower angular accuracy. In this application, the angular accuracy is determined and collected using the dart board GUI as mentioned in the previous chapter. The fixation target is the red dot while the fixation location is the clicked coordinates. There are four dart boards where the user must concentrate on each centre and click after each focus. Subsequently, the coordinates of the targets and clicked positions will be converted into visual angle coordinates. Hence, the visual angle per pixel is computed using Equation 4.1.

$$\theta = 2 \times \tan^{-1}\left(\frac{S}{2D}\right) \times \frac{180}{\pi} \div Re \qquad (4.1)$$

where

θ = Visual angle per pixel, °/px

S = Screen size, cm

D = Viewing distance, cm

Re = Resolution, px

Then, the pixel coordinates are converted into visual angle coordinates using Equation 4.2.

$$(x_V, y_V) = (x \times \theta, y \times \theta) \tag{4.2}$$

where

x = x coordinate in pixel

y = y coordinate in pixel

θ = Visual angle per pixel, °/px

Afterwards, the angular offset between each clicked position and its centre is calculated. The angular offset on each dart board is computed using the formula of Euclidean Distance, which is represented by Equation 4.3.

$$\theta_i = \sqrt{(x_{V2} - x_{V1})^2 + (y_{V2} - y_{V1})^2} \tag{4.3}$$

where

θ_i = Angular offset, °

x_{V2} = x coordinate of fixation location, °

y_{V2} = y coordinate of fixation location, °

x_{V1} = x coordinate of fixation target, °

y_{V1} = y coordinate of fixation target, °

Lastly, the angular offsets from 4 dart boards are added together and the average angular offset is computed. The average angular offset is calculated using the formula of mean, as shown in Equation 4.4, where n = 4. The lower the average angular offset calculated, the higher the angular accuracy.

$$\bar{\theta} = \frac{\sum \theta_i}{n} \tag{4.4}$$

where

$\bar{\theta}$ = Average angular offset, °

Θi = Angular offset, °

n = Number of samples

For the computation of angular precision, the angular precision is calculated as the Root Mean Square (RMS) of the angular distance (in degrees of visual angle) between successive samples. The lower the RMS value, the higher the precision. To obtain the RMS value, firstly, the angular offset between the centre and the clicked location is calculated. Then, the 4 dart boards with 4 clicked positions are combined and remapped, resulting in a single dart board with 4 clicked positions. Subsequently, the precision is computed using formula shown in Equation 4.5.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\theta_i)^2} \qquad (4.5)$$

where

σ = Root mean square of the angular offset, °

n = Number of samples

θi = Angular offset between successive samples, °

3.7 Summary

In the methodology section, the detailed development process of the head-mounted eye tracker system is covered. It includes project planning and milestones, hardware development and software development, performance evaluation. This methodology serves as the foundation for future improvements or development of the project.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, the process of data collection, results in terms of data representation and interpretation, as well as discussion about the results are highlighted.

4.2 Prototype

After completing hardware development, the prototype of the head-mounted eye tracker is successfully fabricated. It consists of a face shield frame as the backbone, a world camera on the top and an eye camera at the bottom with a ball joint and an extender, as shown in Figure 4.1. The total weight of the prototype is 125 g as shown in Figure 4.2.

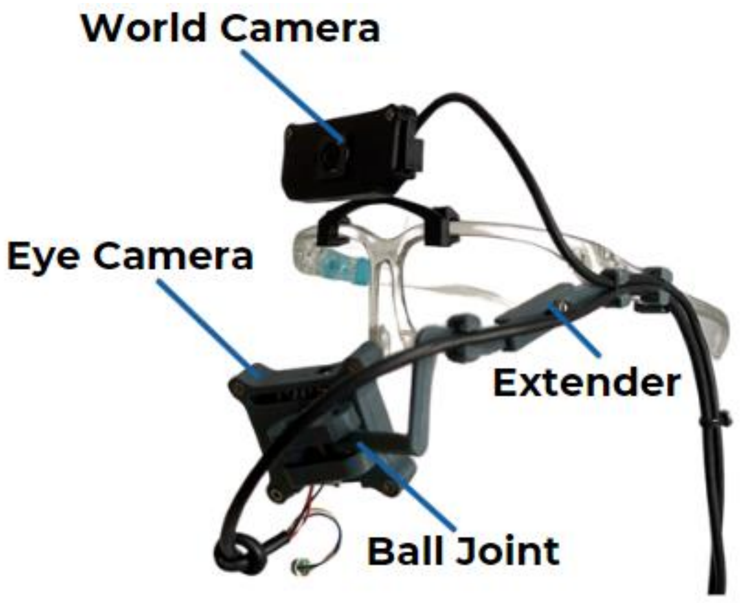


Figure 4.1: Prototype of the Head-mounted Eye Tracker

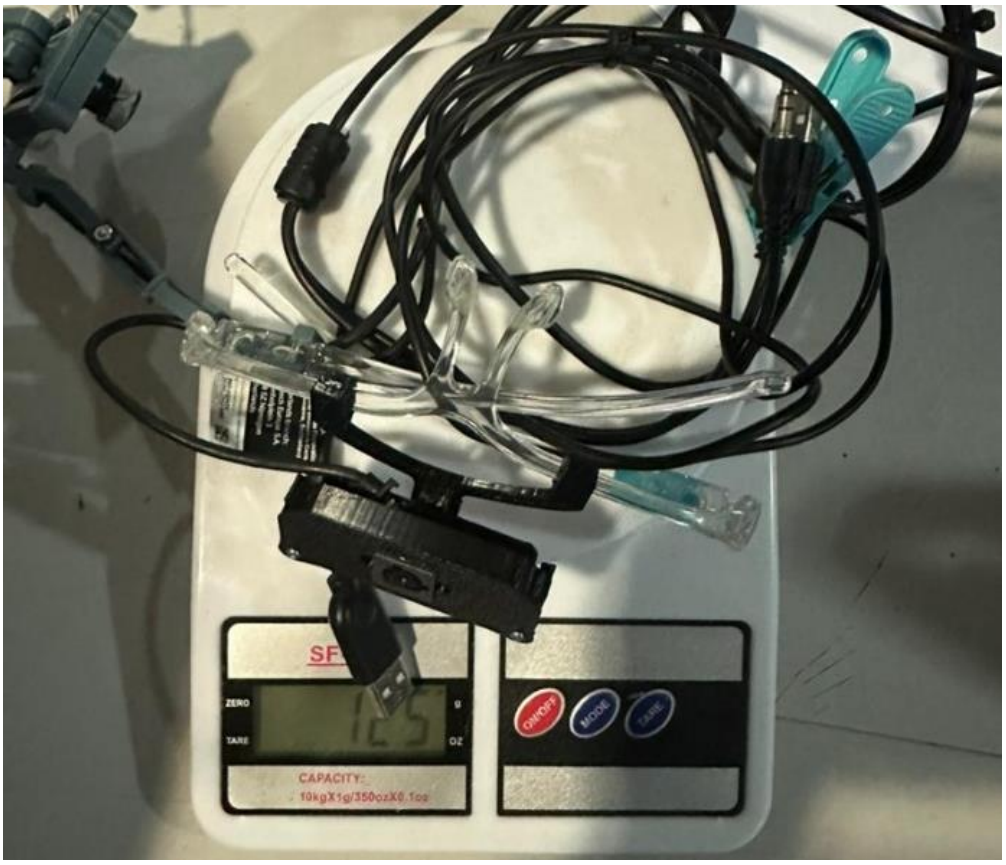
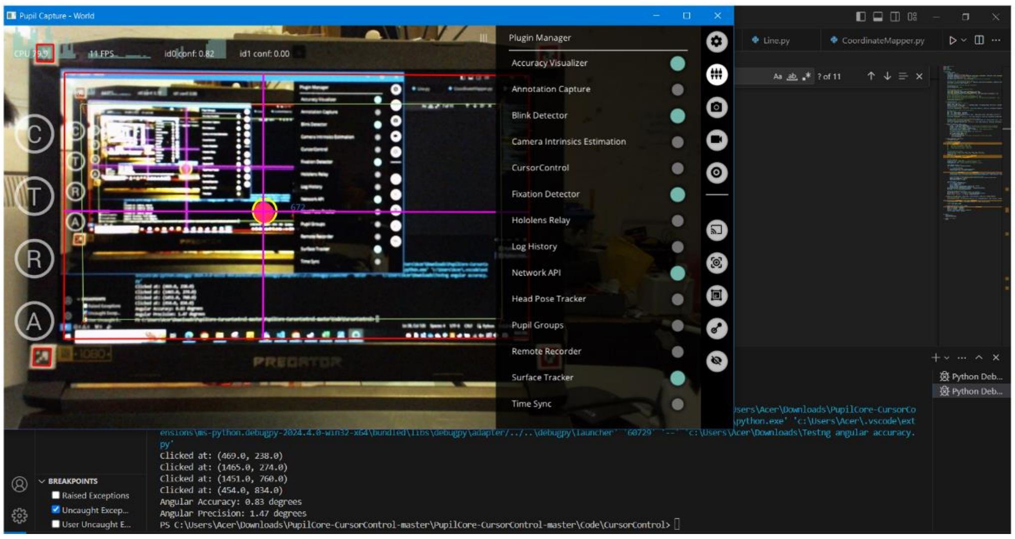


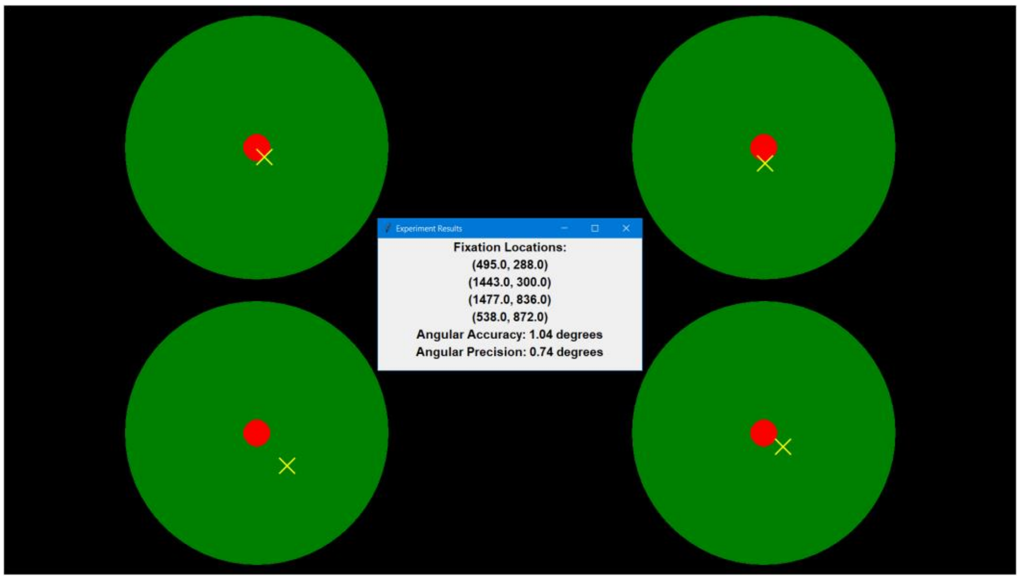
Figure 4.2: Total Weight of Prototype

The cursor control plugin API correctly detects the *AprilTag* and determines the place where the user's eyes are looking by identifying the junction point of the lines. Figure 4.3 depicts the activation of the cursor plugin. Figure 4.4 displays the user interface used for data collecting. It has the ability to gather data from user interactions. In addition, the gaze-controlled mouse cursor has the capability to execute typing by utilising the virtual keyboard. This is useful for those people who have motor disability to voice out their needs through typing. Figure 4.5 displays the graphical user interface (GUI) of the virtual keyboard with written phrases.



```
Users\Acer\Downloads\PupilCore-CursorCo
python.exe' 'c:\Users\Acer\.vscode\ext
ensions\ms-python.debugpy-2024.4.0-win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '60729' '--' 'c:\Users\Acer\Downloads\Testng angular accuracy.
py'
Clicked at: (469.0, 238.0)
Clicked at: (1465.0, 274.0)
Clicked at: (1451.0, 760.0)
Clicked at: (454.0, 834.0)
Angular Accuracy: 0.83 degrees
Angular Precision: 1.47 degrees
PS C:\Users\Acer\Downloads\PupilCore-CursorControl-master\PupilCore-CursorControl-master\Code\CursorControl> 
```

Figure 4.3: Activation of Cursor Control Plugin



Fixation Locations:
(495.0, 288.0)
(1443.0, 300.0)
(1477.0, 836.0)
(538.0, 872.0)
Angular Accuracy: 1.04 degrees
Angular Precision: 0.74 degrees

Figure 4.4: Graphical User Interface for Data Collection

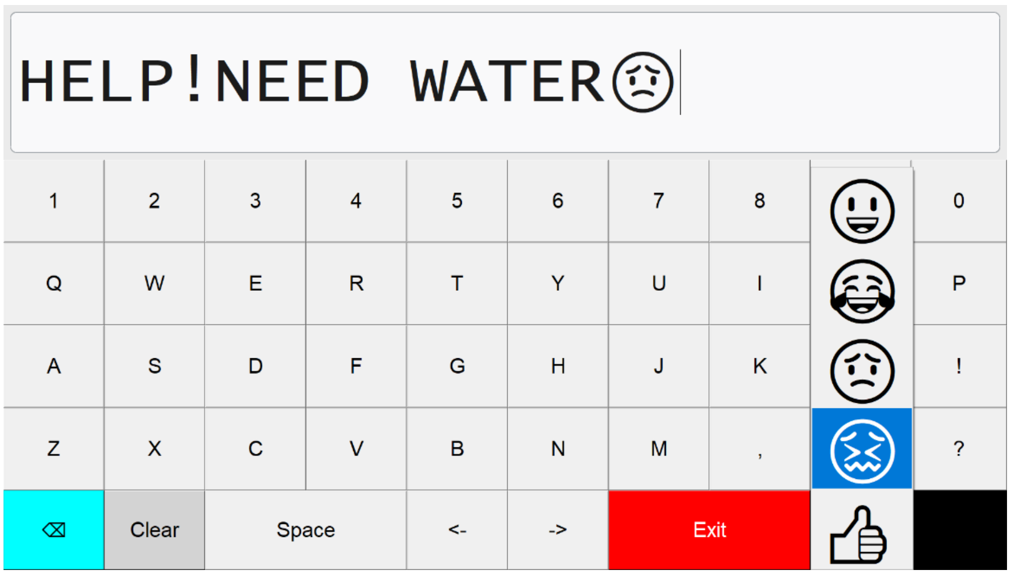


Figure 4.5: Typing Virtual Keyboard using Mouse Cursor

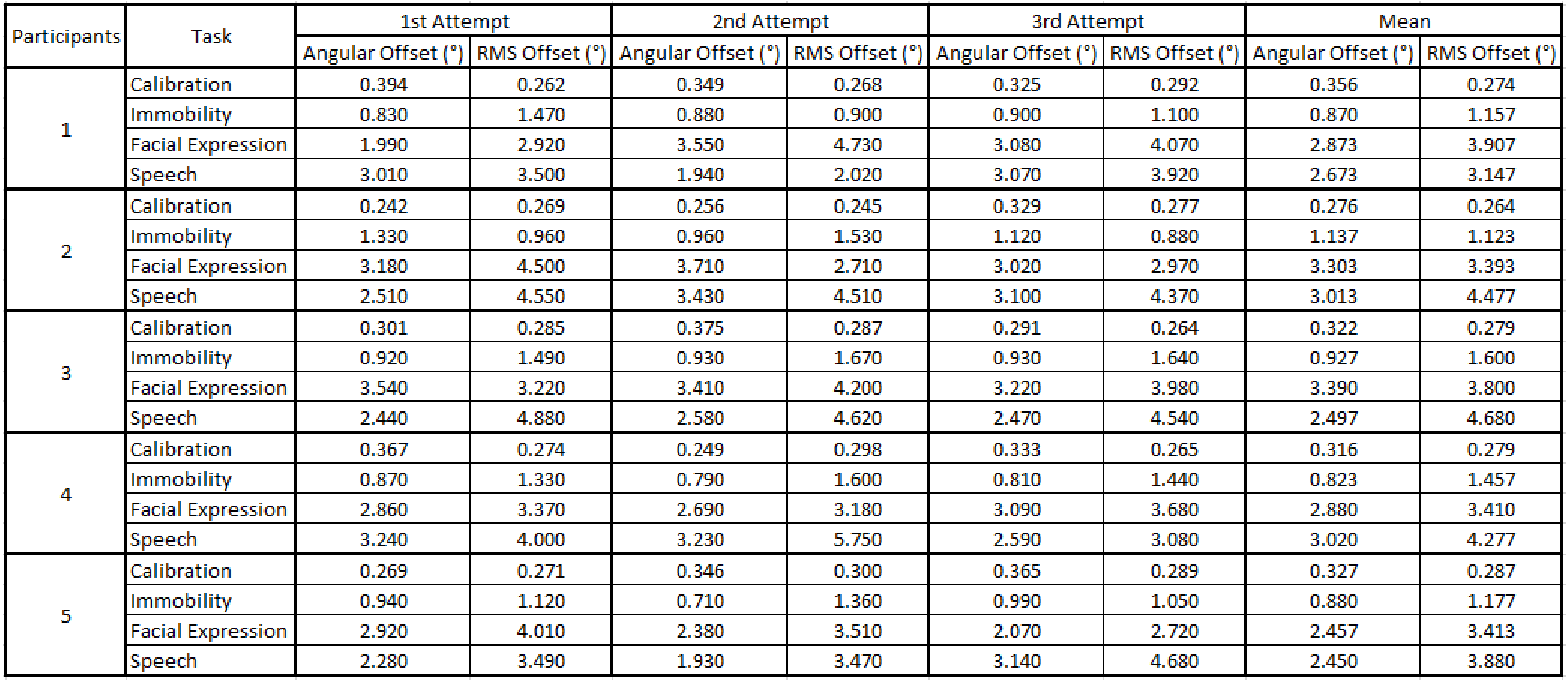
4.3 Results

Data from five participants, each performing three attempts for calibration, immobility, facial expression, and speech tasks, totalling 60 data points, are collected.

4.3.1 Tabulation of Data

The angular accuracy and precision, as determined by the equations in the previous subsection, are displayed to the user on the UI. The acquired data is subsequently recorded and tabulated in a Microsoft Excel spreadsheet. Table 4.1 displays the data presented in a tabular format using Microsoft Excel. The data are recorded in the terms of angular offset and RMS angular offset in unit of degree (°), where the former indicates the angular accuracy while the latter represents the angular precision. The higher the offsets, the lower the angular accuracy and precision. In another words, the closer the offset to 0°, the better the performance of the eye tracker.

Table 4.1: Data Collected from 5 Participants during Calibration, Immobility, Facial Expression and Speech Conditions



Participants	Task	1st Attempt		2nd Attempt		3rd Attempt		Mean	
		Angular Offset (°)	RMS Offset (°)	Angular Offset (°)	RMS Offset (°)	Angular Offset (°)	RMS Offset (°)	Angular Offset (°)	RMS Offset (°)
1	Calibration	0.394	0.262	0.349	0.268	0.325	0.292	0.356	0.274
	Immobility	0.830	1.470	0.880	0.900	0.900	1.100	0.870	1.157
	Facial Expression	1.990	2.920	3.550	4.730	3.080	4.070	2.873	3.907
	Speech	3.010	3.500	1.940	2.020	3.070	3.920	2.673	3.147
2	Calibration	0.242	0.269	0.256	0.245	0.329	0.277	0.276	0.264
	Immobility	1.330	0.960	0.960	1.530	1.120	0.880	1.137	1.123
	Facial Expression	3.180	4.500	3.710	2.710	3.020	2.970	3.303	3.393
	Speech	2.510	4.550	3.430	4.510	3.100	4.370	3.013	4.477
3	Calibration	0.301	0.285	0.375	0.287	0.291	0.264	0.322	0.279
	Immobility	0.920	1.490	0.930	1.670	0.930	1.640	0.927	1.600
	Facial Expression	3.540	3.220	3.410	4.200	3.220	3.980	3.390	3.800
	Speech	2.440	4.880	2.580	4.620	2.470	4.540	2.497	4.680
4	Calibration	0.367	0.274	0.249	0.298	0.333	0.265	0.316	0.279
	Immobility	0.870	1.330	0.790	1.600	0.810	1.440	0.823	1.457
	Facial Expression	2.860	3.370	2.690	3.180	3.090	3.680	2.880	3.410
	Speech	3.240	4.000	3.230	5.750	2.590	3.080	3.020	4.277
5	Calibration	0.269	0.271	0.346	0.300	0.365	0.289	0.327	0.287
	Immobility	0.940	1.120	0.710	1.360	0.990	1.050	0.880	1.177
	Facial Expression	2.920	4.010	2.380	3.510	2.070	2.720	2.457	3.413
	Speech	2.280	3.490	1.930	3.470	3.140	4.680	2.450	3.880

4.3.2 Data Visualization

The data presented in Table 4.1 is subsequently transformed into a boxplot diagram. Figure 4.6 and Figure 4.7 display a graph contrasting between static and dynamic models in terms of angular accuracy and precision, respectively. The term "static model" pertains to the calibration and immobility aspect, whereas the term "dynamic model" pertains to tasks related to facial expression and speech. A separate graph, depicted in Figure 4.12 shows a graphical presentation of the average angular accuracy and precision obtained across 5 participants for each activity.

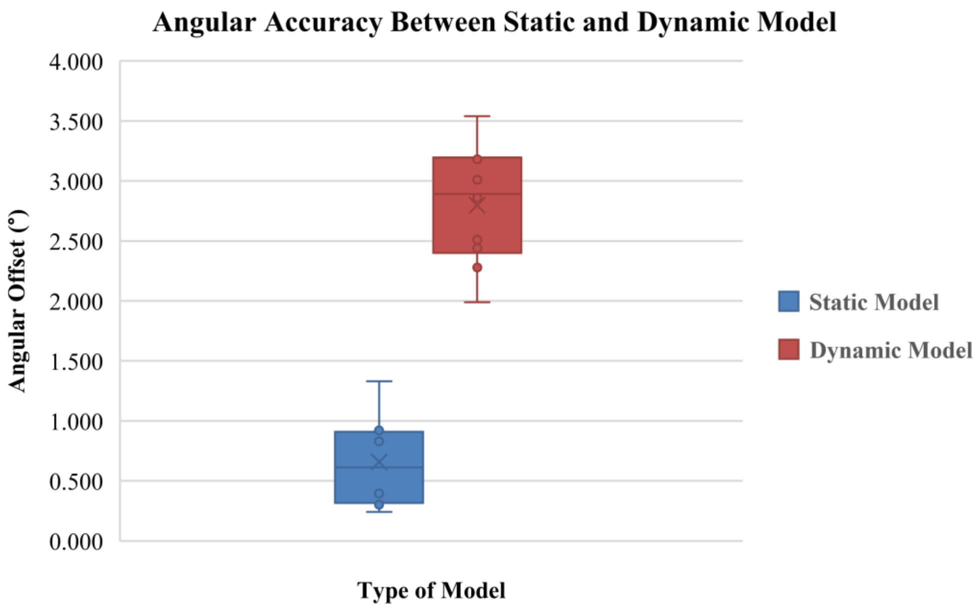


Figure 4.6: Angular Accuracy between Static and Dynamic Model

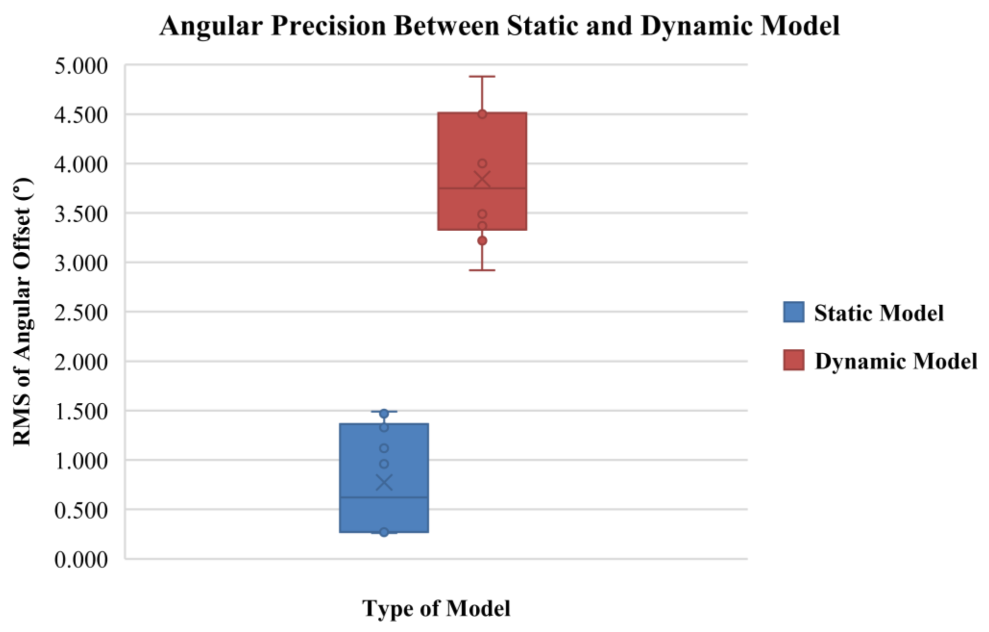


Figure 4.7: Angular Precision between Static and Dynamic Model

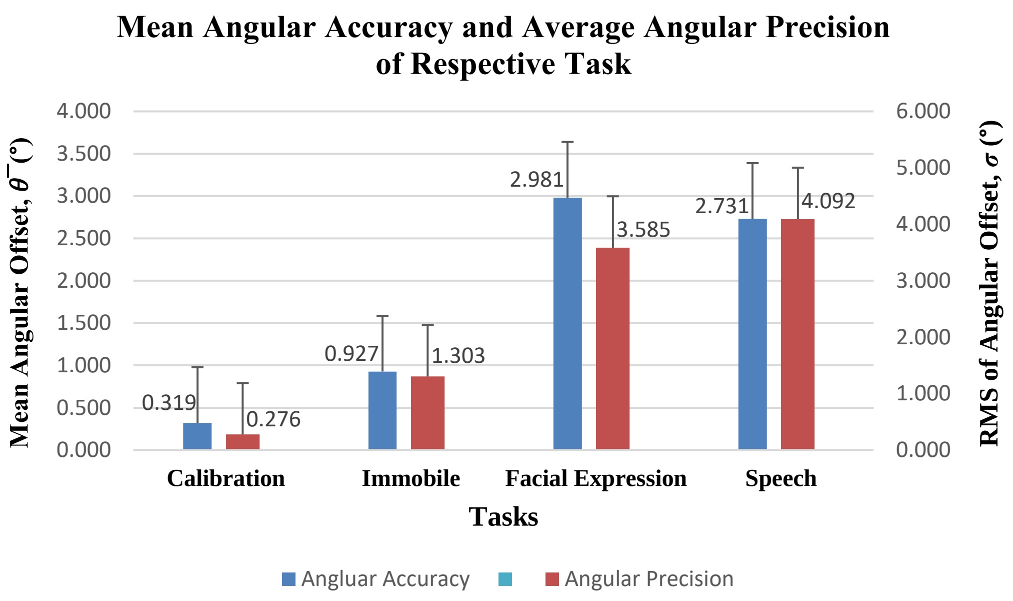


Figure 4.8: Mean Angular Accuracy and Angular Precision for Each Task

4.4 Discussion

The tabulated statistics and data visualizations in Figures 4.6 and Figure 4.7 clearly demonstrate that static tasks result in greater angular accuracy and precision than dynamic jobs. The difference occurs because there are very little movements of the head and face when performing still tasks. This helps keep the eye tracker stable and reduces the chances of it being misaligned.

Nevertheless, there are still outliers that can be mostly attributed to extreme eye strain, which results in user fatigue or reduced concentration. Moreover, variations in the ambient lighting can cause the user's pupils to rapidly dilate or constrict, thereby intensifying any anomalies.

Tasks that include facial expressions and speech, which are constantly changing, show larger variations, leading to a noticeable misalignment of the mouse pointer. The eye tracker faces difficulties in accurately tracking movements such as chin elevation, eyebrow gestures, or nose gestures, resulting in the pointer slipping. Furthermore, occurrences of eye blinking caused by eye discomfort can lead to cursor misclicks, especially when they are far away from the intended targets.

Figure 4.12 illustrates the average angular accuracy and precision for each activity. Calibration activities often produce accurate and exact results in comparison to immobile tasks as the offset is lesser. The slight inaccuracies in the latter may be caused by a delay in the cursor control plugin. Predictions of stabilized gaze positions are generated by averaging many data, which can result in minor mistakes caused by natural eye movement. However, static tasks still show significant angular accuracy and precision when compared to dynamic tasks. Among these dynamic tasks, speech tests have slightly higher accuracy than facial expression tasks. This might be due to the movements of nose and eyes instead of only movement of mouth during speech.

In terms of the average angular accuracy and precision in immobility tasks, this prototype demonstrates satisfactory performance when compared to the current eye trackers. In the study conducted by Macinnes *et al.* (2018) as highlighted in Chapter 2, the angular accuracy and precision of three widely used versions of wearable eye-trackers were evaluated.

The average total angular accuracy and precision are shown in Table 4.2. Despite variations in the experimental setup, including different gaze angles and distances, the results remain comparable to those of the prototype. This is because both experiments involve a gaze angle of 0°, where the target is directly in front of the participant, and the participant remains stationary. The prototype's angular accuracy and precision are evaluated by comparing them to the Pupil Core built by Pupil Labs. The Pupil Labs 120 Hz Binocular produces an average angular offset of 0.84° and an average root mean square (RMS) angular offset of 0.16°, as depicted in Figure 4.14. The prototype's immobile task results in an angular offset of 0.93° and a root mean square (RMS) of angular offset of 1.30°. Angular offset of 0.93° is acceptable as it falls between the offsets of existing eye trackers. For the RMS of angular offset, it has a slight difference of offsets (RMS) between the prototype the existing eye trackers, where existing eye trackers having offsets (RMS) less than 0.50° but the prototype has the offset of more than 1.00°. In fact, the prototype is not as precise as the existing eye tracker, but when it is converted back to pixels, the offset is around 58 pixels, which is around 1.51 cm offset. It can be inferred that the prototype exhibits higher accuracy but intermediate precision. In terms of cost-effectiveness, the construction of the prototype amounts to a total of RM 135.05, as tabulated in Table 4.3, whereas the Pupil Core has a price tag of approximately RM 10,966.60 (€ 2150). Due to the significant disparity in price but minimal variation in angular accuracy and precision, the prototype is considered to be cost-effective.

Table 4.2: Comparison of Overall Accuracy and Precision across Existing Eye Trackers Model and Prototype

Eye-Tracker Model	**Mean Overall Accuracy**	**Mean Overall Precision**
Pupil Labs	0.84°	0.16°
SMI	1.21°	0.19°
Tobii	1.42°	0.34°
Prototype	0.93°	1.30°

Table 4.3: Costing of Prototype

Component	Cost (RM)
Logitech C525	80.00
A03 PC1082	32.30
3D Printing Materials	10.00
Face Shield Frame	6.90
Anti-slip Nose Pads	2.70
IR LED	0.15
Screw	3.00
Total	135.05

Experiment failures can happen when users unintentionally move their heads without warning, usually because the Pupil Capture world camera window is blocked by full-screen user interfaces. This situation may lead to *AprilTags* becoming out of the field of view (FOV) of the world camera, or users may unintentionally move beyond the estimated screen boundary, causing in misalignment of their gaze. As a result, the mouse cursor remains stationary on the screen for long periods due to data loss.

In short, the findings suggest that the accuracy and precision of cursor control are at their best when users stay still under controlled circumstances. Hence, this cursor manipulation programme, in conjunction with a virtual keyboard, can assist individuals with motor impairments in expressing their requirements by utilising a gaze-controlled mouse cursor to input text or vocalise their thoughts.

4.5 Summary

This section focuses on the project's results, namely the data that is gathered. An analysis of data and the use of graphic representations provide a more lucid depiction of the outcomes. These findings have the potential to contribute to future development and improvements. By comparing the new results to the current results, insights can be gained on the extent of advances in terms of accuracy and precision.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

Concisely, a cost-efficient, sturdy, and precise eye tracker that may be worn on the head has been created and produced. The eye tracker can be integrated with cursor control plugin API to function as a gazed-controlled mouse cursor, allowing for typing using the graphical user interface (GUI) of the virtual keyboard. The created eye tracker provides notable benefits in terms of cost and efficacy when compared to commercial options like Tobii or Apple Vision Pro. With respect to precision, the eye tracker has outstanding efficacy, especially in stationary circumstances.

Regarding robustness, although the eye tracker's performance is considered satisfactory, there is potential for enhancement in terms of stability, shock resistance, and endurance. Nevertheless, the main goal has been accomplished, and the objectives have been effectively fulfilled.

5.2 Recommendation for Future Work

When developing hardware, it is advisable to use a covered design for the eye tracker, resembling VR glasses, rather than a simple frame. Although there is a possibility of an overall weight gain, this method has the potential to improve stability and reduce slippage. Furthermore, opting for camera modules with enhanced specs has the potential to enhance the performance of the eye tracker.

In addition to hardware difficulties, the utilization of *AprilTags* may present constraints for the cursor control application due to its susceptibility to misalignment and user movement. Moreover, the visibility of *AprilTags* to the world camera can be influenced by lighting conditions. In this situation, when the world camera identifies the inactive screen using *AprilTags*, only a small amount of user movement is acceptable. To resolve this problem, one proposal is to employ devices that actively monitor and record the user's motions and track the user’s eye instead.

.

REFERENCES

Alhamad, R.A., Karrar, M., Zaher, Q., Al-Issa, A. and Bonny, T., 2022. Visual Typer for the Handicapped. *2022 Advances in Science and Engineering Technology International Conferences, ASET 2022*. https://doi.org/10.1109/ASET53988.2022.9735081.

Chan, H., 2021. Development of a wearable eye tracker to control an autonomous wheelchair.

Collaguazo, H., Córdova, P. and Gordón, C., 2018. Communication and Daily Activities Assistant System for Patient with Amyotrophic Lateral Sclerosis. In: *2018 International Conference on eDemocracy & eGovernment (ICEDEG)*. pp.218–222. https://doi.org/10.1109/ICEDEG.2018.8372373.

Farnsworth, B., 2023. *What is Eye Tracking and How Does it Work? - iMotions*. [online] Available at: <https://imotions.com/blog/learning/best-practice/eye-tracking-work/> [Accessed 17 September 2023].

George, A., 2019. *(PDF) Image based Eye Gaze Tracking and its Applications*. [online] Available at: <https://www.researchgate.net/publication/334388474_Image_based_Eye_Gaze_Tracking_and_its_Applications> [Accessed 17 September 2023].

Glenstrup, A.J. and Engell-Nielsen, T., 1995. Eye Controlled Media: Present and Future State. [online] Available at: <https://api.semanticscholar.org/CorpusID:53846618>.

Hausamann, P., Sinnott, C. and MacNeilage, P.R., 2020. Positional head-eye tracking outside the lab: An open-source solution. *Eye Tracking Research and Applications Symposium (ETRA)*. [online] https://doi.org/10.1145/3379156.3391365.

Klaib, A.F., Alsrehin, N.O., Melhem, W.Y., Bashtawi, H.O. and Magableh, A.A., 2021. Eye tracking algorithms, techniques, tools, and applications with an emphasis on machine learning and Internet of Things technologies. *Expert Systems with Applications*, 166, p.114037. https://doi.org/10.1016/J.ESWA.2020.114037.

Mazhar, O., Shah, T.A., Khan, M.A. and Tehami, S., 2015. A real-time webcam based Eye Ball Tracking System using MATLAB. *2015 IEEE 21st*

International Symposium for Design and Technology in Electronic Packaging, SIITME 2015, pp.139–142. https://doi.org/10.1109/SIITME.2015.7342312.

Niehorster, D.C., Santini, T., Hessels, R.S., Hooge, I.T.C., Kasneci, E. and Nyström, M., 2020. The impact of slippage on the data quality of head-worn eye trackers. *Behavior Research Methods*, [online] 52(3), p.1140. https://doi.org/10.3758/S13428-019-01307-0.

Sabab, S.A., Kabir, M.R., Hussain, S.R., Mahmud, H., Rubaiyeat, H.A. and Hasan, M.K., 2022. VIS-iTrack: Visual Intention Through Gaze Tracking Using Low-Cost Webcam. *IEEE Access*, 10, pp.70779–70792. https://doi.org/10.1109/ACCESS.2022.3187969.

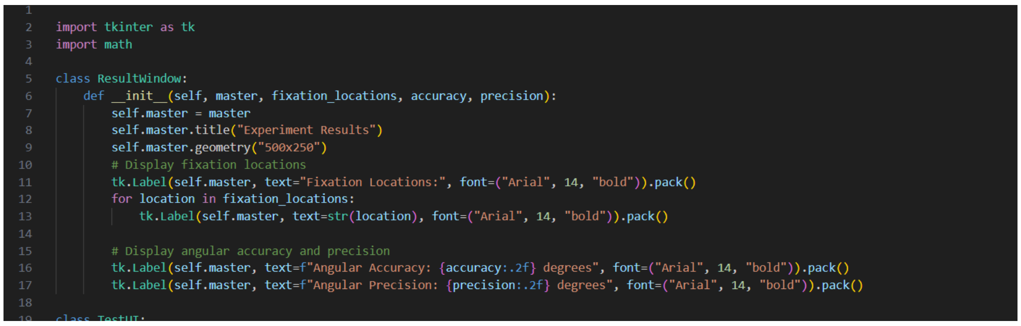
Sibert, L.E. and Jacob, R.J.K., 2000. Evaluation of eye gaze interaction. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. [online] Available at: <https://api.semanticscholar.org/CorpusID:242970>.

Szarlej, P., Carayon, I., Gnatowski, P., Glinka, M., Mroczyńska, M., Brillowska-Dabrowską, A. and Kucińska-Lipka, J., 2021. Composite polyurethane-polylactide (Pur/pla) flexible filaments for 3d fused filament fabrication (fff) of antibacterial wound dressings for skin regeneration. *Materials*, [online] 14(20), p.6054. https://doi.org/10.3390/MA14206054/S1.

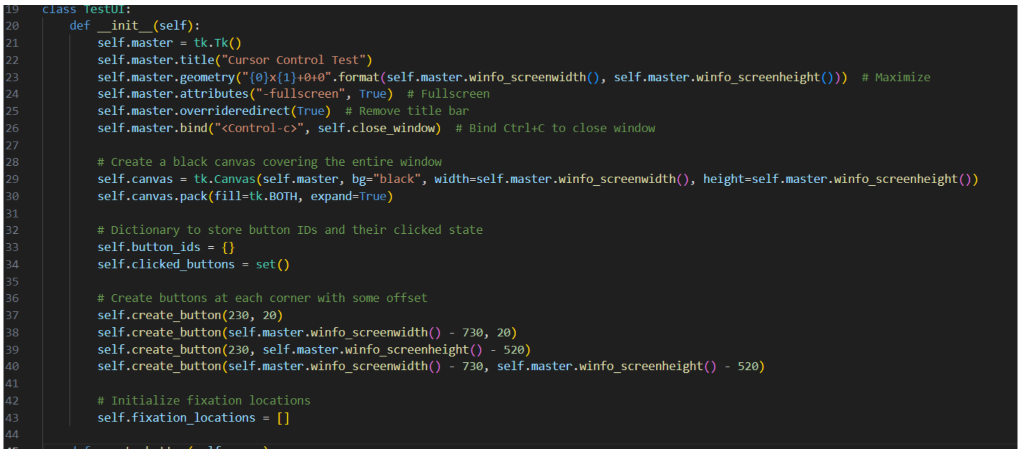
Thankachan, B., 2018. *(PDF) Haptic Feedback to Gaze Events*. [online] https://doi.org/http://dx.doi.org/10.13140/RG.2.2.28643.50729.

APPENDICES

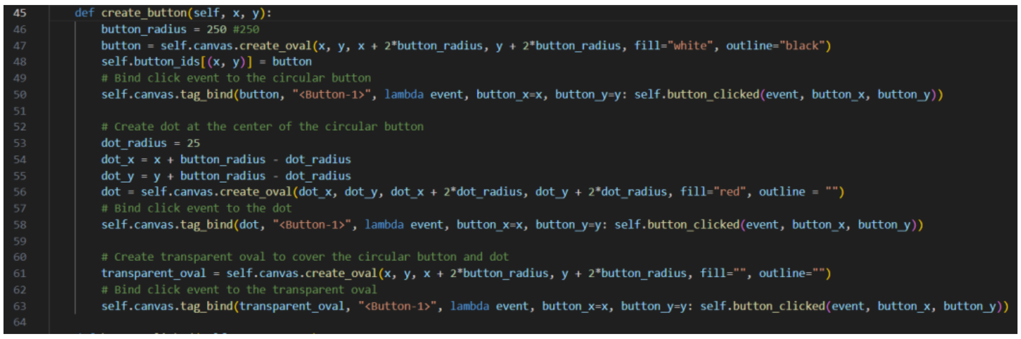
Appendix A: Python Code of Dart Boards User Interface



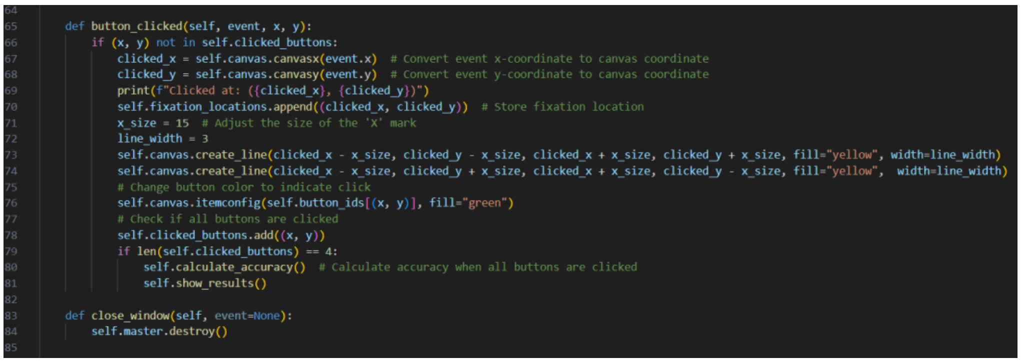
```python

import tkinter as tk
import math

class ResultWindow:
    def __init__(self, master, fixation_locations, accuracy, precision):
        self.master = master
        self.master.title("Experiment Results")
        self.master.geometry("500x250")
        # Display fixation locations
        tk.Label(self.master, text="Fixation Locations:", font=("Arial", 14, "bold")).pack()
        for location in fixation_locations:
            tk.Label(self.master, text=str(location), font=("Arial", 14, "bold")).pack()

        # Display angular accuracy and precision
        tk.Label(self.master, text=f"Angular Accuracy: {accuracy:.2f} degrees", font=("Arial", 14, "bold")).pack()
        tk.Label(self.master, text=f"Angular Precision: {precision:.2f} degrees", font=("Arial", 14, "bold")).pack()

```



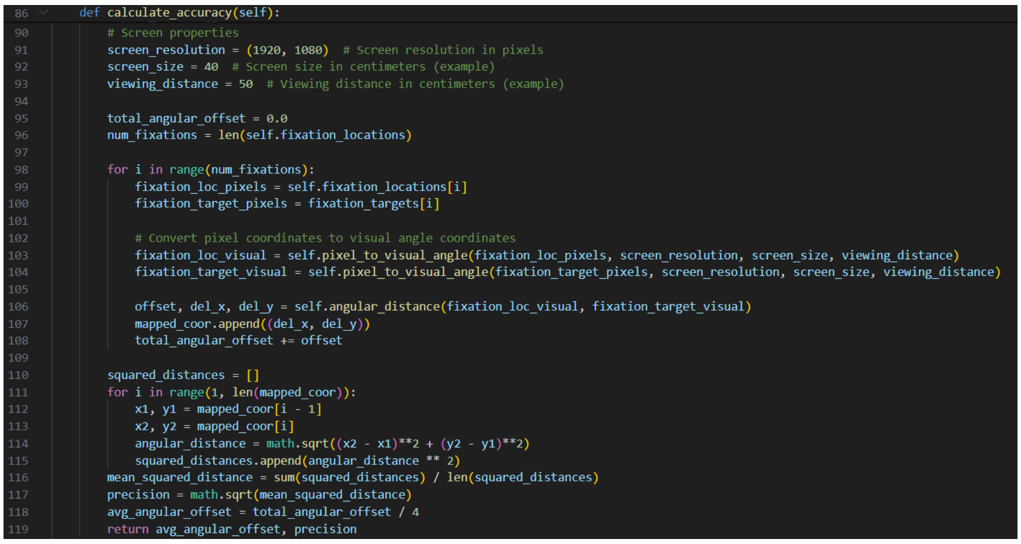
```python
class TestUI:
    def __init__(self):
        self.master = tk.Tk()
        self.master.title("Cursor Control Test")
        self.master.geometry("{0}x{1}+0+0".format(self.master.winfo_screenwidth(), self.master.winfo_screenheight()))  # Maximize
        self.master.attributes("-fullscreen", True)  # Fullscreen
        self.master.overrideredirect(True)  # Remove title bar
        self.master.bind("<Control-c>", self.close_window)  # Bind Ctrl+C to close window

        # Create a black canvas covering the entire window
        self.canvas = tk.Canvas(self.master, bg="black", width=self.master.winfo_screenwidth(), height=self.master.winfo_screenheight())
        self.canvas.pack(fill=tk.BOTH, expand=True)

        # Dictionary to store button IDs and their clicked state
        self.button_ids = {}
        self.clicked_buttons = set()

        # Create buttons at each corner with some offset
        self.create_button(230, 20)
        self.create_button(self.master.winfo_screenwidth() - 730, 20)
        self.create_button(230, self.master.winfo_screenheight() - 520)
        self.create_button(self.master.winfo_screenwidth() - 730, self.master.winfo_screenheight() - 520)

        # Initialize fixation locations
        self.fixation_locations = []

```



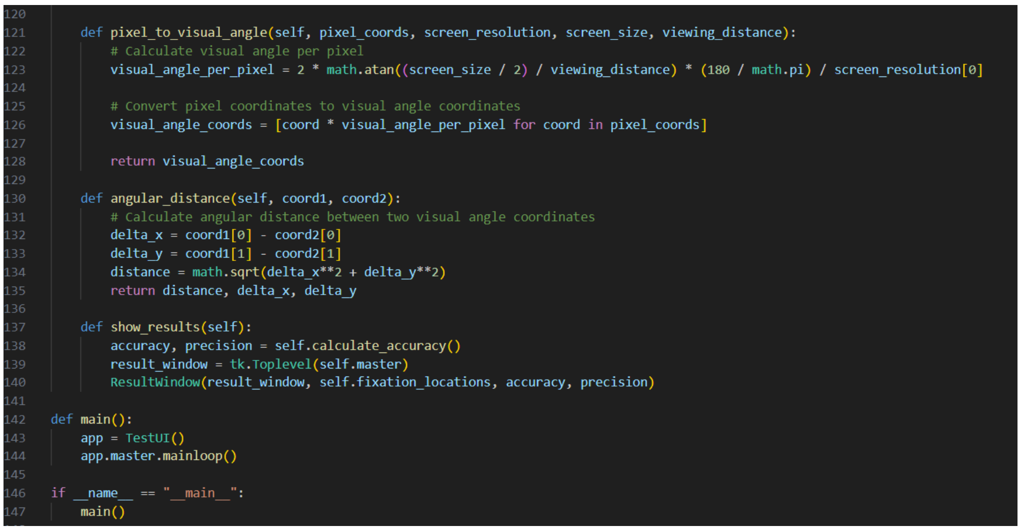
```python
    def create_button(self, x, y):
        button_radius = 250 #250
        button = self.canvas.create_oval(x, y, x + 2*button_radius, y + 2*button_radius, fill="white", outline="black")
        self.button_ids[(x, y)] = button
        # Bind click event to the circular button
        self.canvas.tag_bind(button, "<Button-1>", lambda event, button_x=x, button_y=y: self.button_clicked(event, button_x, button_y))

        # Create dot at the center of the circular button
        dot_radius = 25
        dot_x = x + button_radius - dot_radius
        dot_y = y + button_radius - dot_radius
        dot = self.canvas.create_oval(dot_x, dot_y, dot_x + 2*dot_radius, dot_y + 2*dot_radius, fill="red", outline = "")
        # Bind click event to the dot
        self.canvas.tag_bind(dot, "<Button-1>", lambda event, button_x=x, button_y=y: self.button_clicked(event, button_x, button_y))

        # Create transparent oval to cover the circular button and dot
        transparent_oval = self.canvas.create_oval(x, y, x + 2*button_radius, y + 2*button_radius, fill="", outline="")
        # Bind click event to the transparent oval
        self.canvas.tag_bind(transparent_oval, "<Button-1>", lambda event, button_x=x, button_y=y: self.button_clicked(event, button_x, button_y))

```



```python

    def button_clicked(self, event, x, y):
        if (x, y) not in self.clicked_buttons:
            clicked_x = self.canvas.canvasx(event.x)  # Convert event x-coordinate to canvas coordinate
            clicked_y = self.canvas.canvasy(event.y)  # Convert event y-coordinate to canvas coordinate
            print(f"Clicked at: ({clicked_x}, {clicked_y})")
            self.fixation_locations.append((clicked_x, clicked_y))  # Store fixation location
            x_size = 15  # Adjust the size of the 'X' mark
            line_width = 3
            self.canvas.create_line(clicked_x - x_size, clicked_y - x_size, clicked_x + x_size, clicked_y + x_size, fill="yellow", width=line_width)
            self.canvas.create_line(clicked_x - x_size, clicked_y + x_size, clicked_x + x_size, clicked_y - x_size, fill="yellow",  width=line_width)
            # Change button color to indicate click
            self.canvas.itemconfig(self.button_ids[(x, y)], fill="green")
            # Check if all buttons are clicked
            self.clicked_buttons.add((x, y))
            if len(self.clicked_buttons) == 4:
                self.calculate_accuracy()  # Calculate accuracy when all buttons are clicked
                self.show_results()

    def close_window(self, event=None):
        self.master.destroy()

```

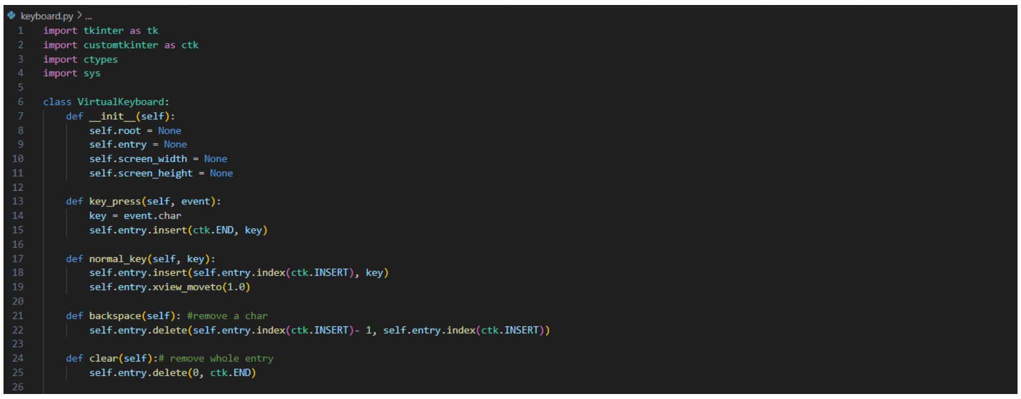


```python
    def calculate_accuracy(self):
        # Screen properties
        screen_resolution = (1920, 1080)  # Screen resolution in pixels
        screen_size = 40  # Screen size in centimeters (example)
        viewing_distance = 50  # Viewing distance in centimeters (example)

        total_angular_offset = 0.0
        num_fixations = len(self.fixation_locations)

        for i in range(num_fixations):
            fixation_loc_pixels = self.fixation_locations[i]
            fixation_target_pixels = fixation_targets[i]

            # Convert pixel coordinates to visual angle coordinates
            fixation_loc_visual = self.pixel_to_visual_angle(fixation_loc_pixels, screen_resolution, screen_size, viewing_distance)
            fixation_target_visual = self.pixel_to_visual_angle(fixation_target_pixels, screen_resolution, screen_size, viewing_distance)

            offset, del_x, del_y = self.angular_distance(fixation_loc_visual, fixation_target_visual)
            mapped_coor.append((del_x, del_y))
            total_angular_offset += offset

        squared_distances = []
        for i in range(1, len(mapped_coor)):
            x1, y1 = mapped_coor[i - 1]
            x2, y2 = mapped_coor[i]
            angular_distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
            squared_distances.append(angular_distance ** 2)
        mean_squared_distance = sum(squared_distances) / len(squared_distances)
        precision = math.sqrt(mean_squared_distance)
        avg_angular_offset = total_angular_offset / 4
        return avg_angular_offset, precision
```



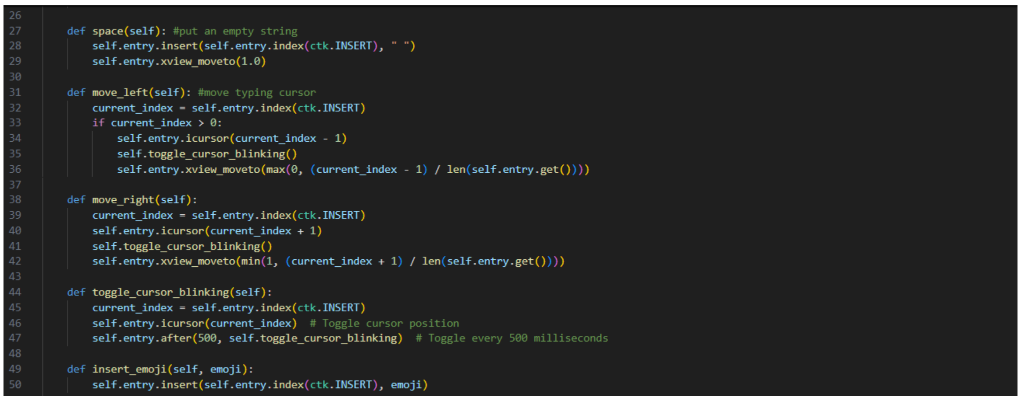
```python

    def pixel_to_visual_angle(self, pixel_coords, screen_resolution, screen_size, viewing_distance):
        # Calculate visual angle per pixel
        visual_angle_per_pixel = 2 * math.atan((screen_size / 2) / viewing_distance) * (180 / math.pi) / screen_resolution[0]

        # Convert pixel coordinates to visual angle coordinates
        visual_angle_coords = [coord * visual_angle_per_pixel for coord in pixel_coords]

        return visual_angle_coords

    def angular_distance(self, coord1, coord2):
        # Calculate angular distance between two visual angle coordinates
        delta_x = coord1[0] - coord2[0]
        delta_y = coord1[1] - coord2[1]
        distance = math.sqrt(delta_x**2 + delta_y**2)
        return distance, delta_x, delta_y

    def show_results(self):
        accuracy, precision = self.calculate_accuracy()
        result_window = tk.Toplevel(self.master)
        ResultWindow(result_window, self.fixation_locations, accuracy, precision)

def main():
    app = TestUI()
    app.master.mainloop()

if __name__ == "__main__":
    main()
```

Appendix B: Python Code of Virtual Keyboard GUI

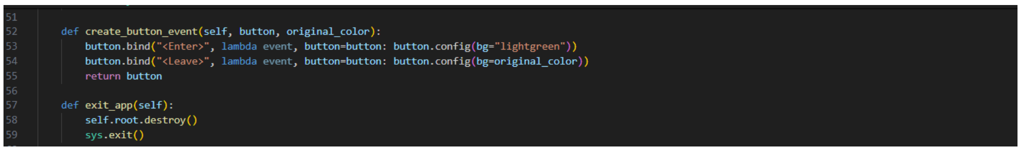


keyboard.py > ...

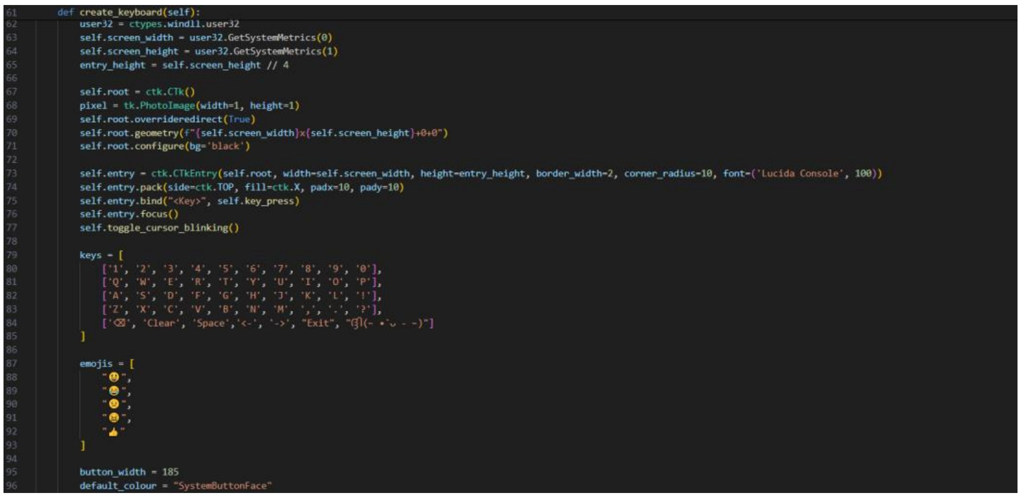
```python
import tkinter as tk
import customtkinter as ctk
import ctypes
import sys

class VirtualKeyboard:
    def __init__(self):
        self.root = None
        self.entry = None
        self.screen_width = None
        self.screen_height = None

    def key_press(self, event):
        key = event.char
        self.entry.insert(ctk.END, key)

    def normal_key(self, key):
        self.entry.insert(self.entry.index(ctk.INSERT), key)
        self.entry.xview_moveto(1.0)

    def backspace(self): #remove a char
        self.entry.delete(self.entry.index(ctk.INSERT)- 1, self.entry.index(ctk.INSERT))

    def clear(self):# remove whole entry
        self.entry.delete(0, ctk.END)

```



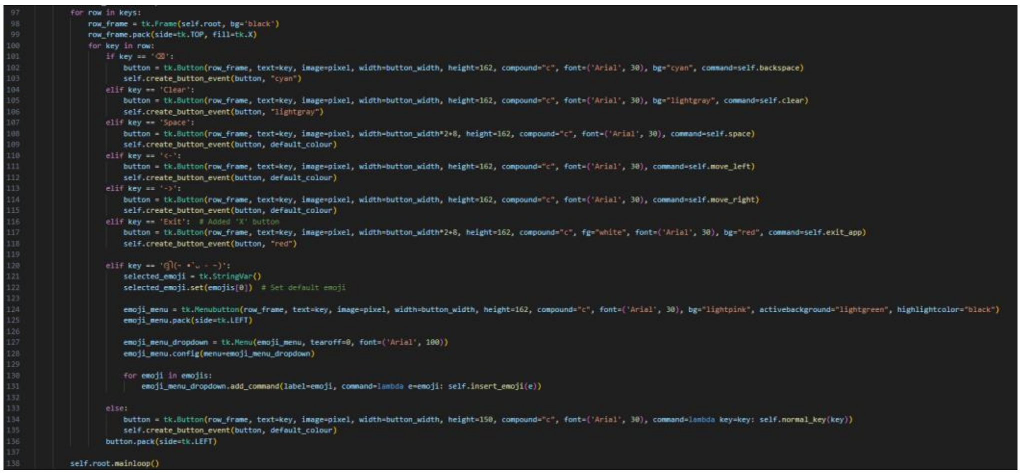
```python

    def space(self): #put an empty string
        self.entry.insert(self.entry.index(ctk.INSERT), " ")
        self.entry.xview_moveto(1.0)

    def move_left(self): #move typing cursor
        current_index = self.entry.index(ctk.INSERT)
        if current_index > 0:
            self.entry.icursor(current_index - 1)
            self.toggle_cursor_blinking()
            self.entry.xview_moveto(max(0, (current_index - 1) / len(self.entry.get())))

    def move_right(self):
        current_index = self.entry.index(ctk.INSERT)
        self.entry.icursor(current_index + 1)
        self.toggle_cursor_blinking()
        self.entry.xview_moveto(min(1, (current_index + 1) / len(self.entry.get())))

    def toggle_cursor_blinking(self):
        current_index = self.entry.index(ctk.INSERT)
        self.entry.icursor(current_index)  # Toggle cursor position
        self.entry.after(500, self.toggle_cursor_blinking)  # Toggle every 500 milliseconds

    def insert_emoji(self, emoji):
        self.entry.insert(self.entry.index(ctk.INSERT), emoji)
```



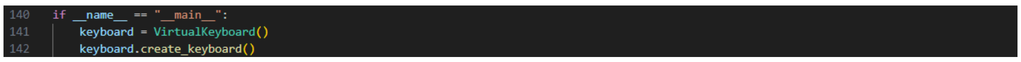
```python

    def create_button_event(self, button, original_color):
        button.bind("<Enter>", lambda event, button=button: button.config(bg="lightgreen"))
        button.bind("<Leave>", lambda event, button=button: button.config(bg=original_color))
        return button

    def exit_app(self):
        self.root.destroy()
        sys.exit()
```



```python
    def create_keyboard(self):
        user32 = ctypes.windll.user32
        self.screen_width = user32.GetSystemMetrics(0)
        self.screen_height = user32.GetSystemMetrics(1)
        entry_height = self.screen_height // 4

        self.root = ctk.CTk()
        pixel = tk.PhotoImage(width=1, height=1)
        self.root.overrideredirect(True)
        self.root.geometry(f"{self.screen_width}x{self.screen_height}+0+0")
        self.root.configure(bg='black')

        self.entry = ctk.CTkEntry(self.root, width=self.screen_width, height=entry_height, border_width=2, corner_radius=10, font=('Lucida Console', 100))
        self.entry.pack(side=ctk.TOP, fill=ctk.X, padx=10, pady=10)
        self.entry.bind("<Key>", self.key_press)
        self.entry.focus()
        self.toggle_cursor_blinking()

        keys = [
            ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0'],
            ['Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P'],
            ['A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', '!'],
            ['Z', 'X', 'C', 'V', 'B', 'N', 'M', ',', '.', '?'],
            ['⌫', 'Clear', 'Space','<-', '->', "Exit", "[illegible](- •ᴗ - -)"]
        ]

        emojis = [
            "😃",
            "😂",
            "😟",
            "😫",
            "👍"
        ]

        button_width = 185
        default_colour = "SystemButtonFace"
```



```
        for row in keys:
            row_frame = tk.Frame(self.root, bg='black')
            row_frame.pack(side=tk.TOP, fill=tk.X)
            for key in row:
                if key == '⌫':
                    button = tk.Button(row_frame, text=key, image=pixel, width=button_width, height=162, compound="c", font=('Arial', 30), bg="cyan", command=self.backspace)
                    self.create_button_event(button, "cyan")
                elif key == 'Clear':
                    button = tk.Button(row_frame, text=key, image=pixel, width=button_width, height=162, compound="c", font=('Arial', 30), bg="lightgray", command=self.clear)
                    self.create_button_event(button, "lightgray")
                elif key == 'Space':
                    button = tk.Button(row_frame, text=key, image=pixel, width=button_width*2+8, height=162, compound="c", font=('Arial', 30), command=self.space)
                    self.create_button_event(button, default_colour)
                elif key == '<-':
                    button = tk.Button(row_frame, text=key, image=pixel, width=button_width, height=162, compound="c", font=('Arial', 30), command=self.move_left)
                    self.create_button_event(button, default_colour)
                elif key == '->':
                    button = tk.Button(row_frame, text=key, image=pixel, width=button_width, height=162, compound="c", font=('Arial', 30), command=self.move_right)
                    self.create_button_event(button, default_colour)
                elif key == 'Exit':  # Added 'X' button
                    button = tk.Button(row_frame, text=key, image=pixel, width=button_width*2+8, height=162, compound="c", fg="white", font=('Arial', 30), bg="red", command=self.exit_app)
                    self.create_button_event(button, "red")

                elif key == 'ญ](- •'ᴗ - -)':
                    selected_emoji = tk.StringVar()
                    selected_emoji.set(emojis[0])  # Set default emoji

                    emoji_menu = tk.Menubutton(row_frame, text=key, image=pixel, width=button_width, height=162, compound="c", font=('Arial', 30), bg="lightpink", activebackground="lightgreen", highlightcolor="black")
                    emoji_menu.pack(side=tk.LEFT)

                    emoji_menu_dropdown = tk.Menu(emoji_menu, tearoff=0, font=('Arial', 100))
                    emoji_menu.config(menu=emoji_menu_dropdown)

                    for emoji in emojis:
                        emoji_menu_dropdown.add_command(label=emoji, command=lambda e=emoji: self.insert_emoji(e))

                else:
                    button = tk.Button(row_frame, text=key, image=pixel, width=button_width, height=150, compound="c", font=('Arial', 30), command=lambda key=key: self.normal_key(key))
                    self.create_button_event(button, default_colour)
                button.pack(side=tk.LEFT)

        self.root.mainloop()
```



```
if __name__ == "__main__":
    keyboard = VirtualKeyboard()
    keyboard.create_keyboard()
```