Webcam Gaze-Tracking for Video Game Input

Gabriella Farrell

Department of Informatics
School of Informatics and Engineering
Technological University Dublin - Blanchardstown Campus
requirements for the degree of
B.Sc. (Hons) in Computing Science

Supervisor: Dr. Matt Smith

05/05/2019

Acknowledgments

Credits to the supervisor of this project, Dr. Matt Smith for advice, feedback and troubleshooting throughout the entire process. Further thanks to Stephen Sheridan for initial direction advice, and examples of similar projects.

Abstract

Gaze/eye tracking is usually achieved by using Infrared spectrum cameras, however, the average HD household webcam in 2018 has the potential to be used as an alternative. This research project explores how webcam gaze tracking can be utilized as an accessible tool for video game input. The implementation is demonstrated by developing a Unity WebGL game named "Drive-On" that relies on the movement of your head/eyes with no input from your hands. This game involves the user navigating a race-track by controlling a car with their head tilts.

Three short calibrations tests are taken as a prerequisite to playing the Drive-On game. Drive-On contained two levels, with two subsequent difficulty settings. These settings changed the mechanics of the game to allow for players of all skill levels to have the ability to complete it. The performance of each player for both the calibration and Drive-On game are recorded, and analyzed in this research project.

Contents

List of Figures

1 Introduction

Eye/gaze tracking is a technology used in a broad range of industries from research and analytics to ophthalmology. This has allowed the collection of data about the patterns in attention and concentration of humans, and what exactly catches our attention on web pages and advertisements [13]. This project focused more on the utility gaze tracking can provide to disabled people in the area of computer games and entertainment. Geze tracking is split into 2 areas; conventional web cam gaze tracking, and expensive near-infrared light cameras [4].

Despite webcam based gaze tracking being less accurate, it has far more support through ongoing open-source projects being run by startup and existing computer vision companies. The objective of this project was to learn about the different gaze tracking technologies available and utilize them to develop a video game that relies solely on the gestures of your head and the movement of your eyes as a virtual "joystick".

While webcam gaze tracking is not as accurate infrared gaze tracking, it proved accurate enough to carry out the aims of this project. With the software developed, the player has full control over the X and Y axis with their head tilts, which can be implemented in any game that uses UP, DOWN, LEFT and RIGHT keypad controls. This paper explores the application of gaze tracking software in video-games, and the further potential of user-interface control for non-gaming related applications.

1.1 Main Research Questions

1.1.1 Can Accurate Gaze Tracking Be Achieved With a Conventional Webcam?

One of the technical challenges for webcam gaze tracking is not having high enough fidelity to draw a contrast between the pupil and the iris [14]. Infrared cameras benefit from active illumination in the infrared spectrum as opposed to the light spectrum. Most modern webcams are HD capable, but the trade-off is that they have a lower framerate. This can be corrected by lowering the resolution, but that means lowering the number of pixels available to get an accurate reading of the pupil.

This is why webcam eye tracking has been overtaken by infrared technologies,

but it does not mean it can still be utilized in an applications where less accuracy is needed i.e. video games. While infrared cameras would be more accurate for this research project, commercial eye tracker companies do not provide open-source code for their hardware.

The aim is to evaluate the available open-source software in detail and investigate the pros and cons of using a webcam versus an infrared camera. To understand this better, A series of tests needed to be conducted:

- The response time of gestures
- The accuracy in different environments (well-lit room, outside, dull dull lighting)
- How the code works, and if it can be improved
- The idea of combining both gaze tracking and head movements to make up for the low contrast capabilities of a webcam

1.1.2 Can This Data Be Read as An Application/Videogame Input?

This question is to be answered through research and development of a software component to link the available gaze tracking software to the Unity [10] video game engine. Unity WebGL holds a library that allows calling JavaScript functions from Unity CSharp scripts and vice-versa.

1.2 Justification/Benefits

1.2.1 Why This Research Project is Needed

This research project will be focused on gaze tracking as an alternative input. The idea of virtually steering a car or ship by leaning your head side-to-side, or destroying asteroids just by looking at them is an interesting direction for game development to take. There are countless creative ways this can be utilized. Webcams are a peripheral that everybody owns, which would make this a widely accessible platform of gaming.

1.2.2 Who Will Benefit From This Research Project

The main beneficiaries of this project will be persons with:

- limited/no ability in arm/hand movement
- Low hand-eye coordination
- Tremors or shaky hand syndrome
- Hands too small for keyboard/mouse control
- Children

Some of these demographics are limited to non-interactive forms of entertainment which has caused a recent cultural backlash, in regards to how video game companies are ignoring disabled gamers [2]. Besides creating regular entertainment-based video-games, this project may be utilized in more specific areas such as children's games for learning and development e.g. nodding head Yes/No, looking at a fruit or color as an answer to a question. Software like this can allow us to see how an individual's eyes behave while playing these games, and provide me valuable information that can be analysed to understand:

- What do people look at most while playing a game?
- What a child looks at, what keeps their attention? (colors, flashing objects?)
- Do certain types of games or game elements cause a physical reaction? (flinching)

1.3 Universal Design

The Drive-On game in this project implemented a design to enable users of varying skill levels can complete and enjoy the game. This was achieved by creating different in-game mechanics to accommodate varying skill levels. The aims of this design are as follows:

- To evaluate user skill level in the input method,
- Adapt/recommend corresponding game difficulty,
- Maximize every user's chance of success.

1.4 Report Structure

The remaining chapters of this report are as follows:

- **Acknowledgments:** This section provides credit to the individuals who advised me throughout the development of this project.
- **Introduction:** This section gives you a general overview of the area and the hypothesis in question.
- **Review of Existing Research:** Relevant outside sources used to help gain a better understanding of the research in this area, and the methodologies employed.
- **System Analysis an Design:** An in depth outline of the theory and perspectives of this software, and the encompassing area and the design of such.
- **Implementation:** An overview of how these technologies were implemented as well as some visuals.
- **Quantitative User Evaluation Method and Results:** The quantified hypotheses and results of testing the data items collected.
- **Technical Evaluation:** Testing the variables between different systems and environments.
- **Conclusions** The overall findings of the project.
- **Further Work** A list of the unfinished and desired components of the system

2 Review of Existing Research

Gaze tracking is a popular field in computer vision research that has made many computer interactions possible for disabled people [6][8]. These interactions range from capturing simple binary motions and translating them into mouse clicks to the ability to use specific body parts to control the mouse. This technology also paved the way for alternative video game inputs and innovative ways of immersing a player into the game.

My project investigates the capabilities of eye/face/head tracking using only generic USB webcams and the ways these capabilities can be used to create an accessible video game. Furthermore, Video games can be used as a tool for child learning [21]. With the capabilities of gaze tracking, Video Games can also allow for children with limited mobility to utilize these as learning tools.

2.1 Common Methodologies

The following is a summary of common methods of non-keyboard inputs to computer systems.

Binary input The theoretical idea of controlling a computer using unconventional ways has been solved using a wide variety of techniques. The simplest form of achieving this is invoking a binary switch operation to tell the mouse when to click. M.Chau et al. implemented a function like this by using an algorithm that uses a webcam to monitor blinks of a the user as a mouse click controller [6]. This algorithm is relevant to this research project as it prompts the idea of utilizing blinking simultaneously with gaze/head tracking to perform certain actions within the video game.

The way this particular implementation of binary inputting was achieved was by comparing the image of the area around the eye with template images of other people's eyes stored in an online database. The algorithm monitors the similarity of the user's eye with the template images in each frame, making a clear break in similarity score when the user blinks. While this method worked with a relatively high accuracy (80 percent success), it would be much too inefficient to run in tandem with a 3d video game due to the constant

comparison needed of each frame to a template. However, this technique can be adjusted to fit this project as the gaze tracking API [19] already outputs a correlation score that represents the position of the users pupils; where the score will temporarily decrease when the user blinks. This sudden change in score can be translated into a binary operation by setting an appropriate threshold, that when crossed, will trigger an event.

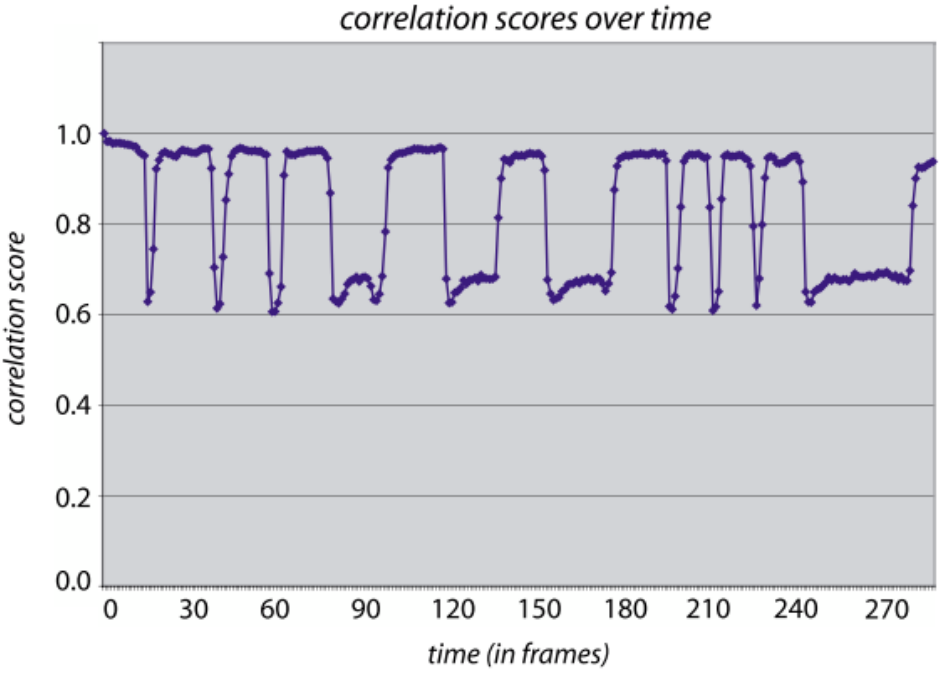


Figure 1: M.Chau et al.'s use of setting a threshold (0.8) to trigger a click event when the score representation of the tracked iris falls below 0.8.

Body Part tracking The next step from enabling users to input binary commands is to allow the user to control and move the mouse cursor using a tracked body part. P.Girraj et al. designed software that uses a USB webcam to monitor the movements of a specified body part (nose, chin, foot etc...) and mirroring those movements with the mouse cursor [8].

Webcam or Infrared Webcam gaze tracking is an outdated endeavor due to the superior technique of using infrared spectrum cameras. Infrared

cameras have the ability to monitor the patterns of light reflected from the convex lens of the eyeball.

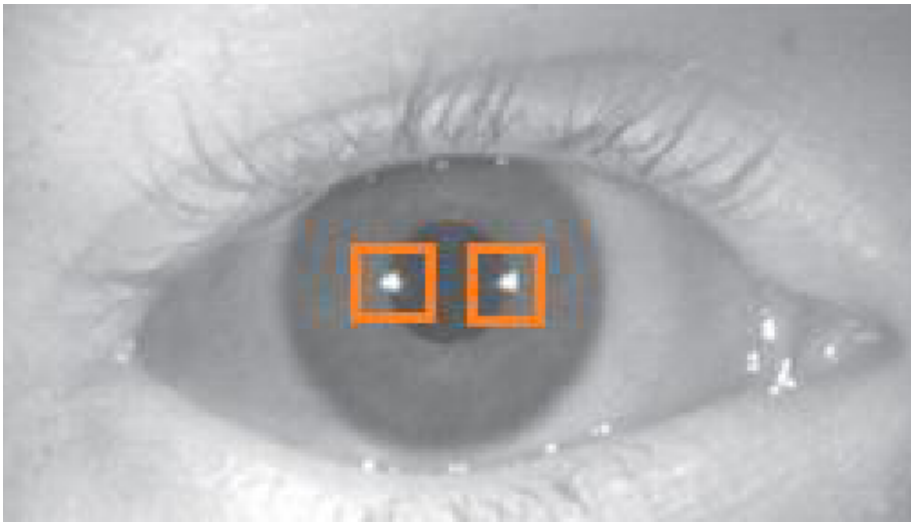


Figure 2: Corneal reflection points tracked by infrared camera

This method of gaze monitoring is much more precise than using a webcam. This however requires careful calibration of the users eyes, requiring the user to sit perfectly still. Infrared tracking also commonly requires head-mounted peripherals to amplify the accuracy. This has proven to a less desirable method of tracking with disabled users due to the discomfort and inability to remain still for calibration [8]. webcams rely on the visible light spectrum, forcing gaze software to discern the location of the eyeball by comparing the contrast of pixels around the eye area. Before comparing the pixel contrast, the software first has to locate the eye area itself. Hoffner et al. demonstrates a useful approach to this problem by placing 6 'landmarks' on the face; chin, nose, mouth corners and eye corners [11]. Once these are placed, the eye areas become easier to single-out.

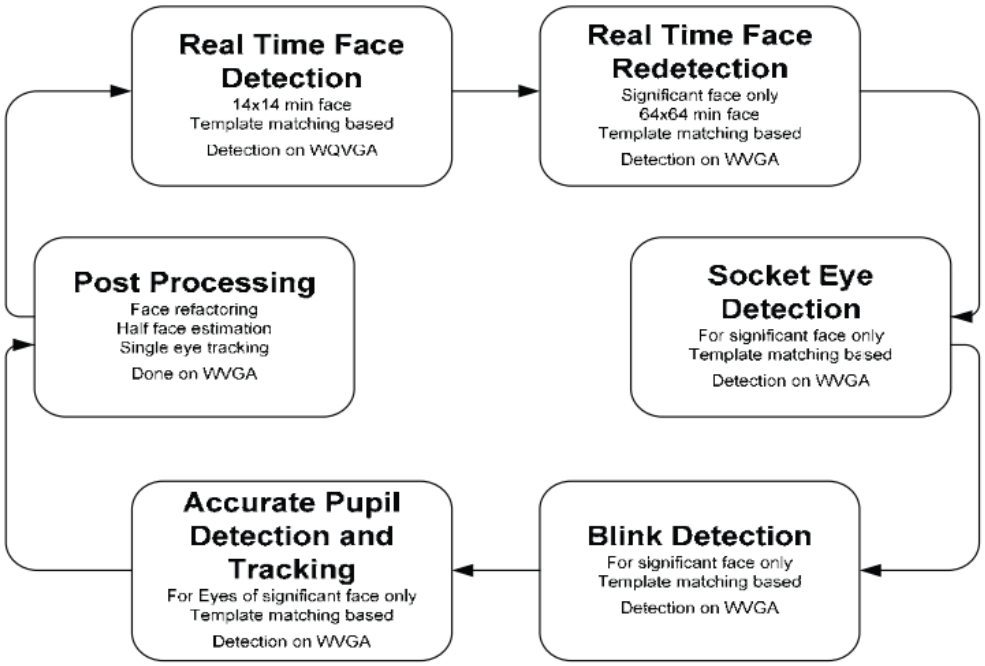


Figure 3: Webcam gaze detection algorthim flow

A common method used to improve facial feature search efficiency is for the algorithm to measure the distance between these landmarks, and ignore patterns that show un-usual measurements such as the eyes being un-naturally far apart, or mouth too far from the chin. This search algorithm is demonstrated M.Chaus blink ION software [6].

Head Tracking Head/face tracking is used by a wide variety of technologies across all platforms (camera software, novelty app overlay filters) and has proven to a an appropriate approach to gaze tracking apps that use a webcam. This is because comparing the pixel contrast of the head to the background is easier than focusing on the eyeball/iris alone. This is why face tracking can be seen in digital camera face detection as far back as early 2000s. Another reason why head tracking is a strong contender for a video game input is because head movements/nods are a universal and natural way for humans to gesture towards objects [1]. S.Wang et al. demonstrated the potential for head tracking in video games by creating a FPS (First Person

Shooter) prototype game which involves the player ducking and leaning to avoid enemy gunfire [22].

2.2 Accessibility of Gaze Tracking

Gaze tracking is a necessary tool for many people afflicted with limitations of dexterity who want to use computers. ALS (Amyotrophic lateral sclerosis [7]), MS (Multiple Sclerosis [16]) and other similar diseases restrict body movements, or incur involuntary jitters/shaking. This prevents these people from using computers normally and by extension, prevent them from enjoying video games. K.Bierre et al. studied this accessibility issue in video games and found that between 10% - 20% of people in the USA are considered "disabled" [3].

One solution to this is household implementation of gaze tracking systems. P.Girraj et al.'s method/approach of simulating mouse movement by tracking a users body part with a webcam is accommodates a broad range of disabilities as the body part used to track is designated by the user. This allows people with limited arm moments use their nose/feet instead [8]. Furthermore, in a survey conducted by A.Al-Rahayfeh & M.Faezipour, eye and head movement were shown to be the least affected by common movement-inhabiting disabilities [1]. This alone plays a major part in deciding the best methodology to tie-into a video game, as the key here is to cater to the largest demographic.

One of the struggles found by using this system is mouse click event, and how it can be invoked. One of the downsides of tracking an alternate body part than the eyes is that the user cannot blink to click anymore. most algorithms compensate for this by invoking a click event after placing the cursor over an object for a number of seconds [8]. This arises a common eye tracking problem called the "Midas Touch Problem" [12]. This problem describes an inability to hold the mouse in an idle position, because of the potential to accidentally click something while idle. In most cases this problem was combated by allowing rest areas within the program in question, to allow the user to rest the cursor there without clicking anything.

2.3 Gaze Tracking in Video Games

Since its inception, the video game industry has designed countless alternative video game peripherals. VR (Virtual Reality [20]) gaming is a good example of this as it takes the player out of the real world and immerses them in the game by tricking their eyes and subsequently, their minds to believing they are within a virtual world.

Carter et al. explored the idea of combining gaze tracking with motion tracking as a video game input [5]. The game itself was a simple idea of taking sheep and placing them into a pen, with a game over condition if the player puts a wolf in the pen instead of a sheep. This was a proof od concept demo game which used a Microsoft Kinect motion tracker [23], along side with a gaze tracking algorithm. The player looks at the object (sheep) they want to pick up, and gestures their hand to pick the object up. the object selection is driven by the gaze capturing by a webcam and the picking up is driven by the motion/gesture capturing by the Kinect. Webcams have a general visual angle of error of about 3 degrees. This means that the gaze tracker can only estimate the position of the pupils gaze with about 3cm at 1 meter, and 9cm at 2 meters [5]. to compensate for this passive inaccuracy, a nearest object first algorithm was implemented. This meant that the object selection was based on the object with the least distance from the estimated gaze position. This algorithm is relevant because it virtually lowers the margin of error common in webcam tracking, and is especially useful in an environment such as a video game.

This research is particularly interesting as it inadvertently revealed some interesting observations pertaining to the players enjoyment and/or frustration with the gaze tracking system. the majority of users felt the game to be significantly more enjoyable when they played it using their eyes and gestures. However, this enjoyment is marred when the users are shown a visualization of where the algorithm is estimating their gaze to be. This is because the inaccuracy of the webcam gaze tracker is amplified when the user focuses on the cursor itself, which creates a negative feedback loop. Users found themselves naturally trying to compensate for the inaccuracy of the cursor by overshooting their gaze to one side, or the other, depending on the location. The users who played with this visualization disabled rated the experience much higher than the players who experienced the game with the

added visual representation of the gaze tracker. The consensus that games with gaze tracking immerse the player more into the game is a common theme across these research projects. Nacke et al. experienced similar feedback of player immersion with their own demo game [17]. Gowases et al. covered the relevance of gaze tracking as a video game input and concluded that players could immerse themselves more by using this as a feature [9]. The theory behind this is that utilizing extra senses help breaks the barrier between the video game and reality.

Gaze tracking brings many possibilities to the gaming platform. Nanu et al. explored some of these ideas [18]. These possibilities can also be used to make the argument that gaze tracking in video games can be the middle ground between traditional gaming and VR gaming.

Some interesting concepts reviewed by Nanu et al. include:

- Single/double blinking as a 'fire' button
- Mouth can be opened and closed as an alternative binary input
- Close one eye to bring up weapon scope
- Potential to monitor human emotions, to enhance role-play experiences
- Altering the game at runtime based on the users physical emotional response content...

Educational Potential Children with neurological disabilities (Autism, ADHD) can quickly fall behind in learning through conventional methods. This has invoked the development of many child-learning toys and games. One problem with these devices is that the nature of these disabilities prevent the child from maintaining concentration and focus [21]. Gaze tracking can greatly benefit this area of education by providing an alternative avenue of interaction for kids who can not be captivated into playing (learning) long enough, by being a more exciting way of interacting with the game.

2.4 Conclusion

The focus of my project is to understand how gaze/head tracking can be used as a virtual controller, specifically in a video game. The aim is to offer an interesting alternative to joystick controls, while remaining simple, non-invasive and affordable. To achieve this, this project focused on using a webcam as the only input device for the game. Players can then utilize the movements of their head or eyes to play the game. Throughout research of this field, many unique and interesting approaches to this problem were designed. The results obtained from some of the reviewed research show that webcam gaze tracking is a viable option to design a game around, and that this form of interaction with games has shown to be enjoyable and immersive.

This endeavor is further supported by the inherent accessibility it provides to people with disabilities. Most forms of entertainment have alternative options to assist people with deaf, blind and colorblind people. These options can involve using text to speech, speech to text or inverting certain colors. The range of accessibility options for people with movement impairing conditions is much smaller.

3 System Analysis and Design

The two major components of this system will be the webcam and the gaze tracking algorithm. The roles of these components are as follows:

3.1 Webcam

The webcam image is used to calculate the approximate axes of the head and eyes, which is then represented and interpreted by the front end software. The ideal fidelity of the webcam should be 1920 x 1080 pixels with 30 frames per second, however, the tests garnered for this research project involved a standard laptop webcam capturing ant 1366 x 768 pixels at 30 frames per second.

3.2 Gaze Tracking Algorithm

This algorithm measures the distance between each facial landmark, as well as the distance these landmarks have moved from a previous point. This is then outputted as a series of floating point numbers between 0 and 1.

3.3 Technical Analysis

Some of the technical hurdles this project will need to overcome:

- Amount of resources needed to run smoothly,
- Minimum frame rate needed for the gaze tracking software to function consistently,
- Resolution of the WebGL game,
- Optimal web browser to run it.
- Internet speed affecting the polling rate.

3.3.1 Frame rate and Polling

The frame rate of the is directly related to the delay incurred by the head tilt data polling. A lower frame rate will cause a sluggish control delay, regardless of the polling rate from the server to the API. This was controlled by running

the game at a lower resolution and removing all unnecessary objects from the scene.

3.3.2 System Resources

This game requires two browser tabs to be active. One for the game and one for the head tracking process. Each of these applications are relatively resource intensive as both the server and the game are performing large amounts of input/output operations which are intensive on the disk and memory. The additional overhead of the in-game 3d rendering requiring performing large GPU operations can create a resource intensive application.

3.4 Non-Technical Analysis

There are many non-technical facets to this project to keep in consideration before looking into the technical elements, These include the following:

- Fidelity of webcam,
- Ideal lighting circumstances required for camera to clearly see the face,
- Ideal requirements to be met for your camera attain the information it needs (how close to sit to the camera, where to position it),
- Facial obstructions (glasses).

3.5 Design Process

The project software architecture is built around the XLabs Gaze API. This API is closed-source, but offers a multitude of useful tools. This API is split into two components; Gaze tracking and head tracking. While the gaze tracking tools were used in the research phase, the API's head tracking branch is what this software was built using.

One technical challenge was bridging the gap between the external head tracking API and the Unity game. The head tracking is a native browser extension application that is triggered by code run on a web server. A NodeJS server was written to import the the API code and run it on a local web-server. This turns on the webcam and begins polling the picture for data pertaining to the location of the persons head. This data was then stored on the local server by using a JavaScript AJAX post method, which saves the data as a JSON file at a frequency of around 40Hz. The two data items stored were "roll"(left/right tilts) and "pitch"(up/down tilts).

3.5.1 Functionality

To use this head tracking data as a joystick, A C# script is required in the Unity game development stage. This script queries the stored head tilt data json file by using the UnityWebRequest class. This class allows us to directly query the head tilt data which is being stored on the local server at 40Hz. The flow of data is annotated in Figure 4 as follows:

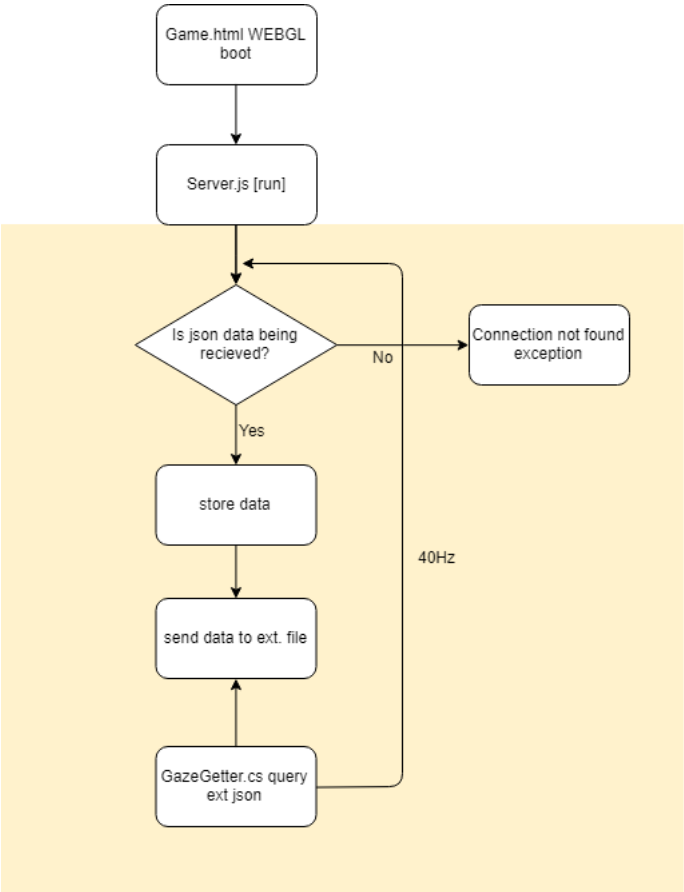


Figure 4: API data flow chart

The initial iteration of translating this data into a game controller was a discreet method. Thresholds were set for both data items to represent LEFT, RIGHT, UP and DOWN tits. (roll thresholds: 0.1 and -0.1, pitch thresholds: 0.5 and 0.7). Once beyond these thresholds, the script would

move the ball in the appropriate direction. While this method worked well for a prototype, it does not offer the accuracy and control that translating tilts of varying magnitudes would give.

Finding the optimal metrics was an iterative process that evolved naturally throughout the development of this project. Some of these were tested more in-depth which is covered in the Testing chapter.

3.5.2 Software Stack

The hierarchy of software and API's needed to combine the Gaze tracking script to the Unity application are as follows:

- Javascript gaze-tracking browser plugin,
- Apache web server,
- Node.js server and routing,
- JavaScript Ajax,
- Unity game written in CS and built as a WebGL application (to run in browser).

Due to the XLabs head tilt API being a native browser extension, building the game as a WebGL application that can be run in the same browser is the best method of deployment. This is due to the extra system resources needed to run a standalone application. The browser is required to collect the gaze data, using another browser tab is less resource intensive than a standalone application. Minimizing the delay between input and output, and maximizing the frame rate provided the smoothest experience, as delayed responses in game from head tilts proved to be disorientating and jarring.

3.5.3 Component Diagram

The hierarchy of the components in this software is annotated in Figure 5.

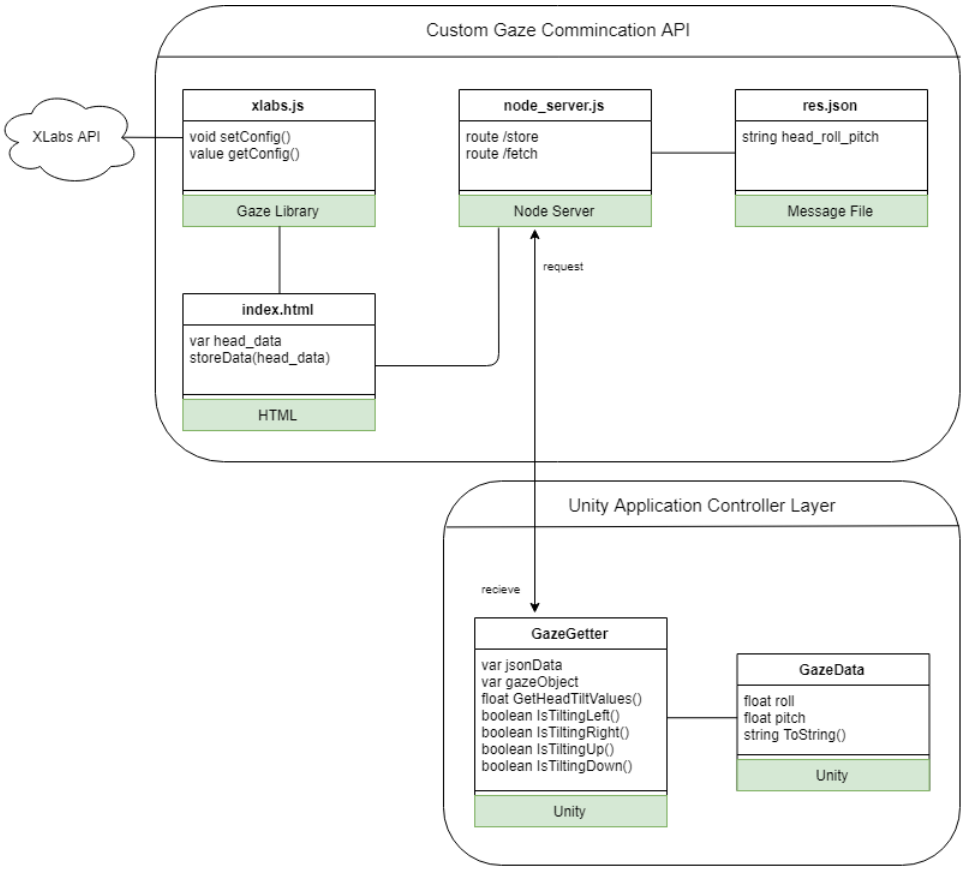


Figure 5: Component Diagram

- **xlabs.js**: communicates with the XLabs server to receive configuration data.
- **server.js**: node server that uses routing to store and fetch the gaze data being created by index.html.
- **index.html**: activates browser extension and generates gaze data. Sends data to the '/store' node route.
- **res.json**: JSON file containing gaze data.
- **GazeGetter**: C# class that HTTP requests gaze data via the '/fetch' node route. This polls at around 40Hz.

- **GazeData**: Object used to convert json data into separate usable floats.

3.5.4 Gaze Tracking Algorithm

The floating point numbers generated from this algorithm, which describe the location of the facial landmarks is stored as a JSON object. This JSON object is read and interpreted by the C# game driver program, and extracts the values of interest: roll(side to side head tilts) and pitch(up and down head tilts).

3.5.5 Game Feature Design

The Drive-On game is a timer based driving game where the objective is to finish the course in the shortest time. To generate a diverse collection of performance data, two levels were made, both with two choices of difficulty. These levels were designed to individually target both aspects of the head tilting input: horizontal and vertical. Level one has an emphasis on speed, which is controlled by vertical head tilts. Level two has an emphasis on control, which is controlled by horizontal head tilts. The varying difficulty of these levels was designed as follows:

While the objective of each level stays the same (finish as fast as possible), The mechanics vary depending on difficulty chosen.

3.5.6 Drive-On Level 1

Level 1 will be designed to have a bias towards vertical head tilting, by having easy turns with an emphasis on speed(vertical UP tilt). This is to investigate the potential correlation between Calibration 3 - Vertical and Level 1.

- Easy difficulty: The verges on the edges of the track are raise, making it so the player cannot leave the track.
- Normal difficulty: The verges are removed, making it so the player can leave the track, however, if the player leaves the track they will be reset to their most recent checkpoint.

3.5.7 Drive-On Level 2

This level, similarly to level 1, will be deigned with a bias on horizontal head tilting. This will be done by designing an obstacle course track, requiring the player to make precise turns (Horizontal LEFT and RIGHT tilts).

- Easy difficulty: The ramps are wide. Falling off will reset the player to their last checkpoint gotten.
- Normal difficulty: The ramps are narrow. Falling off will reset the player to the beginning of the level.

The design of both Easy and Normal difficulty for these levels will involve changing the following game mechanics:

	Level 1	Level 2
Easy	Collect all checkpoints around track Track boundary walls Re spawn at last checkpoint	Collect all checkpoints on ramps Wide ramps Re spawn at last checkpoint
Normal	Collect all checkpoints around track No track boundary walls Re spawn at start of level	Collect all checkpoints on ramps Narrow ramps Re spawn at start of level

Table 1: Level Features

4 Implementation

The following is a summary of each system implementation described in this chapter.

- Prototype 1 - Block Dodge: This is the first prototype developed for the software. The aims of this prototype were to demonstrate the first iteration of the translation from head tilt data, to video-game directional controls. This only focused on the roll data(horizontal head tilts).
- Calibration 1 - Minimum User Control: This calibration scene was created as the first visual feedback of the users head tilts. This scene is intended to allow the user to adjust and learn the basic range of head movements that are required by the game.
- Calibration 2 - Horizontal: This calibration is a short test sequence, to measure the users accuracy when tilting on the horizontal plane only.
- Calibration 3 - Vertical: This scene is the same as Calibration 2, except it is done on the vertical plane. When completed the user is shown a general feedback screen, indicating their accuracy attained in the Calibration 2 and Calibration 3.
- Performance Data Collection: This scene informs the user of their level of skill, in relation the level of skill required to complete each level of the game. They can now make an informed decision on the level of difficulty they choose to play the game at.
- Prototype 2 - Tetris: This prototype was implemented as a proof of concept to show roll(horizontal head tilts) and pitch(vertical head tilts) being used together in an application.
- Drive-On Game: This was the final implementation of the head tilt input controller, to combine the functionality of each preceding implementation.

4.1 Prototype 1 - Block Dodge

The first prototype for this software was a simple game where the players, represented by a red ball, dodges falling blocks by tilting their head left or

right. This was the first implementation of the head tracking data translation to a game controller (see figure 6). While analyzing the head tilt data (roll represented as a floating point value), it was made apparent that each item of data fell within a range.

- roll

  fully tilted left = 0.3

  fully tilted right = -0.3.

Figure 6: Prototype 1 - Block Dodge

4.1.1 Goals of Prototype 1

This prototype was developed as a proof of concept to prove that the gaze data being recieved from the server could be manipulated and used as a controller for a unity application. During this stage of development, only the roll value was being used, therefore there was no vertical controller yet.

4.2 Calibration 1 - Minimum User Control

The calibration scene is the first interaction the player has with the system. This 30 second scene is used to calculate the accuracy of the players head tilting, and adjust the games difficulty scale based on their score. This is to help the game remain playable for people with less dexterity than others.

The calibration begins with a discreet calibration scene(tilt head in direction of highlighted square). Precision does not matter here, it is a test that the program is working and that the user can reach each angle of head tilt. If this section is not complete, The user user is prompted to re-orient the camera and confirm the lighting is sufficient.

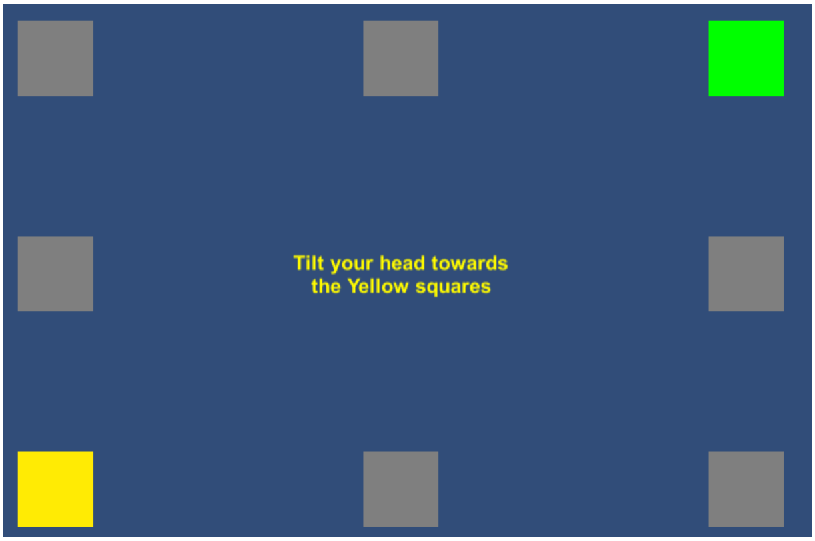


Figure 7: Descreet Calibration Scene. When a square turns yellow, the player tilts their head in the direction of that square. The square will turn green if the tilt was accurate. The exercise ends when all squares are green.

4.3 Calibration 2 - Horizontal

Next is the horizontal calibration (slide ball into pink squares with head tilts). This section involves the player controlling a ball with their head tilts (similar to prototype 1), and positioning the ball inside the highlighted boxes as showing in Figure 8.

Figure 8: Horizontal Calibration Scene

This scene counts down from 30 seconds, while changing the location and size

of the target squares as the time goes on. This difficult of this calibration scales with the time by making the target squares narrower as can be seen in Figure 9, requiring the players need to make very precise tilts. This is to analyze how precise the player is with their head tilts.

Figure 9: Horizontal Calibration Scene

4.4 Calibration 3 - Vertical

This scene has the exact same mechanics as Calibration 2 - Horizontal, except the target(pink) squares are no positioned along the Y axis as can be seen in 10. This requires the player to tilt their head UP and DOWN to move the ball to the squares. Upon analyzing the float ranges representing pitch movement, The ranges recorded are as follows.

- pitch

  fully tilted up = 0.8

  fully tilted down = 0.4.

Figure 10: Vertical Calibration Scene

Similarly to Calibration 2, The difficulty of this scene scales with the time, by making the target squares narrower.

4.5 Performance Data Collection

The players performance for both Calibration 2 and Calibration 3 are calculated by using the following equation:

$$accuracy = \frac{timeInSquare}{allowanceMod(totalTime)} * \frac{100}{1}$$

Figure 11: Formula to determine players calibration accuracy

- **accuracy**: the final % score of the players performance during the calibration.
- **timeInSquare**: the ammount of seconds the player maneuvered the 'ball' into the highlighted area.
- **totalTime**: The countdown timer for the calibration test.
- **allowanceMod**: The modifier applied to the total time to make up for 'travel time' between highlighted squares. eg. totalTime = 30s, allowanceMod = 0.75, therefore the player will be scored on their timeInsquare / 22.5.

This calculation is done separately for Calibration 2 - Horizontal and Calibration 3 - Vertical, and combined together when both calibrations are complete. The product of this combination is then divided by 2, to produce the mean performance attained. The separate performance for Horizontal and Vertical are still maintained for correlation testing.

4.5.1 Logging Results

The results from the calibration are logged using the unity 'PlayerPrefs' class. This class allows us to store a value with a key, and subsequently retrieve this value using the corresponding key. This data is deleted when the game is stopped, therefore the data was logged to an external text file for analyzing each players performance.

```
4/1/2019 3:43:44 PM
HorizontalAccuracy, 54
VerticalAccuracy, 80
Mean, 67

4/1/2019 3:49:44 PM
HorizontalAccuracy: 45
VerticalAccuracy, 58
Mean, 51.5

4/1/2019 3:52:46 PM
HorizontalAccuracy: 72
VerticalAccuracy, 63
Mean, 67.5

4/1/2019 3:55:29 PM
HorizontalAccuracy: 14
VerticalAccuracy, 49
Mean, 31.5
```

Figure 12: This log file shows the score each player attained during calibration.

4.6 Prototype 2 - Tetris

The second prototype for this game was the classic Tetris game. Unlike prototype 1, this game included use of both horizontal and vertical head tilt input. During this stage, a generalized input controller layer API was created, containing a collection of useful Get methods which returned the head tilt data. This custom input layer was created to provide a universal controller that can be adapted to any video game that uses UP, DOWN, LEFT and RIGHT control mechanics. Once complete, adapting this layer to the Tetris game required minimal 'hacking' of the controller code. Combining the float ranges representing head tilts on both axes, the final analyze ranges are as follows.

- roll

  fully tilted left = 0.3

  fully tilted right = -0.3.

- pitch

  fully tilted up = 0.8

  fully tilted down = 0.4.

- when head is perfectly centered

  roll rests at 0.0

pitch rests at 0.6.

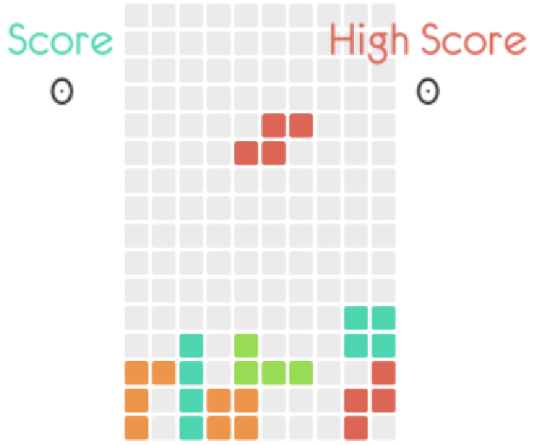


Figure 13: Prototype 2 - Tetris

By default, the blocks in the Tetris were controlled by UP, DOWN, LEFT and RIGHT keypad arrows. These inputs were responsible for the following:

- UP: Rotate the Tetris block,
- DOWN: Speed up the descent of the falling block,
- LEFT: Move the block one space to the left,
- RIGHT: Move the block one space to the right.

The only changes made to these default controls was substituting the directional arrow key press for the subsequent head tilts. This provided a solid proof of concept for the slot-in ability of the custom head tilt controller layer.

4.7 Drive-On Game - Version 1

The controls for Drive-On were implemented using the custom input layer built for the prototype 2, with minor changes. The controls corresponding to the head tilts are as follows:

- UP TILT: Accelerate,
- Down TILT: Decelerate, reverse when stopped,
- LEFT TILT: Turn left,
- RIGHT TILT: Turn right.

Modifying the thresholds that register a particular head tilt direction needed to be changed on the pitch(UP and DOWN) axis. This was because the threshold for tilting up was too high, making it so the player needed to tilt their head up too far to accelerate, obstructing the view of the screen. The pitch threshold was lowered to combat this.

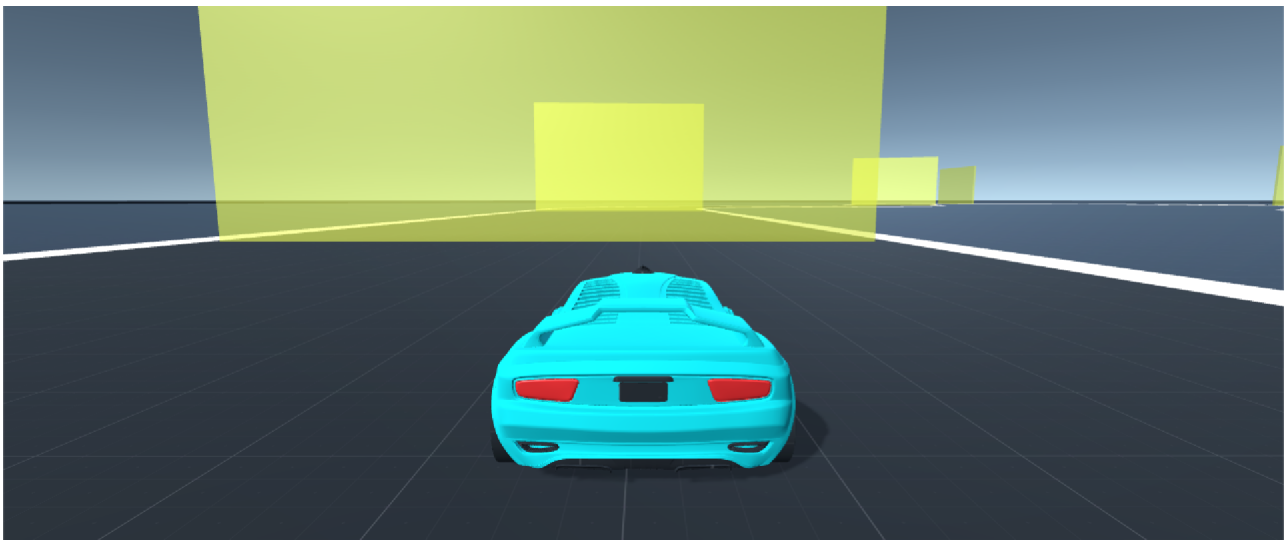


Figure 14: Level 1 player point of view.

4.7.1 Level 1

This level was implemented with a bias towards vertical tilting. To do this, a wide track was made to create wide turns with enough space for sloppy turning. The subsequent Easy and Normal difficulty modes as described in Table 1 can be seen in Figure 15 (Easy) and Figure 16.

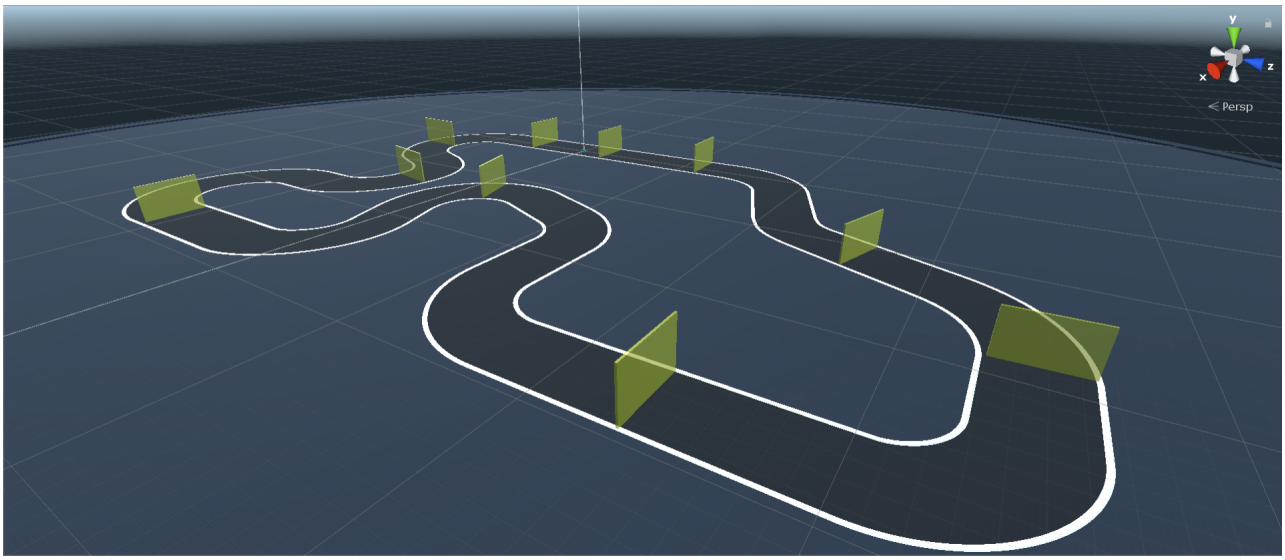


Figure 15: Level 1 track. Normal Difficulty.

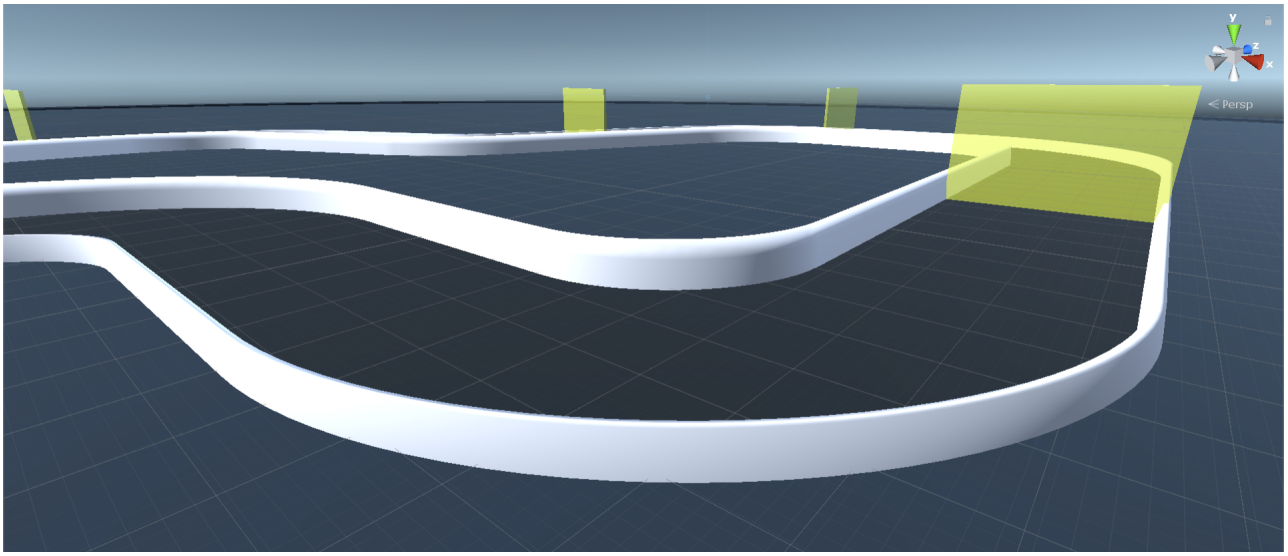


Figure 16: Level 1 Easy difficulty. Edge verges are raised.

4.7.2 Level 2

This level was implemented with a bias towards Horizontal tilting. To do this, the same track was used, except with ramps that the player must carefully navigate up without falling off. This can be seen in Figure 17.

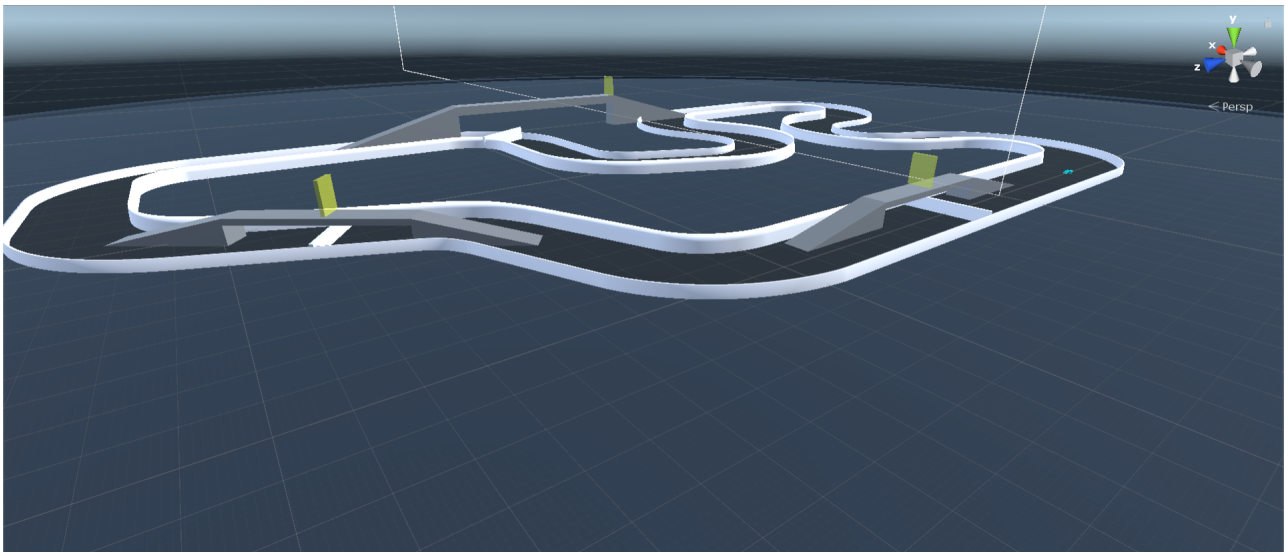


Figure 17: Level 2 track. Easy Difficulty.

The subsequent Easy and Normal difficulty modes as described in Table 1 can be seen in the visual representation of the table in Figure 18.

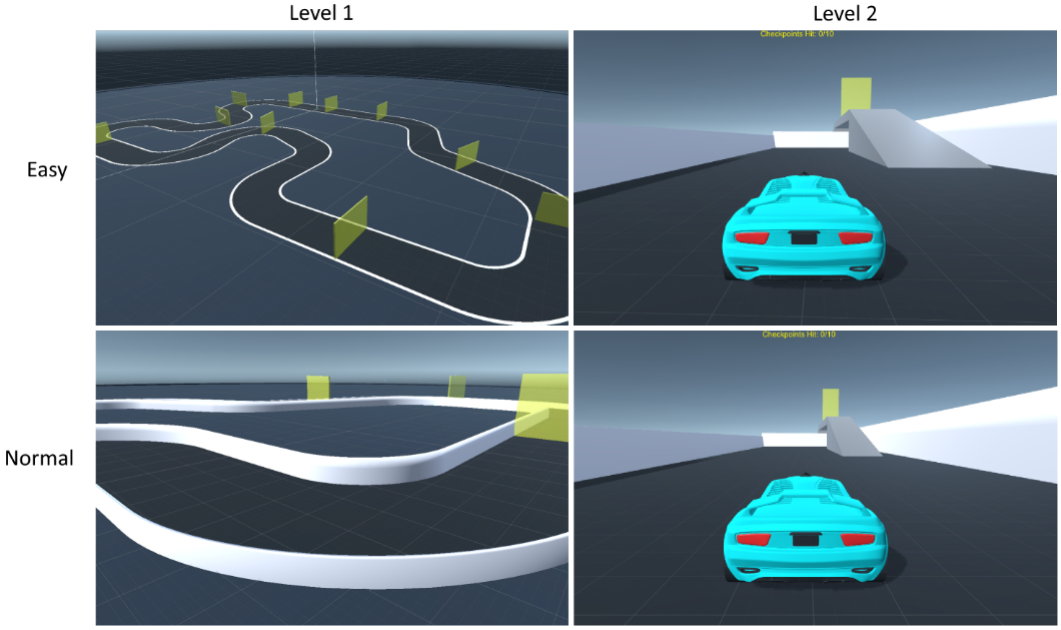


Figure 18: Visual representation of Table 1.

4.7.3 XLabs Gaze Learning Mode

The XLabs gaze API discreetly adjusts its accuracy throughout an iterative training process. It does this by associating the persons head and face position patterns with the outputted coordinates. When a head is recognized by the program, the patterns created when a person tilts in a particular direction are used to re-adjust the parsing process, so the outputted values representing tilts remain uniform within the general ranges regardless of differences in peoples varying positions on camera. This was implemented in this version of Drive-On, as well as Calibration 1, 2 and 3. This was to accommodate for the varying participants that tested the game.

4.8 Drive-On - Verson 2

During testing of the game, it became apparent that it was sometimes difficult to gauge how far to tilt to produce consistent turns. By only having the car as visual feedback of your head tilting controller in action, it was not uncommon to turn far too much, or far too little. To combat this, a horizontal slider was added to Drive-On - Version 1. The white nub on this slider moves

in proportion with your horizontal head tilts, providing a virtual steering wheel-like controller. This made it much easier to learn and gauge the tilting required for the turns. This can be seen at the bottom of Figure 19.

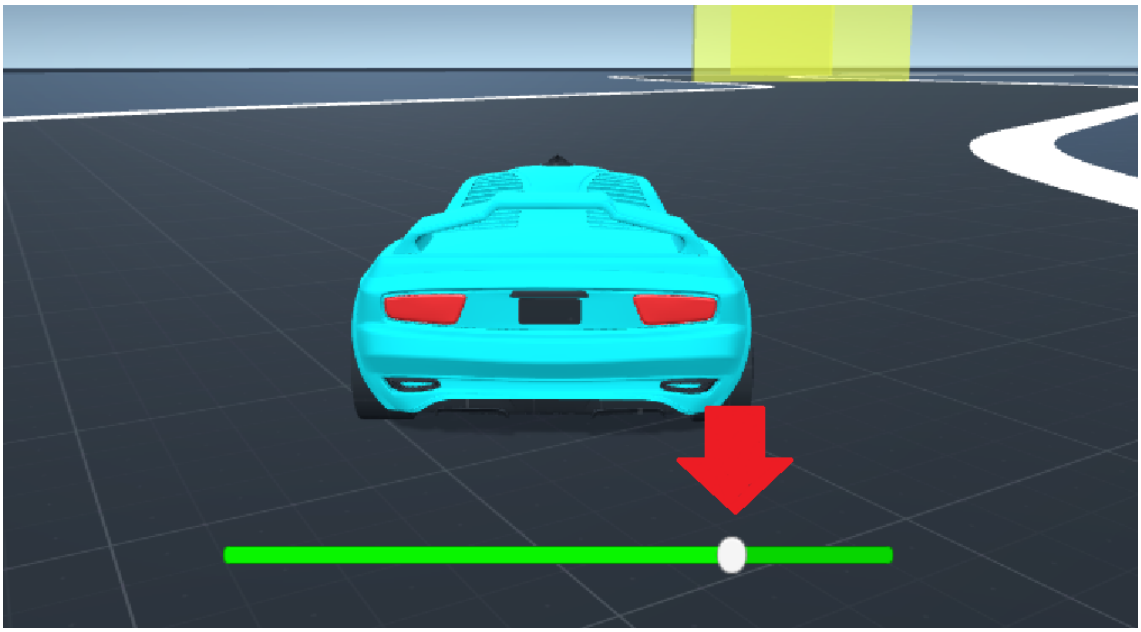


Figure 19: Horizontal slider to give visual feedback of head tilts.

5 Qualitative Evaluation Method and Results

The users who tested this software were given a survey to help distinguish any quality issues and whether the the software can be adjusted to suit these. The informal survey was conducted using a Likert-scale format [15], which gave us qualitative data about the subjects. Please see Figures 20(Survey before playing) and Figure 22(Survey after playing).

Pre-experiment

	-2	-1	0	1	2
1. I play video games recreationally	[ ]	[ ]	[ ]	[ ]	[ ]
2. I consider my dexterity as optimal	[ ]	[ ]	[ ]	[ ]	[ ]
3. I struggle with hand-eye coordination in video games	[ ]	[ ]	[ ]	[ ]	[ ]
4. My reflexes are good	[ ]	[ ]	[ ]	[ ]	[ ]
5. I am interested in alternative ways to play video games	[ ]	[ ]	[ ]	[ ]	[ ]
6. I have access to a generic webcam	[ ]	[ ]	[ ]	[ ]	[ ]
7. I consider myself adept at using my computer	[ ]	[ ]	[ ]	[ ]	[ ]

Figure 20: Pre-experiment survey.

5.1 Pre-Experiment

The format of this Likert survey was a range of numbers, lower numbers representing disagreement with the proposed statement, and higher numbers representing agreement. These numbers as seen in Figure 20 are -2, -1, 0, 1, 2. The results gathered from this format of Likert survey allow us to get a relative value for the overall consensus of each question. This can be calculated by using the following:

- Number of questions * -2 = min value,

- Number of questions * 2 = max value.

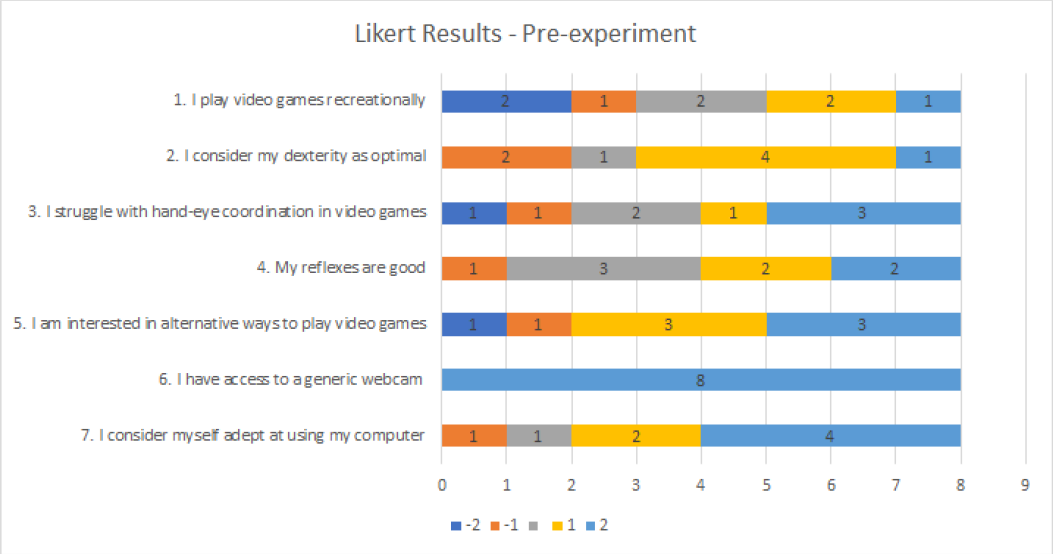


Figure 21: Pre-experiment survey results.

The results shown in Figure 21 can be represented by multiplying the carnality of each answer by the number of participants. For example: Question 1: (2*-2) + (1*-1) + (2*0) + (2*2) + (1*2) = 1. In this pre-experiment there were 8 participants. Therefore, The min value is -16 and max value is 16. Question one gave us a result of 1, telling us that consensus was almost exactly balanced between disagreement and agreement. The results for each question using this formula are as follows:

- Question 1: 1
- Question 2: 4
- Question 3: 4
- Question 4: 5
- Question 5: 6
- Question 6: 16

- Question 7: 9

As can be seen from these results, a conclusion can be drawn that there is an overwhelming agreement with question 6, as well as a strong agreement with question 7. The lowest value here is given by question 1, telling us that it will most likely not be a predictive data item.

5.2 Post-Experiment

After the participant played through the game, they were given another survey in a similar style to the previous one, however this survey was intended to collect some less informal data, to get a better understanding of the relationships between the players background and their opinion of the experience.

Post-experiment

	-2	-1	0	1	2
1. The experience was enjoyable	[ ]	[ ]	[ ]	[ ]	[ ]
2. The controls were easy/intuitive	[ ]	[ ]	[ ]	[ ]	[ ]
3. The controls frustrated me	[ ]	[ ]	[ ]	[ ]	[ ]
4. The objective of the game was clear	[ ]	[ ]	[ ]	[ ]	[ ]
5. I would like to use this input in other games instead of my hands	[ ]	[ ]	[ ]	[ ]	[ ]

Figure 22: Post-experiment survey.

The results of the post-experiment survey were analyzed using the same techniques as the previous survey. The results gathered are shown in Figure 23 as follows:

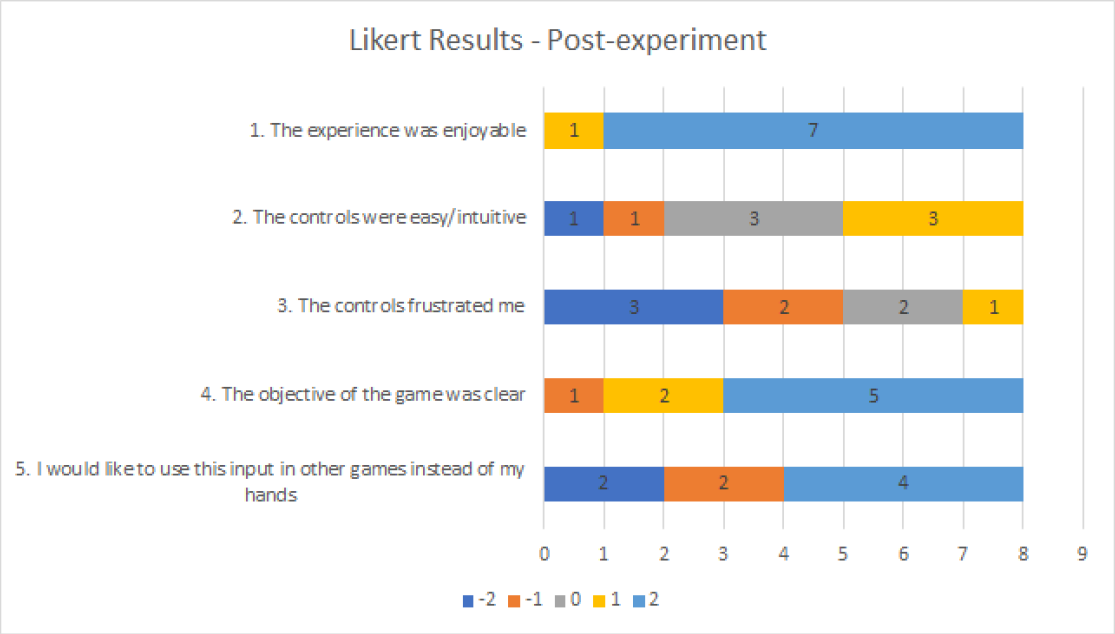


Figure 23: Post-experiment survey results.

- Question 1: 15
- Question 2: 0
- Question 3: -7
- Question 4: 11
- Question 5: 2

5.3 Conclusions

The results here provoked some interesting theories. As evident from the results of question 1 in Figure 23, the overall consensus of the participants enjoyment of the game was very positive, however there is some discrepancies with their desire to use this as a tool in other settings shown bt question 5. The frustration is an indication of the performance of the joystick itself, as after all, this software is essentially a new take on the traditional games joystick. This indicated that there is still a desire for refinement of the software. Note that while the results for question 3 are negative, this represents

a positive indication. This is because the question asks if the controls were frustrating. Disagreement with this question are consistent with the results of question 1. Question 3 can be reversed when analyzing this data.

6 Quantitative User Evaluation Method and Results

The process of quantitative data collection for this project involved investigating the participants performance in relation to the games mechanics and the Likert data collected. This data was investigated via 3 hypotheses targeting the participants horizontal and vertical head tilt performance, and the impact their performance had on their enjoyment of the game. The data collected from each participant was labeled and represented in a CSV file. Each participant amounted to 22 pieces of analyzable data, which was analyzed using Rapid Miner Studio. The participant data included:

- Vertical/Horizontal calibration performance,
- level 1 "easy" and "normal" performances (labeled level 1 'a' or 'b'),
- level 2 "easy" and "normal" performances (labeled level 2 'a' or 'b'),
- The result of the 7 questions asked to each participant before playing,
- The result of the 5 questions asked to each participant after playing,
- The subsequent means of each calibration performance and level performance.

6.1 Vertical Head Tilt Performance

Hypothesis H1: *Players who score higher on the vertical calibration stage perform better on level 1 than level 2.*

The mechanics of game level 1 as described in the 'Implementation' chapter involved the player driving around the track in the shortest possible time. The vertical head tilting mechanic is responsible for controlling the speed of the car whereas the horizontal head tilt mechanic controls the steering. While level 1 does contain turns for the car to make, the most important mechanic is arguably the acceleration and deceleration of the car to make these turns while still maintaining a good time. Therefore, this vertical mechanic was investigated in regards to level 1, to find a correlation between

the participants vertical head tilt abilities and their speed of completing the level.

This was done by analyzing the performance of the participants vertical head tilt calibration, and comparing it to the mean of the participants level 1 'easy' and 'normal' performance. The same was done with the vertical head tilt calibration results in relation to level 2 - easy and level 2 - normal in order to reveal any patterns present. note: the time measured to complete each level is in seconds, and a lower time is 'better' performance.

Step 1: Comparing Vertical calibration performance to level 1 performance.

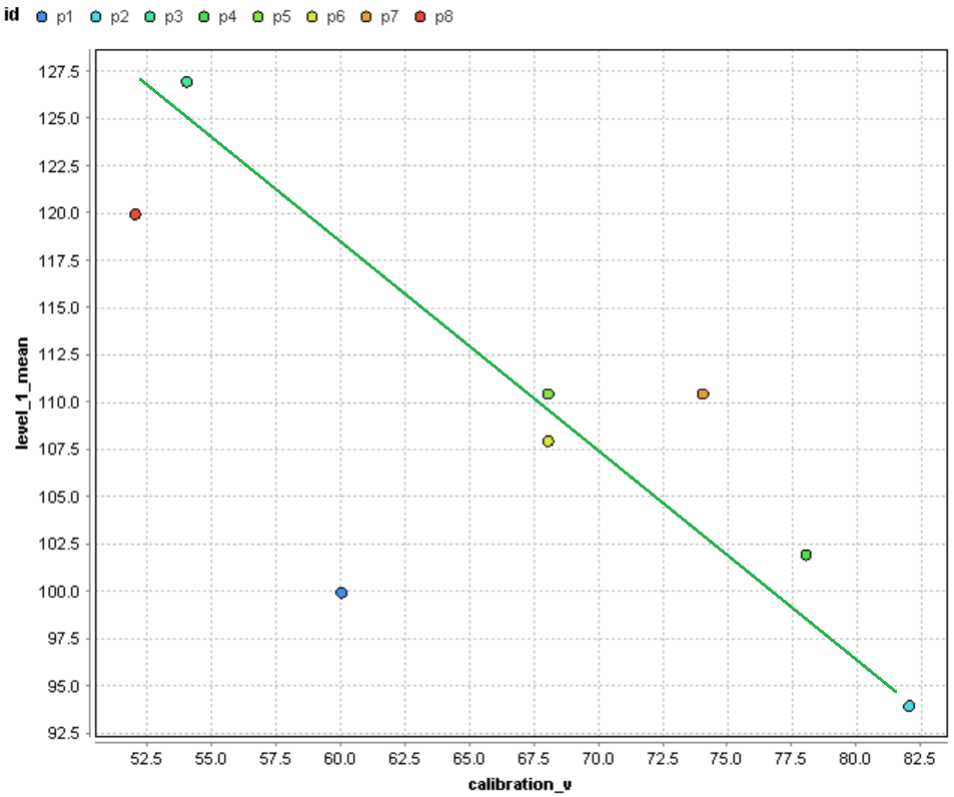


Figure 24: Vertical calibration performance compared to level 1 performance.

Step 2: Comparing Horizontal calibration performance to level 1 performance.

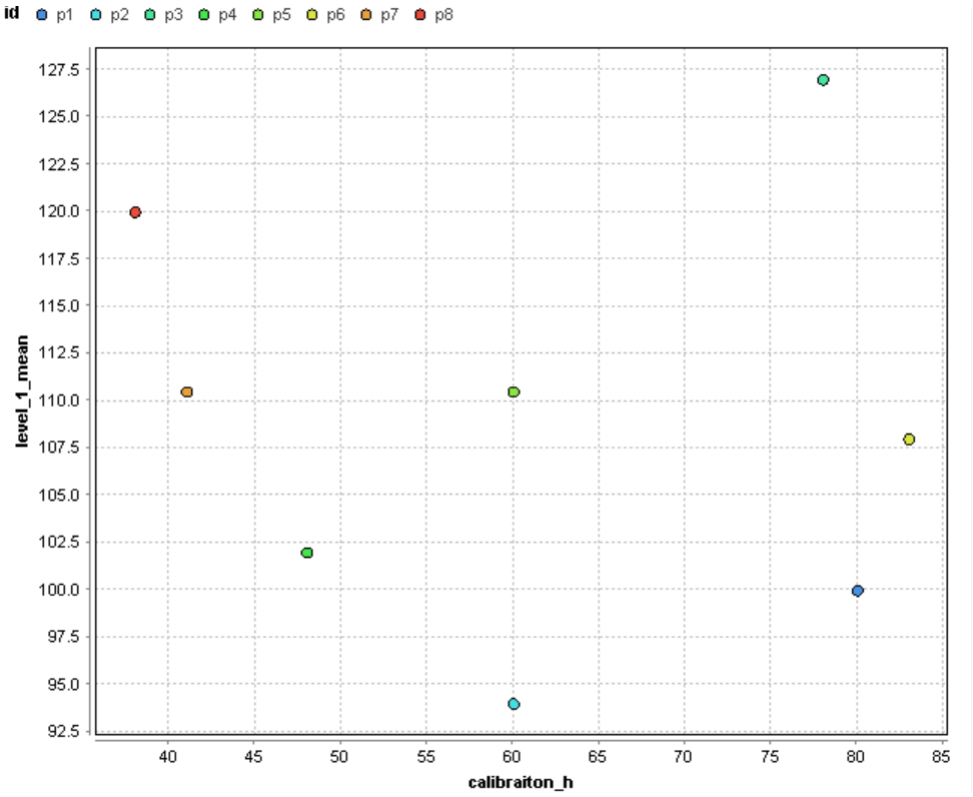


Figure 25: Horizontal calibration performance compared to level 1 performance.

Step 3: Analysis.
As can be seen by **figure 24** and **figure 25**, there is a clear correlation between the vertical calibration performance and level 1 time, while no clear correlation between vertical calibration performance and level 2 time. This relationship was represented in Figure 26 and Figure 27:

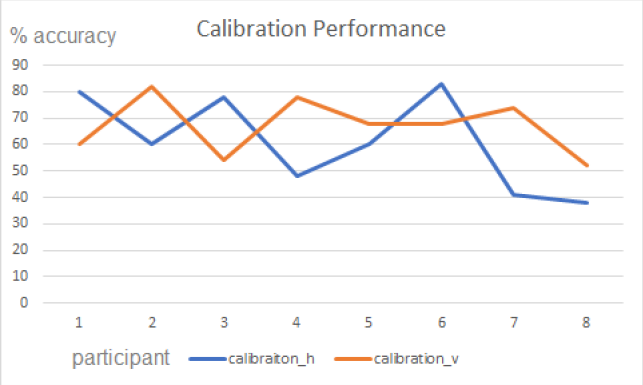


Figure 26: Calibration performance.

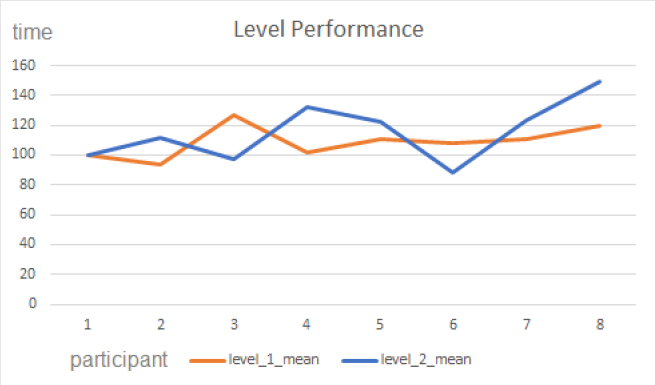


Figure 27: Level performance.

Conclusions:
The higher calibration accuracy for each participant is positively correlated to the lower(better) time/performance in the subsequent level. Therefore Hypothesis H1 is supported by the evidence that Level 1 performance is directly positively correlated with level 1 time/performance.

6.2 Horizontal Head Tilt Performance

Hypothesis H2: *Players who score higher on the horizontal calibration perform better on level 2 than level 1.*

The mechanics in level 2 involve the player carefully and methodically aligning the car to navigate up narrow ramps to progress through the level. While performance in this level is measured the same way as level 1 (time taken to complete it), the skill needed to complete level 2 differ from level 1, as level 2 is designed to require better horizontal head tilting accuracy. This theory was tested to find a correlation between horizontal head tilt performance and level 2 performance.

Step 1: Comparing Vertical head tilt performance to level 2 performance.

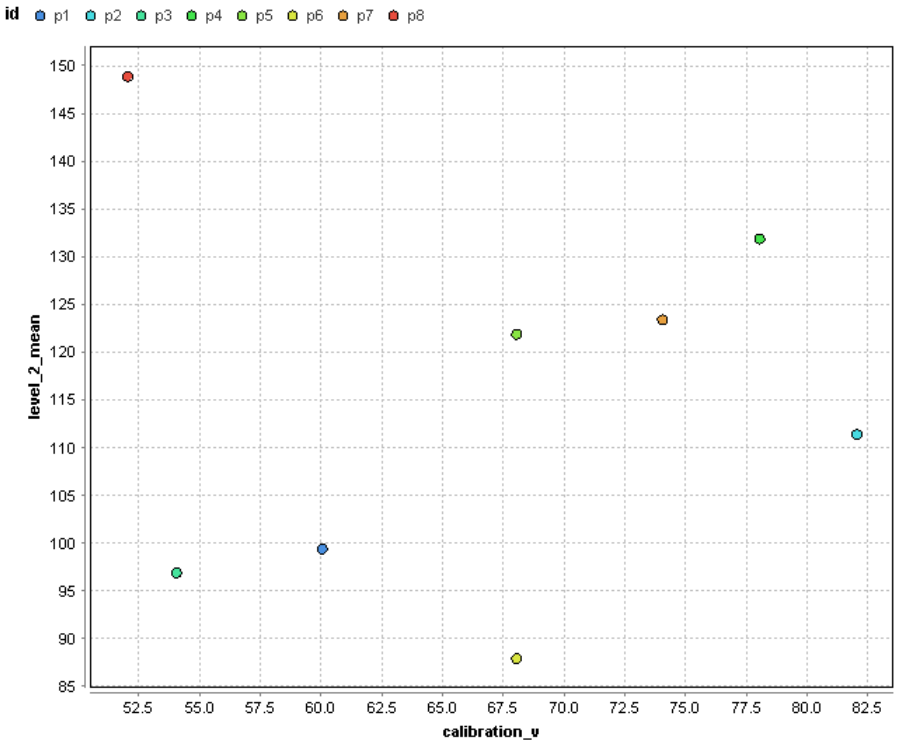


Figure 28: Vertical calibration performance compared to level 2 performance.

Step 2: Comparing Horizontal head tilt performance to level 2 performance.

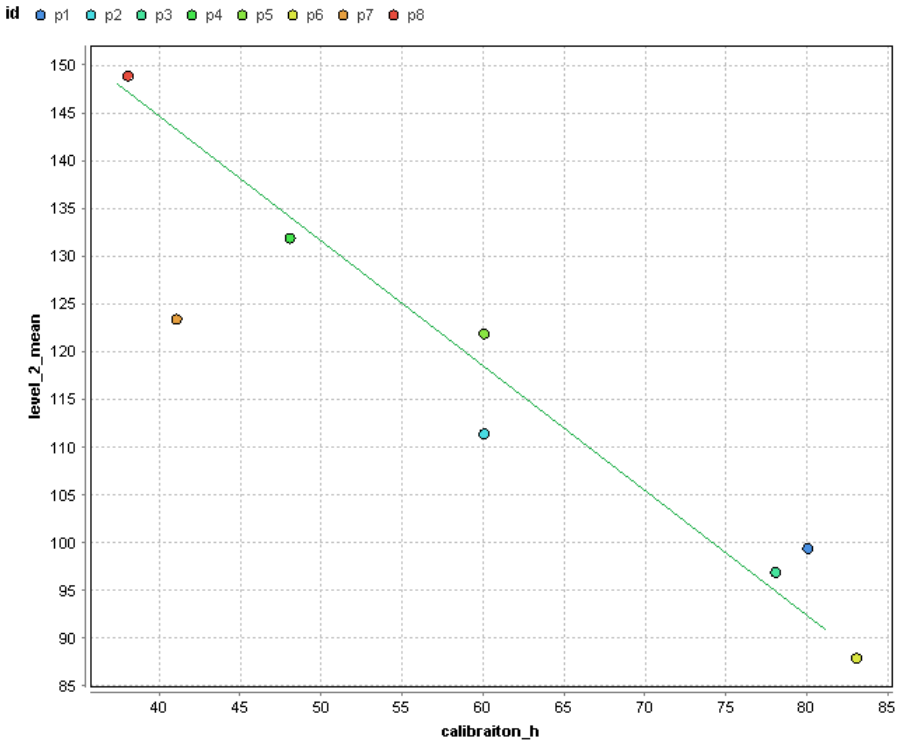


Figure 29: Horizontal calibration performance compared to level 2 performance.

Step 3: Conclusions.
In **figure 28** There is no substantial correlation present. In **figure 29** shows a clear positive correlation. This lines up with the proposed hypotheses and is further confirmed when referring back to the level and calibration line charts in **figure 26** and **figure 27**.

6.3 Desire For This Software

Hypothesis H3: *Participants who performed well on the game have a higher desire to utilize head tracking software as an alternate input in other games.*

In the Likert survey presented to the participant after their experience playing the game, question 5 asks the participant if they would 'like to use this input in other games' instead of their hands. This results of this question in

particular were investigate to find a correlation between game performance and desire to use the software again.

Step 1: Comparing game performance to answer chosen for question 5 on the likert survey after playing.
The procedure for this experiment involved creating representing the answer of the survey as integers (-2 = Strongly disagree ... 2 = Strongly Agree) as well as a calculation of the players average overall performance in both levels of the game. This data set was analyzed using Rapid Miner Studio and the results are as follows:

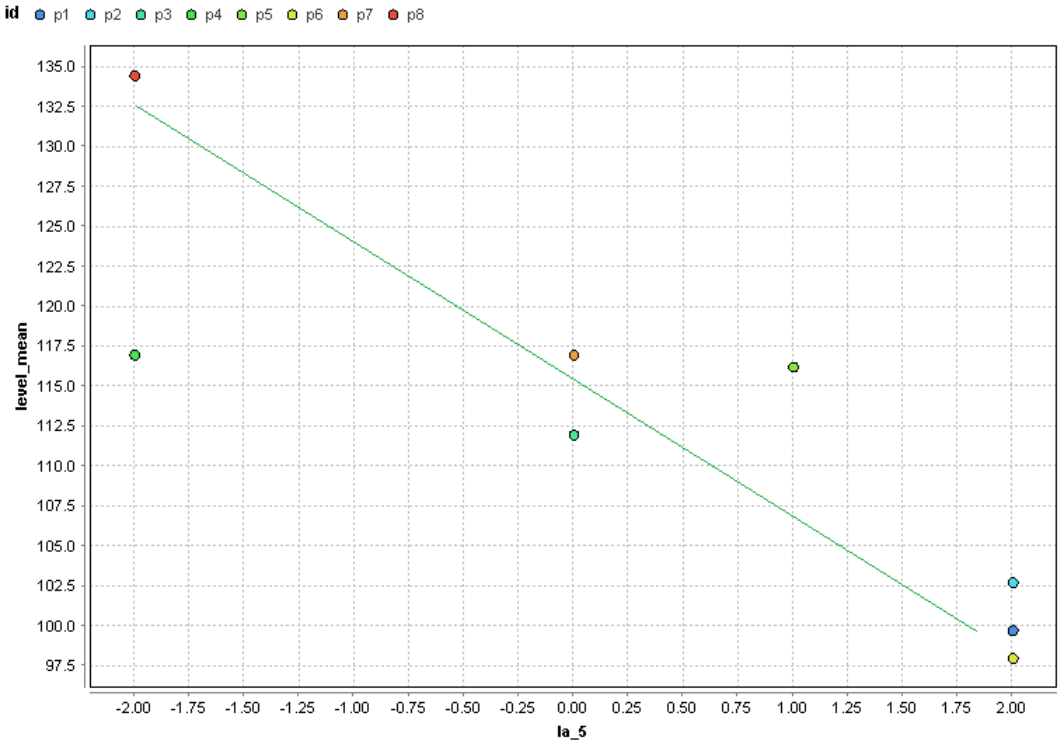


Figure 30: Chosen answer (between -2 and 2) rating the participants desire to use the software plotted again their overall performance in the game.

Step 2: Conclusions.
As can be seen in **figure 30**, The participants performance (seconds taken

to complete game. lower amount of seconds = better performance) and their rating given for the question of their desire to use this software in the future shows a positive correlation. This denotes that players may be less likely to enjoy the software if the game is too difficult.

7 Technical Evaluation

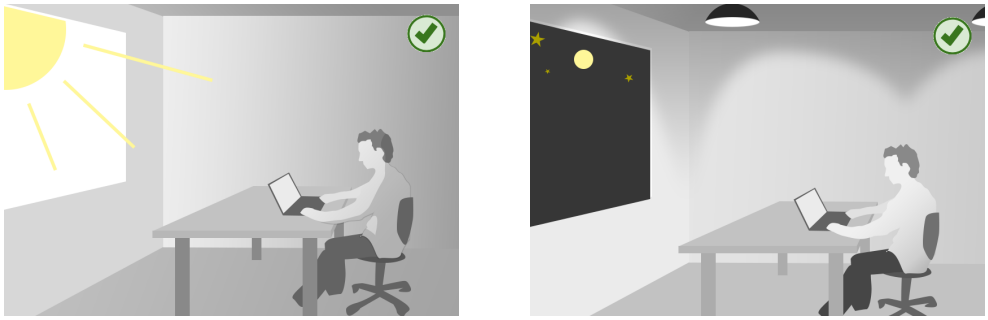
Finding the optimal metrics for the software physical environment was an important step in maximizing game performance. When running this software, there are three major physical variable which need to be adjusted for optimal software performance. These variables are:

- Webcam lighting
- Internet connection (as the gaze monitoring algorithm is fetched from an external source.)
- Physical system resources.

These variables were investigated in the following sections.

7.1 Webcam Environment Requirements

The lighting played a major role in the mechanics of this project. Bad lighting would produce inconsistent polling of the head location data, which in turn created a frustratingly choppy gaming experience. It was made apparent that a strong light source directly behind the camera provided the most consistent experience, as well as a uniformly lit room at night time.



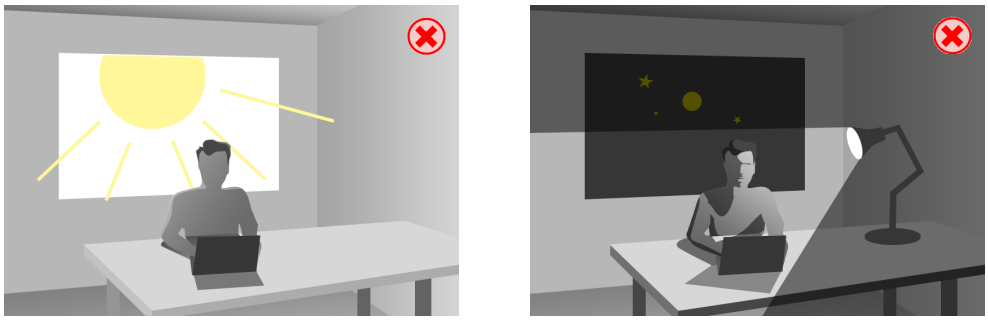
(a) Day time ideal environment

(b) Night time ideal environment

Figure 31: Ideal webcam environment

The inverse of this was light sources that were behind the subject and directional light coming from the sides. This would produce false positives in

the facial monitoring program as the landmarks of the face would be more difficult to discern when there is a shadow being cast on the face.



(a) Bad natural light sorce position (b) bad artificial light source position

Figure 32: Bad webcam environment

7.2 Internet Speed Testing

The process of testing the relationship between internet speed and performance of the game began by testing the game with three different connection architectures.

1. Private wireless broadband
2. Public wireless broadband
3. 4G mobile data

The performance was measured by way of recording the polling frequency of the head tilt data. This frequency was found to have a direct relationship with the responsiveness of the in-game vehicle, being controlled by head tilts. Additionally, the amount of time between an action(head tilt) and response in-game. The results are as follows:

	Polling Frequency (Hz)	Latency (ms)
Private Wireless Broadband	43	710
Public Wireless Broadband	36	835
4G Mobile Data	39	817

These results may appear to be indicative of the varying negative effects(higher latency) caused by slower network connections, however, with

the standard deviation being ¡ 100ms, This is extremely minimal/negligible in the context of input latency.

Testing this software involved having a number of players blind test the calibration scene. The data gathered from these tests were analyzed using R to find how intuitive the gaze-tracking system is, and what accuracy first time users could achieve.

The metrics used to make this assessment were:

- Distance between players face and the webcam,
- Facial obstructions (hair, glasses),
- The players self-scored dexterity recorded in the likert scale survey,
- The accuracy achieved after completing the test based on the accuracy formula.

7.3 System Resource Testing

Throughout the lifetime of the game software, there arose a mysterious problem where there were massive latency delays between a head tilt, and the in-game controller reacting. These delays appeared sporadically and were initially believed to be a problem caused by using a slower internet connection. As seen by the previous internet testing, there was not obvious reason discovered for these latency spikes. The next suspect was the resource usage of the system where this software and game have been running on

To test the relationship between system resources and game performance, four tests were devised. These tests involved running the game at different resolutions, and monitoring the system resources and game performance variables. These tests were intended to prove the correlation between FPS (frames per second) and the polling rate at which head tilt data is being received by the game, and the subsequent reaction by the in game controller. The results were as follows:

Resolution	FPS	CPU %	GPU %	Memory %	Frequency Hz	Delay ms
560 x 270	210	90	68	57	12	780
770 x 380	157	92	75	57	12	820
950 x 450	127	91	83	65	10	2100
1280 x 550	95	92	97	59	11	3800

Some of the attributes in this table were revealed to have no different impacts on game performance. These include CPU, Memory, and Frequency (frequency of head tilt data received).This discourages the initial theory that a lower frame rate decreases the polling frequency. This tells us that the frequency is not involved in the large delay spikes between head tilt and controller action, however, as notable in the table, raising the resolution was decreasing fps and increasing GPU usage as well as the delay.

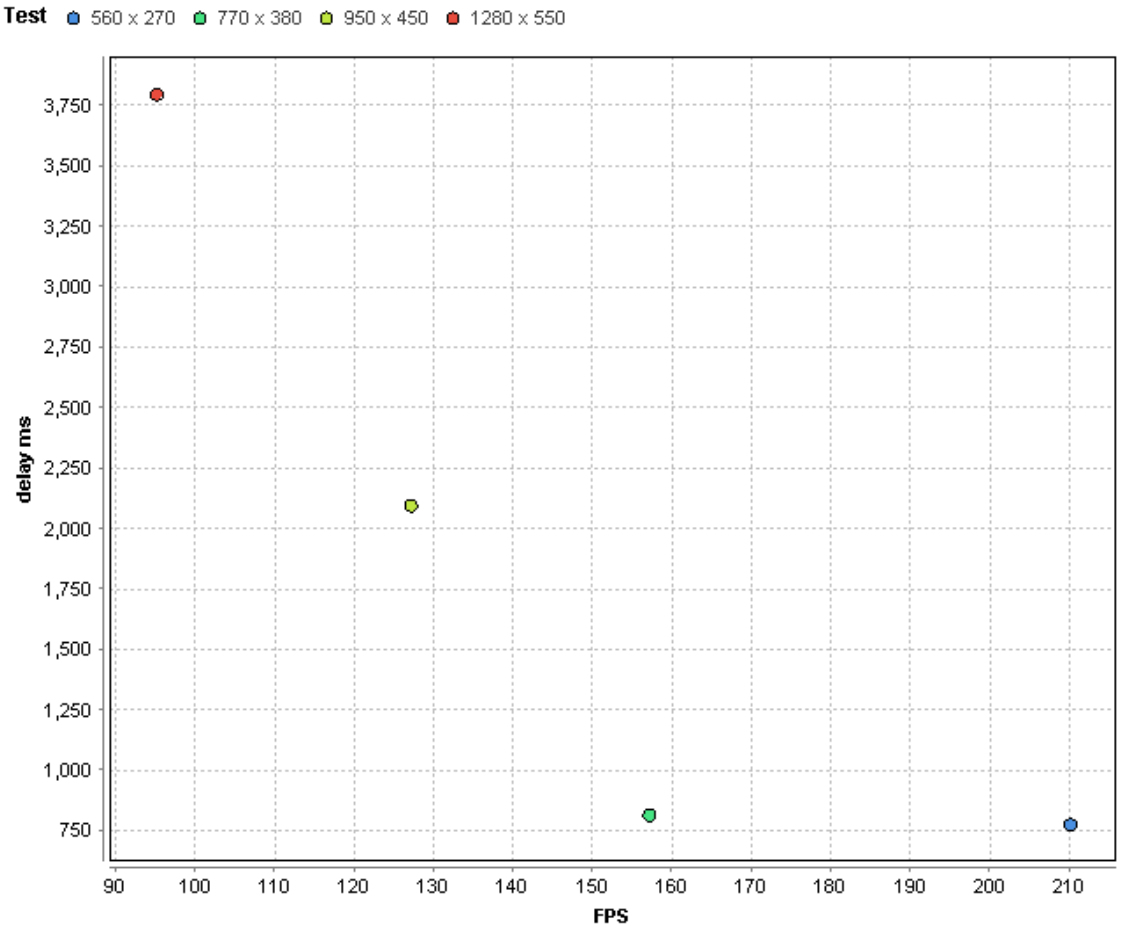


Figure 33: Relationship between FPS and latency

This scatter plot proves the theory that lower FPS results in lower latency. 820ms latency between head movement and in-game movement is playable, meaning it is a low enough latency to go unnoticed. The rise of in game latency relative to resolution is exponential. This concludes that, for the system running these tests, Keeping GPU usage below 80% was crucial, otherwise creating a frustrating experience due to long delays in game.

8 Conclusions

This research project had three major aims. These aims included:

1. Creating ab XLLabs gaze communication API for Unity,
2. Creating an application input layer that can use head tilt data to control other games,
3. Analyze the usability and robustness of this software through testing.

While using gaze tracking with a webcam is not ideal, This project proves the alternate route of head tracking can be used in place of eyeball tracking, without need for an infrared camera. Tracking the head limits the uses for gaze tracking, however, it lowered the camera fidelity requirement enough to make a usable controller layer for games that use UP, DOWN, LEFT and RIGHT inputs. Furthermore, due to the direct translation achieved of head tilts to keypad arrow controls, This implementation can further be adapted to user-interfaces with similar functionality. While there are numerous accessibility options for colorblind and hearing impaired people, this research project intends to contribute to avenues of accessible gaming and application navigation options, for people with limited limb-mobility.

9 Further Work

Due to time constraints, some components of this project were limited. Further research for this project would include the following:

- Acquisition of an infrared camera,
- Implantation of any Open-source infrared gaze tracking libraries,
- Comparing user performance using different hardware set-ups,
- Optimization of the Drive-On game, to maximize the performance potential of the gaze tracking algorithm,
- A proof of concept and implementation of a user interface controlled by head tilts

References

[1] Amer Al-Rahayfeh and Miad Faezipour. Eye Tracking and Head Movement Detection: A State-of-Art Survey. *IEEE journal of translational engineering in health and medicine*, 1:2100212, 2013.

[2] Nick Battalgia. *The video game industry needs to fix its relationship with disabled gamers*. 2018.

[3] Kevin Bierre, Jonathan Chetwynd, Barrie Ellis, D M Hinn, Stephanie Ludi, and Thomas Westin. Game not over: Accessibility issues in video games. *Proc of the 3rd International Conference on Universal Access in HumanComputer Interaction*, (November 2014):22–27, 2005.

[4] PHD Bryn Farnsworth. *Imotions eye tracking work*. 2018.

[5] Marcus Carter, Joshua Newn, Eduardo Velloso, and Frank Vetere. Remote Gaze and Gesture Tracking on the Microsoft Kinect. *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction on - OzCHI '15*, (December):167–176, 2015.

[6] Michael Chau and Margrit Betke. Real Time Eye Tracking and Blink Detection with USB Cameras. *Boston University Computer Science Technical Report*, (2005-12):1–10, 2005.

[7] Christopher Crockford, Judith Newton, Katie Lonergan, Theresa Chiwera, Tom Booth, Siddharthan Chandran, Shuna Colville, Mark Heverin, Iain Mays, Suvankar Pal, Niall Pender, Marta Pinto-Grau, Ratko Radakovic, Christopher E. Shaw, Laura Stephenson, Robert Swingler, Alice Vajda, Ammar Al-Chalabi, Orla Hardiman, and Sharon Abrahams. Als-specific cognitive and behavior changes associated with advancing disease stage in als. *Neurology*, 91(15):e1370–e1380, 2018.

[8] Prof Girraj and Prasad Rathor. A New DCT Precoding Based RSLM Method for PAPR Reduction in OFDM System. 3(6):1302–1305, 2012.

[9] Tersia Gowases, Roman Bednarik, and Markku Tukiainen. Gaze vs. mouse in games: The effects on user experience. *Proceedings of the International Conference on Computers in Education*, pages 773–777, 2008.

[10] David Helgason and Joachim Ante. https://unity.com/.

[11] Sebastian Höffner and Prof Gunther Heidemann. Institute of Cognitive Science Biologically oriented Computer Vision Master ' s Thesis Gaze Tracking Using Common Webcams. 2018.

[12] Robert J. K. Jacob. What you look at is what you get: eye movement-based interaction techniques. *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*, pages 11–18, 1990.

[13] Robert J. K. Jacob. *Eye Tracking in Human–Computer Interaction and Usability Research*. 2015.

[14] Ole Baunbæk Jensen. *Webcam-Based Eye Tracking vs. an Eye Tracker*. 2018.

[15] Ankur Joshi, Saket Kale, Satish Chandel, and D K Pal. Article no.BJAST.2015.157 Opinion Article Joshi et al. BJAST(4):157, 2015.

[16] John F. Kurtzke. Rating neurologic impairment in multiple sclerosis. *Neurology*, 33(11):1444–1444, 1983.

[17] Lennart E. Nacke, Sophie Stellmach, Dennis Sasse, and Craig A. Lindley. Gameplay experience in a gaze interaction game. pages 49–54, 2010.

[18] Florin Nanu, Stefan Petrescu, Peter Corcoran, and Petronel Bigioi. Face and gaze tracking as input methods for gaming design. *2011 IEEE International Games Innovation Conference, IGIC 2011*, pages 115–116, 2011.

[19] Joshua Newn, Frank Vetere, Eduardo Velloso, and Marcus Carter. Exploring the Effects of Gaze Awareness on Multiplayer Gameplay.

[20] Marie-Laure Ryan. Narrative as Virtual Reality Immersion and Interactivity in Literature and Electronic Media. Technical report, 2001.

[21] D.W. W Shaffer, K.D. D Kurt Squire, R. Halverson, J.P. P Gee, and Psychology Division. Mark Griffiths The educational benefits of videogames Videogames have great positive potential in. *Phi Delta Kappan*, 20(3):49—-62, 2002.

[22] Shuo Wang, Xiaocao Xiong, Yan Xu, Chao Wang, Weiwei Zhang, Xiaofeng Dai, and Dongmei Zhang. Face-tracking as an augmented input in video games. *Proceedings of the SIGCHI conference on Human Factors in computing systems - CHI '06*, (January 2006):1097, 2006.

[23] Z. Zhang. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10, Feb 2012.

A Project Plan

The plan for this project is described in the following gantt chart (see figure 34).

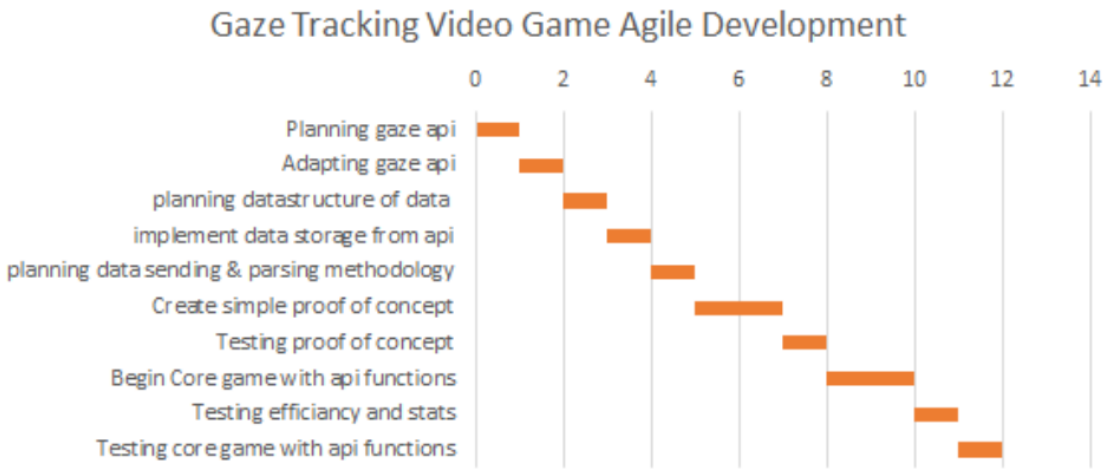


Figure 34: Project Plan Gantt Chart

As you can see from figure 34, A lot of time was spent creating the proof of concept. This however was a crucial step in the development of this software as the core elements needed to be heavily refined to fit into the unity engine before beginning the game.

B Project Diary

Date	Activity	Actions
October 8 2018	Meeting with supervisor	Demonstrated gaze api and discussed project direction.
October 13 2018	Research how to use Js scriptes in Unity.	Make an attempt at a 'hello world' js class called from unity script.
October 15 2018	Meeting with supervisor	Discussion about js and how to use within unity. Get unity talking to js script before beginning prototype
October 16	Began implementing js script in unity	Did not work because unity could not access it before being built. Attempt to find another solution
October 22	Meeting with supervisor	Suggested i research another way around the issue
October 26	Implemented new method of interpreting gaze data in unity	Used Node.js to query the gaze API, stringify it and send it to local server.js, then write the json data to res.json at 40Hz. Shown in console
October 28	Finished literature review	lit review added to thesis paper and draft sent to supervisor
October 29	Meeting with supervisor	Demonstrated new method, confirmed that it will work for my project. supervisor suggested i limit the updates per second to around 20.
November 3	Implemented a handle to control the refresh rate in unity.	Alows me to set the refresh rat between 1 and 40
November 8	Research on how to interperate json data in unity at run time	Found Unity Webhandler class which allows me to query the res.json file on local server.
November 9	Implemented While loop for webhandler query	This loop allows me to read the api data from res.json at the assigned refresh rate
November 11	Meeting with supervisor	Supervisor happy with methods used, suggested some tweaks involving modularizing the gaze getting class. Suggested to implement this in a prototype. discussed calibration.
November 12	Implemented player controller to control ball sprite with head movements	the movements are choppy using Transform.translate. Need to find a smoothing solution
November 19	Meeting with supervisor	Demonstrated movement with ball sprite, supervisor suggested i gamify the scene for the prototype.
November 19	Wrote skeleton thesis	thesis layed out and added some content to chapters to prepare for prototype demo of interim report.
November 20	Implemented calibration scene	User must tilt the ball into the highlighted squares. accuracy recorded and captured in PlayerPrefs class.
November 24	Implemented prototype game scene	Added falling blocks to the ball sprite scene, where the user must dodge the falling blocks using their head movements.
November 25	Implemented movement range controller	Head tilts gave a binary force to the ball sprite(left or right), range controller allows player to use all values in between to move ball
November 26	Meeting with supervisor	Supervisor suggested i implement Lerp class to smooth movement. Discussed player calibration scene to get a reading on accuracy. Suggested some changes to formula used to calculate calibration accuracy.
December 3	Redigned calibration formula	Formula now depicts a more accurate representation of players performance.
December 10	Meeting with supervisor	Draft thesis discussed, some suggestions on what content to add to chapters to be ready for demo.

Date	Activity	Actions
January 30 2019	Meeting with supervisor	Demonstrated first draft of report
January 31 2019	Updated report	Rearranged chapter skeletons
February 6 2019	Meeting with supervisor	Demonstration of Calibration 1
February 8 2019	Implemented pitch movement	Did not fully implement, just displayed floats
February 13 2019	Meeting with supervisor	Suggested i make a prototype 2 with h+v
February 15 2019	Imported tetris mobile game to begin prototype 2	Used unity app store
February 23 2019	Meeting with supervisor	Demonstrated problem with prototype 2
February 24 2019	Implemented V into P2	Modified server.js to pass both values to same JSON file, modified gazegetter to split both values.
March 2 2019	Meeting with supervisor	Suggested calibration 2 and 3
March 5 2019	implemented calibration 2 and 3, hor and vert. Modified accuracy equasion.	Split 2 and 3 into separate scenes, combined results of both.
March 16 2019	Meeting with supervisor	Demonstrated previous work. got advice.
March 17 2019	Began Drive-On	Used unity standard assets car scene to test on
March 19 2019	Created separate controller layer to put in car game. Modified methods.	Modified methods to return discreet values if needed for turns.
March 23 2019	Meeting with supervisor	Demonstrated Drive-On demo 1
March 25 2019	Worked on Drive-On	Optimized gaze getter to poll at 2x the requirement needed
March 29 2019	Wrote up Design of thesis	Modified previous sections to fit current iteration of software.
April 05 2019	Meeting with supervisor	advised on combining features from proto 2 and proto 1.
April 06 2019	Finished Drive-On	implemented score calculation and logging system
April 09 2019	Worked on thesis	Got thesis up to date with current iteration of software.
April 10-April 25	Further developed thesis	Wrote and tested hypotheses
April 25-April 03	Software	combined every feature to prepare for demo

C Code Listings

C.1 index.html

```
<html>
 <body>

  <pre id="edl-data-head"></pre>
  <pre id="edl-data-gaze"></pre>

  <script
    src="https://code.jquery.com/jquery-3.3.1.min.js"
    integrity="sha256-FgpCb/KJQlLNfOu91ta32o/
       NMZxltwRo8QtmkMRdAu8="
    crossorigin="anonymous"></script>
  <script src="./xlabs.js"></script>
  <script type="text/javascript">

  console.log( "Demo");

  function storeData(roll, pitch)
  {
    $.ajax({
      type: "POST",
      url: '/store',
      data: {'roll': roll, 'pitch': pitch},
      dataType: 'application/json'
    });
  }
  function sleep(milliseconds) {
    // console.log("wait");
    var start = new Date().getTime();
    for (var i = 0; i < 1e7; i++) {
    if ((new Date().getTime() - start) > milliseconds){
      break;
    }
  }
  }

  var Demo = {
    update : function() {
      var headroll = xLabs.getConfig( "state.head.roll" );
      var headpitch = xLabs.getConfig( "state.head.pitch" )
         ;
```

```
39        console.log(JSON.stringify({ roll: headroll, pitch:
              headpitch }));
40        storeData(headroll, headpitch);
41        // sleep(100);
42      },
43
44      ready : function() {
45        xLabs.setConfig( "system.mode", "training" );
46        xLabs.setConfig( "browser.canvas.paintHeadPose", "0"
              );
47        window.addEventListener( "beforeunload", function() {
48            xLabs.setConfig( "system.mode", "off" );
49        });
50      }
51    };
52
53
54
55
56    xLabs.setup( Demo.ready, Demo.update, null, "b0708f99
          -1835-40b6-8855-6a4f688b5fb8" );
57
58    </script>
59   </body>
60 </html>
```

C.2 xlabs.js

```
1  window.xLabs = {
2
3
4    config : null,
5    callbackReady : null,
6    callbackState : null,
7    callbackIdPath : null,
8    requestAccessIdx : 0,
9    developerToken : null,
10
11   XLABS_EXTENSION_ID : "licbccoefgmmbgipcgclfgpbicijnlga",
12
13
14   isApiReady : function() {
15     return document.documentElement.getAttribute( 'data-
           xlabs-extension-ready' ) == "1";
16   },
```

```
17
18    getConfig : function( path ) {
19      var value = xLabs.getObjectProperty( xLabs.config, path
          );
20      //console.log( "getConfig( "+path+" = "+ value + " )" )
          ;
21      return value;
22    },
23
24    setConfig : function( path, value ) {
25      window.postMessage( {
26        target: "xLabs",
27        token: xLabs.token, // may be null
28        config: {
29          path: path,
30          value: value
31        }
32      }, "*" );
33    },
34
35
36    getObjectProperty : function( object, path ) {
37      if( ( object == undefined ) || ( object == null ) ) {
38        return "";
39      }
40      //console.log( "Uril util"+path );
41      var parts = path.split('.'),
42          last = parts.pop(),
43          l = parts.length,
44          i = 1,
45          current = parts[ 0 ];
46
47      while( ( object = object[ current ] ) && i < l ) {
48        current = parts[ i ];
49        //console.log( "Util object: "+JSON.stringify( object
              ) );
50        i++;
51      }
52
53      if( object ) {
54        //console.log( "Util result: "+object[ last ] );
55        return object[ last ];
56      }
57    },
58
```

```
59    t1 : 0,
60
61    resetCalibrationTruth : function() {
62      xLabs.t1 = 0;
63    },
64
65    updateCalibrationTruth : function( xScreen, yScreen ) {
66      var t1 = xLabs.t1;
67      var t2 = xLabs.getTimestamp();
68
69      if( t1 <= 0 ) { // none set
70        t1 = t2;
71        t2 = t2 +1; // ensure duration at least 1 and
              positive
72      }
73
74      xLabs.addCalibrationTruth( t1, t2, xScreen, yScreen );
75
76      xLabs.t1 = t2; // change the timestamp
77    },
78
79    addCalibrationTruth : function( t1, t2, xScreen, yScreen
         ) {
80      // Defines ordering of values
81      // t1,t2,xs,ys
82      // For truth, also used for clicks
83      var csv = t1 + "," + t2 + "," + parseInt( xScreen  ) +
           "," + parseInt( yScreen );
84      //console.log( "xLabs truth: "+csv );
85      xLabs.setConfig( "truth.append", csv );
86    },
87
88    calibrate : function( id ) {
89      var request = "3p";
90      if( id ) {
91        request = id;
92      }
93
94      xLabs.setConfig( "calibration.request", request );
95      console.log( "xLabs: Calibrating..." );
96    },
97
98    calibrationClear : function() {
99      xLabs.setConfig( "calibration.clear", request );
100     console.log( "xLabs: Clearing calibration..." );
```

```
101   },
102
103
104   getTimestamp : function() {
105     // unified function to get suitable timestamps
106     var dateTime = new Date();
107     var timestamp = dateTime.getTime();
108     return timestamp;
109   },
110
111
112   getDpi : function() {
113     var dppx = window.devicePixelRatio ||
114       (     window.matchMedia
115         && window.matchMedia( "(min-resolution: 2dppx), (-
              webkit-min-device-pixel-ratio: 1.5),(-moz-min-
              device-pixel-ratio: 1.5),(min-device-pixel-ratio
              : 1.5)" ).matches? 2 : 1 )
116       || 1;
117
118     var w = ( screen.width  * dppx );
119     var h = ( screen.height * dppx );
120     return this.calcDpi( w, h, 13.3, 'd' );
121   },
122
123   calcDpi : function( w, h, d, opt ) {
124     // Calculate PPI/DPI
125     // Source: http://dpi.lv/
126     w>0 || (w=1);
127     h>0 || (h=1);
128     opt || (opt='d');
129     var dpi = (opt=='d' ? Math.sqrt(w*w + h*h) : opt=='w' ?
            w : h) / d;
130     return dpi>0 ? Math.round(dpi) : 0;
131   },
132
133
134   devicePixelRatio : function() {
135     var ratio = parseFloat( xLabs.getConfig("browser.screen
          .devicePixelRatioWithoutZoom") );
136     if( !ratio ) {
137       return null
138     }
139     var ratio = parseInt( ratio );
140     if( ratio === 0 ) {
```

```
141       return null;
142     }
143     return window.devicePixelRatio / ratio;
144   },
145
146
147   documentOffset : function() {
148     if( !xLabs.documentOffsetReady() ) {
149       throw "xLabs: Should not call scr2doc() unless mouse
              moved, i.e. browser.document.offset.ready == 1";
150     }
151     var x = parseInt( xLabs.getConfig( "browser.document.
          offset.x" ) );
152     var y = parseInt( xLabs.getConfig( "browser.document.
          offset.y" ) );
153     return { x: x, y: y };
154   },
155
156   documentOffsetReady : function() {
157     var ready = xLabs.getConfig( "browser.document.offset.
          ready" );
158     if( ready.localeCompare( "1" ) != 0 ) {
159       return false;
160     }
161     return true;
162   },
163
164   scr2docX: function( screenX ) {
165     if( !xLabs.documentOffsetReady() ) {
166       throw "xLabs: Should not call scr2doc() unless mouse
              moved, i.e. browser.document.offset.ready == 1";
167     }
168
169     var xOffset = parseInt(xLabs.getConfig( "browser.
          document.offset.x" ));
170     return screenX - window.screenX - xOffset;
171   },
172
173   scr2docY: function( screenY ) {
174     if( !xLabs.documentOffsetReady() ) {
175       throw "xLabs: Should not call scr2doc() unless mouse
              moved, i.e. browser.document.offset.ready == 1";
176     }
177
178     var yOffset = parseInt(xLabs.getConfig( "browser.
```

```
        document.offset.y" ));
    return screenY - window.screenY - yOffset;
  },

  scr2doc: function( screenX, screenY ) {
    return {
      x: xLabs.scr2docX( screenX ),
      y: xLabs.scr2docY( screenY )
    }
  },

  doc2scrX: function( documentX ) {
    if( !xLabs.documentOffsetReady() ) {
      throw "xLabs: Should not call scr2doc() unless mouse
          moved, i.e. browser.document.offset.ready == 1";
    }
    var xOffset = parseInt(xLabs.getConfig( "browser.
        document.offset.x" ));
    return documentX + window.screenX + xOffset;
  },

  doc2scrY: function( documentY ) {
    if( !xLabs.documentOffsetReady() ) {
      throw "xLabs: Should not call scr2doc() unless mouse
          moved, i.e. browser.document.offset.ready == 1";
    }
    var yOffset = parseInt(xLabs.getConfig( "browser.
        document.offset.y" ));
    return documentY + window.screenY + yOffset;
  },

  doc2scr: function( documentX, documentY ) {
    return {
      x: xLabs.doc2scrX( documentX ),
      y: xLabs.doc2scrY( documentY )
    }
  },

  onApiReady : function() {

    xLabs.pageCheck()

    if( xLabs.callbackReady != null ) {
      xLabs.callbackReady();
    }
```

```
219   },
220
221   onApiState : function( config ) {
222     xLabs.config = config;
223     if( xLabs.callbackState != null ) {
224       xLabs.callbackState();
225     }
226   },
227
228   onApiIdPath : function( detail ) {
229     if( xLabs.callbackIdPath != null ) {
230       xLabs.callbackIdPath( detail.id, detail.path );
231     }
232   },
233
234   // Returns the version number of the extension, or null
        if extension not installed.
235   extensionVersion : function() {
236     return document.documentElement.getAttribute('data-
          xlabs-extension-version');
237   },
238
239   hasExtension : function() {
240     return document.documentElement.getAttribute('data-
          xlabs-extension') ||
241       document.documentElement.getAttribute('data-xlabs-
            extension-version') // to be compatible with <
            2.5.2
242   },
243
244   isExtension : function() {
245     return chrome.runtime.getManifest !== undefined
246   },
247
248   pageCheck : function() {
249     console.log( "xlabs.js pageCheck() called")
250     var message = {
251       target : "xLabs",
252       action: "request-access",
253       token: xLabs.developerToken // may be null or
            undefined
254     };
255
256     // An extension is asking for permission, send a
          message directly to the background script.
```

```
257     if( xLabs.isExtension() ) {
258       chrome.runtime.sendMessage( xLabs.XLABS_EXTENSION_ID,
              message );
259       console.log( "send message to xlabs background script
              with extension ID");
260     }
261     // From the a website
262     else {
263       window.postMessage( message, "*" );
264       console.log( message );
265       console.log( "posting message to content script.");
266     }
267   },
268
269   setup : function( callbackReady, callbackState,
         callbackIdPath, developerToken ) {
270     if( !xLabs.hasExtension() ) {
271       alert("xLabs chrome extension is not installed");
272       return;
273     }
274
275     xLabs.callbackReady = callbackReady;
276     xLabs.callbackState = callbackState;
277     xLabs.callbackIdPath = callbackIdPath;
278
279     if( developerToken ) {
280       xLabs.developerToken = developerToken
281     }
282
283     // If the API is already setup, then we can call the
          callback without needing
284     // to listen to the ready event. But since it's meant
          to be a callback we
285     // shall defer calling the callback in the event loop,
          which would be the expectation
286     // when registering callbacks.
287     if( xLabs.isApiReady() ) {
288       setTimeout( function() {
289         xLabs.onApiReady();
290       }, 0 );
291     }
292     // Not ready yet, we can wait for the event.
293     else {
294       // add event listeners
295       document.addEventListener( "xLabsApiReady", function
```

```
            () {
296           xLabs.onApiReady();
297         })
298       }
299
300       document.addEventListener( "xLabsApiState", function(
              event ) {
301         xLabs.onApiState( event.detail );
302       })
303
304       document.addEventListener( "xLabsApiIdPath", function(
              event ) {
305         xLabs.onApiIdPath( event.detail );
306       })
307     }
308
309 };
310
311
312
313 // Usage: xLabs.setup( myCallbackFnReady, myCallbackFnState
        );
```

C.3 index.js

```
1  const express = require('express')
2  const app = express()
3  var fs = require("fs");
4
5  app.get('/', (req, res) => res.sendFile(__dirname + '/
       public/index.html'));
6  app.use(express.static('public'));
7  app.use(express.json());
8  app.use(express.urlencoded());
9  app.listen(8080, () => console.log('listening on port
       8080!'));
10 // fs.writeFile("./sample.txt", "hi", (err) => {
11 //     if (err) {
12 //         console.error(err);
13 //         return;
14 //     };
15 //     console.log("File has been created");
16 // });
17
18 // POST method route
```

```
app.post('/store', function (req, res) {
  res.send('POST request to the homepage');
  console.log(JSON.stringify(req.body));
  fs.writeFile("./res.json", JSON.stringify(req.body), (err
      ) => {
      if (err) {
          console.error(err);
          return;
      };
      console.log("File has been created");
 });
})

app.get('/fetchData', function(req, res) {
  res.send(fs.readFileSync('./res.json'));
});
```

C.4 GazeController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Networking;
using System.Threading;


public class GazeController : MonoBehaviour
{
    [Range(0, 1)] [SerializeField] public float
       rollTiltLeftThreshold = 0.1f;
    [Range(0, 1)] [SerializeField] public float
       rollTiltRightThreshold = 0.1f;
    [Range(0, 1)] [SerializeField] public float
       pitchtiltUpThreshold = 0.55f;
    [Range(0, 1)] [SerializeField] public float
       pitchtiltDownThreshold = 0.7f;


    public float roll;
    public float pitch;
    public bool left, right, up, down = false;

    public GazeControllerData gazeDataObject = new
       GazeControllerData();
```

```
22      void Start()
23      {
24          //gazeDataObject.head = -1;
25          gazeDataObject.roll = -1;
26          gazeDataObject.pitch = -1;
27          StartCoroutine(GetText());
28
29      }
30
31      public IEnumerator GetText()
32      {
33          while (true)
34          {
35              //yield return new WaitForSeconds(0.5f);
36              using (UnityWebRequest www = UnityWebRequest.
                    Get("http://localhost:8080/fetchData"))
37              {
38                  yield return www.SendWebRequest();
39
40                  if (www.isNetworkError || www.isHttpError)
41                  {
42                      //Debug.Log(www.error);
43                  }
44                  else
45                  {
46                      string jsonDataReceived = www.
                            downloadHandler.text;
47                      int pos = jsonDataReceived.IndexOf("}")
                            ;
48                      jsonDataReceived = jsonDataReceived.
                            Substring(0, pos+1);
49                      //jsonDataReceived = jsonDataReceived.
50
51                      GazeControllerData gazeDataObject2 =
                            new GazeControllerData();
52                      try
53                      {
54                          gazeDataObject2 = JsonUtility.
                                FromJson<GazeControllerData>(
                                jsonDataReceived);
55                      } catch(System.Exception e)
56                      {
57                          Debug.LogError(jsonDataReceived);
58                      }
59
```

```
60                      //if empty obj, use previous
61                      if (null != gazeDataObject2)
62                      {
63                          gazeDataObject = gazeDataObject2;
64                      }
65
66                      this.roll = gazeDataObject.roll;
67                      this.roll = gazeDataObject.pitch;
68
69                      left = this.IsTiltingLeft();
70                      right = this.IsTiltingRight();
71                      up = this.IsTiltingUp();
72                      down = this.IsTiltingDown();
73                  }
74              }
75          }
76      }
77
78
79
80      public bool IsTiltingLeft()
81      {
82          return this.gazeDataObject.roll >
                rollTiltLeftThreshold;
83      }
84
85      public bool IsTiltingRight()
86      {
87          return this.gazeDataObject.roll < -
                rollTiltRightThreshold;
88      }
89
90      public bool IsTiltingUp()
91      {
92          return this.gazeDataObject.pitch <
                pitchtiltUpThreshold;
93
94      }
95
96      public bool IsTiltingDown()
97      {
98          return this.gazeDataObject.pitch >
                pitchtiltDownThreshold;
99      }
100
```

```
101     public float GetRollValue()
102     {
103         //if (this.gazeDataObject.roll >
                rollTiltLeftThreshold && this.gazeDataObject.
                roll < rollTiltRightThreshold)
104         if(IsTiltingLeft() || IsTiltingRight())
105             return this.gazeDataObject.roll;
106         return 0;
107     }
108
109     public float GetPitchValue()
110     {
111         //if (IsTiltingDown() || IsTiltingUp())
112             return this.gazeDataObject.pitch;
113         //return 0;
114     }
115
116     public float getRollValueForAxis()
117     {
118         if(this.IsTiltingLeft())
119         {
120             return -1.0f;
121         }
122         if(IsTiltingRight())
123         {
124             return 1.0f;
125         }
126         return 0.0f;
127     }
128
129     public float getPitchValueForAxis()
130     {
131         if (this.IsTiltingUp())
132         {
133             return 1.0f;
134         }
135         if (IsTiltingDown())
136         {
137             return -1.0f;
138         }
139         return 0.0f;
140     }
141
142     public float getRollValueForCarAxis()
143     {
```

```
144         if (this.IsTiltingLeft())
145         {
146             return -1.0f;
147         }
148         if (IsTiltingRight())
149         {
150             return 1.0f;
151         }
152         return 0.0f;
153     }
154 
155 
156     public float getPitchValueForCarAxis()
157     {
158         if (IsTiltingDown())
159         {
160             return -1.0f;
161         }
162         return 1.0f;
163     }
164 
165 }
```

C.5 GazeControllerData.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  
5  [System.Serializable]
6  public class GazeControllerData
7  {
8  
9      //public float head;
10     public float roll;
11     public float pitch;
12 
13     public static GazeControllerData FromJSON(string
           jsonString)
14     {
15         return JsonUtility.FromJson<GazeControllerData>(
               jsonString);
16     }
17 
18     public override string ToString()
```

```
19      {
20          //return "head = " + head;
21          return "roll = " + roll;
22      }
23
24
25  }
```

C.6 AccuracyCalculator.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class AccuracyCalculator {
6
7      //private float accuracyRatio;
8      //private float timer;
9      //private float allowanceModifier;
10     //private float lifespan;
11
12     //private float lifespan;
13     private float totalTime;
14     private float allowanceMod;
15     public float timeInSquare;
16
17     public AccuracyCalculator(float lifespan, float timer,
           float timeInSquare)
18     {
19         this.totalTime = timer;
20         //this.lifespan = lifespan;
21         this.timeInSquare = timeInSquare;
22         this.allowanceMod = 0.75f;
23
24         //this.timer = timer;
25         //this.lifespan = lifespan;
26         //allowanceModifier = timer * 1f;
27         //accuracyRatio = 0f;
28         //accuracyRatio = (timer + allowanceModifier) /
               lifespan;
29         //accuracyRatio /= 100;
30     }
31
32     public float Accuracy()
33     {
```

```
34          return Mathf.Ceil(100 * (timeInSquare / (
                allowanceMod * totalTime)));
35          //return Mathf.Ceil(accuracyRatio * lifespan * 10);
36      }
37  }
```

C.7 CarUserControl.cs

```
1  using System;
2  using UnityEngine;
3  using UnityStandardAssets.CrossPlatformInput;
4  using UnityEngine.UI;
5  using UnityEngine.SceneManagement;
6
7  namespace UnityStandardAssets.Vehicles.Car
8  {
9      [RequireComponent(typeof (CarController))]
10     public class CarUserControl : MonoBehaviour
11     {
12         private CarController m_Car; // the car controller
               we want to use
13         public GazeController controller;
14         public Slider hslider, vslider;
15         public float hMod, vMod;
16         private int mode = -1;
17         public Text modeText;
18         private bool isKeyPressed = false;
19
20         private void Awake()
21         {
22             if (SceneManager.GetActiveScene() ==
                   SceneManager.GetSceneByName("
                   Car_Level_1_Easy") || SceneManager.
                   GetActiveScene() == SceneManager.
                   GetSceneByName("Car_Level_1_Normal"))
23             {
24                 hMod = 0.2f;
25                 vMod = 0.33f;
26             }
27             else
28             {
29                 hMod = 0.2f;
30                 vMod = 0.33f;
31
32             }
```

```
33
34              // get the car controller
35              m_Car = GetComponent<CarController>();
36              controller = FindObjectOfType<GazeController>()
                    ;
37              //hslider.value = 0.0f;
38              vslider.value = 0.6f;
39
40          }
41
42
43          private void FixedUpdate()
44          {
45              //pass the input to the car!
46
47              if(Input.GetKeyDown(KeyCode.C))
48              {
49                  mode = -mode;
50                  Debug.Log(mode);
51              }
52
53              if(mode == -1)
54              {
55                  modeText.text = "Steering Mode: Head";
56                  float h = controller.getRollValueForAxis()
                        * hMod;
57                  hslider.value = Mathf.Lerp(hslider.value, -
                        controller.GetRollValue(), 0.1f);
58                  float v = controller.
                        getPitchValueForCarAxis() * vMod;
59                  //vslider.value = Mathf.Lerp(vslider.value,
                         controller.GetPitchValue(), 0.1f);
60                  float handbrake = CrossPlatformInputManager
                        .GetAxis("Jump");
61
62                  m_Car.Move(h, v, v, handbrake);
63
64              }
65              else if (mode == 1){
66                  modeText.text = "Steering Mode: Keyboard";
67                  float h = CrossPlatformInputManager.GetAxis
                        ("Horizontal") * hMod;
68                  hslider.value = Mathf.Lerp(hslider.value, h
                        , 0.1f);
69
```

```
70                  float v = CrossPlatformInputManager.GetAxis
                        ("Vertical") * vMod;
71
72                  float handbrake = CrossPlatformInputManager
                        .GetAxis("Jump");
73
74                  m_Car.Move(h, v, v, handbrake);
75
76              }
77              else
78              {
79                  float h = 0;
80                  float v = 0;
81                  float handbrake = CrossPlatformInputManager
                        .GetAxis("Jump");
82
83                  m_Car.Move(h, v, v, handbrake);
84
85              }
86
87              //float h = CrossPlatformInputManager.GetAxis("
                    Horizontal");
88              //float v = CrossPlatformInputManager.GetAxis("
                    Vertical");
89
90              //float h = controller.getRollValueForAxis() *
                    0.2f;
91              //hslider.value = Mathf.Lerp(hslider.value, -
                    controller.GetRollValue(), 0.1f);
92              //float v = controller.getPitchValueForCarAxis
                    () * 0.3f;
93              //vslider.value = Mathf.Lerp(vslider.value,
                    controller.GetPitchValue(), 0.1f);
94  #if !MOBILE_INPUT
95              //float handbrake = CrossPlatformInputManager.
                    GetAxis("Jump");
96  #else
97              //m_Car.Move(h, v, v, 0f);
98  #endif
99          }
100
101         public void SetHModifier(float mod)
102         {
103             this.hMod = mod;
104         }
```

```
105             public void SetVModifier(float mod)
106             {
107                 this.vMod = mod;
108             }
109         }
110 
111 
112 
113 }
```

D System Information

Operating System: Windows 10 Pro 64-bit (10.0, Build 17134) (17134.rs4_release.180410-1804)
BIOS: BIOS Date: 12/23/10 19:51:38 Ver: 24.03 (type: BIOS)
Processor: AMD Phenom(tm) II X4 960T Processor (4 CPUs), 3.0GHz
Memory: 8192MB RAM
Available OS Memory: 8190MB RAM
Card name: AMD Radeon R9 200 Series