Technical University of Crete
School of Electrical and Computer Engineering

Diploma Thesis

Multimodal User Interface for Autonomous Driving in Augmented Reality simulated in Virtual Reality

Georgios Protopapadakis

Thesis Committee
Professor Katerina Mania
Professor Michail G. Lagoudakis
Professor Antonios Deligiannakis

Chania | Crete, October 2024

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολη Ηλεκτρολογων Μηχανικων & Μηχανικων Υπολογιστων

Διπλωματική Εργασία

Πολυτροπική διεπαφή χρήστη για αυτόνομη οδήγηση σε επαυξημένη πραγματικότητα προσομοιωμένη σε εικονική πραγματικότητα

Γεώργιος Πρωτοπαπαδάκης

Εξεταστική επιτροπή

Καθηγήτρια Αικατερίνη Μανιά

Καθηγητής Μιχαήλ Γ. Λαγουδάκης

Καθηγητής Αντώνιος Δεληγιαννάκης

Χανιά | Κρήτη, Οκτώβριος 2024

Abstract

As autonomous vehicles become more common in daily life, developing effective and intuitive User Interfaces (UIs) for passenger interaction is becoming increasingly important. Multimodal interaction techniques, like eye tracking and voice commands, seem like promising solutions by providing hands-free and intuitive control methods. This thesis introduces a VR-based simulator designed to explore these interactions on an Augmented Reality (AR) windshield display (WSD) within a mobile office setting in an autonomously driven vehicle. The system combines gaze-based interaction techniques, such as eye blink and dwell-time, as well as voice commands, to offer a hands-free interface for controlling vehicle functions, such as air conditioning and radio, as well as office tasks, like phone calls and messaging.

The system uses the HTC Vive Pro Eye with SRanipal SDK for real-time eye tracking, together with TobiiXR SDK for advanced gaze-based interactions. Built using the Unity game engine, the simulator enables the users to experience an autonomous vehicle, while navigating an urban environment. Unity's Speech and Dictation Recognizers also help to implement voice command interactions and speech to text writing. This system's capabilities were evaluated based on user studies, measuring task completion times, errors throughout specific tasks, and user preferences across different levels of VR familiarity. Additionally, the system explores the concept of mobile office, where users can stay productive, while the vehicle autonomously navigates.

The detailed performance evaluation includes metrics, such as interaction responsiveness and error rates for each eye-gaze modality. Usability Testing and User Experience Metrics evaluation methods were used with 12 participants, providing feedback on the effectiveness of the interaction techniques. The results show a preference for eye blink and voice command interactions for speed, though dwell-time demonstrated potential for improved accuracy with training. The study also emphasizes the importance of user trust in autonomous vehicles, enhanced by real-time feedback from the vehicle. This work contributes to the evolving field of AR-driven user interfaces in autonomous driving, offering insights into future interface design and user interaction techniques for highly automated systems.

Περίληψη

Καθώς τα αυτόνομα οχήματα γίνονται όλο και πιο συνηθισμένα στην καθημερινή ζωή, η ανάπτυξη αποτελεσματικών και διαισθητικών Διεπαφών Χρήστη (UIs) για την αλληλεπίδραση των επιβατών γίνεται όλο και πιο σημαντική. Τεχνικές πολυτροπικής αλληλεπίδρασης, όπως ο εντοπισμός ματιών και οι φωνητικές εντολές, φαίνονται να προσφέρουν ελπιδοφόρες λύσεις παρέχοντας διαισθητικές μεθόδους ελέγχου χωρίς τη χρήση χεριών. Αυτή η διπλωματική εργασία παρουσιάζει έναν προσομοιωτή βασισμένο στην Εικονική Πραγματικότητα (VR), σχεδιασμένο για την εξερεύνηση αυτών των αλληλεπιδράσεων σε μια Οθόνη Επαυξημένης Πραγματικότητας (AR) στο παρμπρίζ, μέσα σε ένα περιβάλλον κινητού γραφείου σε αυτόνομα κινούμενο όχημα. Το σύστημα συνδυάζει τεχνικές αλληλεπίδρασης με τη χρήση του βλέμματος, όπως το κλείσιμο των ματιών και ο χρόνος παρατεταμένου βλέμματος, καθώς και φωνητικές εντολές, για να προσφέρει μια διεπαφή χωρίς τη χρήση χεριών για τον έλεγχο λειτουργιών του οχήματος, όπως το κλιματιστικό και το ραδιόφωνο, καθώς και εργασίες γραφείου, όπως οι τηλεφωνικές κλήσεις και η αποστολή μηνυμάτων.

Το σύστημα χρησιμοποιεί την συσκευή απεικόνισης HTC Vive Pro Eye με τις βιβλιοθήκες SRanipal SDK για την παρακολούθηση των ματιών σε πραγματικό χρόνο, σε συνδυασμό με το TobiiXR SDK για προηγμένες αλληλεπιδράσεις που βασίζονται στο βλέμμα. Κατασκευασμένο με τη μηχανή παιχνιδιών Unity, ο προσομοιωτής επιτρέπει στους χρήστες να βιώσουν την εμπειρία ενός αυτόνομου οχήματος που κινείται σε ένα αστικό περιβάλλον. Οι Αναγνωριστές Ομιλίας και Υπαγόρευσης της Unity χρησιμοποιούνται επίσης για την υλοποίηση φωνητικών εντολών και τη μετατροπή ομιλίας σε κείμενο. Οι δυνατότητες αυτού του συστήματος αξιολογήθηκαν με βάση μελέτες χρηστών, μετρώτας τους χρόνους ολοκλήρωσης εργασιών, τα λάθη κατά τη διάρκεια συγκεκριμένων εργασιών και τις προτιμήσεις των χρηστών σε διάφορα επίπεδα εξοικείωσης με την τεχνολογία Εικονικής Πραγματικότητας. Επιπλέον, το σύστημα εξερευνά την έννοια του κινητού γραφείου, όπου οι χρήστες μπορούν να παραμένουν παραγωγικοί ενώ το όχημα κινείται αυτόνομα.

Η λεπτομερής αξιολόγηση απόδοσης επικεντρώνεται σε μετρήσεις αλληλεπίδρασης χρήστη, συμπεριλαμβανομένης της απόκρισης και των ποσοστών σφάλματος για κάθε μέθοδο αλληλεπίδρασης μέσω του βλέμματος. Μεθοδολογίες Αξιολόγησης Χρηστικότητας και Μετρήσεις Εμπειρίας Χρήστη χρησιμοποιήθηκαν με τη συμμετοχή 12 ατόμων, προσφέροντας ανατροφοδότηση για την αποτελεσματικότητα των τεχνικών αλληλεπίδρασης. Τα αποτελέσματα δείχνουν προτίμηση για τις αλληλεπιδράσεις με το κλείσιμο των ματιών και τις φωνητικές

εντολές ως προς την ταχύτητα, ενώ ο παρατεταμένος χρόνος του βλέμματος έδειξε προοπτική για βελτίωση της ακρίβειας με περαιτέρω εκπαίδευση. Η μελέτη τονίζει επίσης τη σημασία της εμπιστοσύνης των χρηστών στα αυτόνομα οχήματα, που ενισχύεται από την ανατροφοδότηση του χρήστη σε πραγματικό χρόνο από το όχημα. Η εργασία αυτή συμβάλλει στο εξελισσόμενο πεδίο των διεπαφών χρήστη που βασίζονται στην Επαυξημένη Πραγματικότητα στα αυτόνομα οχήματα, προσφέροντας πληροφορίες για τον σχεδιασμό διεπαφών και τεχνικών αλληλεπίδρασης για συστήματα υψηλής αυτοματοποίησης.

Acknowledgements

Initially, I would like to thank my supervisor professor Katerina Mania for her guidance and support throughout the whole development of this thesis.

Next, I want to thank the members of Surreal team, for providing support and feedback in any aspect of the process I needed them, and for the wonderful atmosphere inside the laboratory.

Additionally, I want to specially thank my beloved friends Anna, Michalis, Konstantina and Christos for their effort and support throughout the implementation of this thesis, and their love and unforgettable memories we built together throughout all these university years.

Finally, I want to thank my family for their endless support and encouragement throughout this journey.

Contents

List of Figures

Chapter 1

Introduction

1.1 Brief Description

In recent years, Virtual Reality (VR) has gained increased popularity due to the affordable prices of Head Mounted Displays (HMDs) in the consumer market. VR is no longer a technology strictly used in science laboratories, making it possible for every user to experience the immersion it can offer. Virtual Reality driving simulators are used effectively for vehicle system development, human factor study, and other purposes. This is possible by enabling to reproduce actual driving conditions in a safe and tightly controlled environment. VR technology has been widely used in engineering and scientific visualization due to its ability to create a life like simulation which is needed for complex systems like Autonomous Vehicles (AVs).

A self driving car is a vehicle that is capable of sensing its environment and move safely with little to no human input. Self driving vehicles include a variety of sensors to comprehend their surrounding environment, such as radar, LiDAR, sonar, GPS, thermographic cameras, odometry, and inertial measurement units. Compared with human-driven (conventional) vehicles, AVs offer many advantages such as reduction in the number of vehicle crashes by eliminating human error, increased mobility of disabled and elderly people, improved traffic flow and fuel efficiency and increased productivity of travel time by allowing the driver to engage in other activities. However, the interaction between humans and autonomous systems remains a critical challenge. The development of intuitive and efficient user interfaces (UIs) is crucial to ensure that users can effectively communicate with and control autonomous vehicles.

Augmented Reality (AR) displays is a contemporary way of displaying information about the car and interact with it. AR aims at simplifying the user's life by bringing virtual information not only to his immediate surroundings, but also to any indirect view of the real world environment and enhances the perception and interaction of users with the real world. By integrating VR and AR technologies, we can design and evaluate advanced user interfaces in a controlled yet immersive environment. Automated driving (AD) further increases drivers' desire for non driving related tasks (NDRTs). When it comes to AR windshield displays (WSDs) about cars, the driver should have the less possible distraction. Touching selection conducts to a higher distraction for drivers, therefore gaze-based interaction is a suggestion.

Figure 1.1: Urban Environment

This thesis explores the development and implementation of a multimodal AR User Interface for autonomous driving cars, within a VR simulated environment. The core of this project is a simulator built in Unity game engine, featuring a virtual reality environment where an autonomous driver navigates the roads of a city. The initial environment

1.1 was sourced from GitHub and adapted from a desktop application to a VR version, with modifications to suit the new platform. The user interface, designed for AR, allows users to interact with the autonomous vehicle using eye tracking and voice commands, providing hands free and intuitive interactions. More specifically, the UI can be used for simple car functions such as radio and A/C as well as for simple office work like sending and reading messages and voice calling. Additionally, the interface informs users of the autonomous driver's actions through warning notifications in order to amplify user's trust towards the autonomous vehicle.

Figure 1.2: User's View in Main Window of the UI

The simulator 1.2 was created with the use of the Unity game engine offering the programming environment for the development of 3D computer graphics applications and the associated scripts were written in C#. The challenge of this thesis was the technical implementation of the VR itself as well as the creation of a user friendly and intuitive experience. Gaze-based interaction often lacks intuitiveness for every-day users, thus extensive research is necessary to refine its accuracy and responsiveness. To achieve a user friendly outcome and an easy to handle UI, it was essential to conduct studies on

user behavior, interface design principles, and the integration of multimodal inputs to ensure an effective and natural interaction experience. The final VR simulator presented in this thesis, can be experienced with the use of HTC Vive Pro head mounted display (HMD).

The implementation of this project integrates various advanced tools and technologies for effective and intuitive interaction within the VR environment. For eye tracking, the HTC Vive Pro headset was utilized, supported by the SRanipal SDK for real-time data on eye movements, which enables the implementation of gaze-based selection techniques such as eye blink and eye dwell. Additionally, the TobiiXR SDK was used to enhance gaze tracking accuracy and interaction processing, as well as the libraries to implement interaction functionalities. For voice commands, Unity's Speech and Dictation Recognizers were integrated, allowing users to control the system using predefined verbal commands and text to speech writing respectively, offering an intuitive, hands-free experience. This integration ensures that both eye tracking and voice recognition work in parallel, creating a, multimodal user experience in the VR environment.

After the end of the implementation process, the evaluation of the project took place with the participation of 12 people with difference experience in VR. The evaluation process includes a brief training of the users in a test scene to get used to the UI layout, functionalities and interaction techniques, while getting familiar with the experimental procedure. The experiment included a metric evaluation (errors and task completion times) about 3 specific tasks with different difficulty, for each of the eye gaze selection techniques while the AV is navigating around the city environment. After that, users continued cruising around the city while discovering the rest of the UI functionalities.

Moreover, after the ending of the simulation, the users were given questionnaires to get their feedback about their experience. Analyzing this feedback, we got results about the effectiveness of the interaction techniques used in the interface and user preferences. Also, using this feedback, there was an analysis about the trust they gained towards the AV and their thoughts on the mobile office concept based on their experience during the simulation.

1.2 Purpose of the Thesis

Technology may evolve rapidly and changes it's course on a frequent basis, but the main objective remains the same, to make human life easier. Due to the rapid evolution of AVs that have already entered the market for the past few years, the commercial use will become affordable in the near future. The aim of this thesis is to show that transportation time inside an AV can also be productive while maintaining the user's road awareness and nurture the trust between users and AVs at the same time. The primary objectives of this research are to:

- Develop a comprehensive VR environment that accurately simulates urban driving conditions for an autonomous vehicle.
- Design and implement an AR-based user interface that enables intuitive interaction through eye tracking and voice commands.
- Research gazeable layouts to avoid collision between gaze-focusable components for better interaction accuracy and less user fatigue.
- Implement non driving related tasks for simple office-work functions.
- Evaluate the usability and effectiveness of the AR UI in enhancing user interaction with the functions of the interface.

The thesis aims to prove that gaze-based interaction can become a tool for everyday use as the AD is cruising the user to the preferred destination. As the safety of the user is and always will be the highest priority, working with a transparent WSD interface can guarantee for the user's road awareness. By integrating AR within a VR-simulated environment, this research aims to develop the trust between users and autonomous vehicles, offering insights into the design of future user interfaces that enhance safety, trust, and user experience in autonomous driving contexts.

1.3 Structure of the Thesis

Chapter 1 is a brief introduction to Virtual Reality simulators, Augmented Reality User Interfaces, Autonomous Drivers and a presentation of the developed simulator of this thesis. Also, there is a brief demonstration of our evaluation process and the direction of our results.

Chapter 2 describes the research that was conducted in order to better understand the state of the art of the technologies used throughout this thesis. The chapter starts with the history of VR along with an explanation why VR is immersive and interactive, followed by an analysis on state of the Head Mounted Displays (HDMs) and why we chose this specific device. Then, a brief history of AR applications is showcased along with an analysis about interaction techniques in AR, concluding to which of them are used in this project and why. Afterwards, we delve into Autonomous Driving System technologies, interaction between users and vehicles, and the benefits of VR simulation and testing in Autonomous Vehicles. In the area of UI design, we explain the key factors to implementing an effective Interface to be used in AVs, and the ways of evaluating them. Finally, we explain the importance of improving User Experience in Immersive technologies and the impact of different evaluation approaches.

Chapter 3 explains the technological requirements used in this thesis. It starts with a closer look to Unity game engine and its components used for implementing various project functionalities. Then, Unity's built in Speech Recognition and Web Requests are explained, as well as software needed for integrating HTC Vive Pro into the project. Tobii XR SDK, the eye tracking software used, and the research about its interaction with the HMD is reported.

Chapter 4 delves into user's experience navigating through the application and the available interactions withing the UI. Both starting menus and UIs layout are explained, while also providing use case diagrams about UI interactions for better visual comprehension.

Chapter 5 explains the technical implementation of this thesis, using information and tools that have been explained earlier. The simulation environment, including scene usage and loading, and desktop to VR integration are explained. Then, the functionality of the AV and the integration of eye gaze and speech interactions using Tobii XR SDK and Unity's Speech and Dictation Recognizer accordingly are explained. The implementation

of the User Interface functionalities for each of the UI windows and their interconnection, as well as the use of Audio and Unity Events is also presented.

Chapter 6 provides the explanation of the evaluation process, the train of thought behind designing it, and the metric and UX results as well as their combined analysis.

Chapter 7 presents the conclusions of our multimodal UI VR simulator. It concludes in the results of our work about the effectiveness of the chosen interactions and the advancement of the future mobile office concept. Last, it refers to limitations deduced during the implementation and evaluation of our work, and provides suggestions about the future development of this or similar WSD Interfaces.

Chapter 2

Research Overview

2.1 Introduction

The development of effective and intuitive user interfaces for autonomous vehicles (AVs) is a challenge that requires an understanding of various intersecting domains. At the beginning of this thesis, research on various topics was conducted. The core topics include the medium, which is Virtual Reality (VR), Augmented Reality (AR), Autonomous Driving (AD), User Interface (UI) design and multimodal interaction techniques. We also present a research on the state of the art Head Mounted Displays (HMDs) available along with a brief history on them and their key characteristics that led upon choosing the right one.

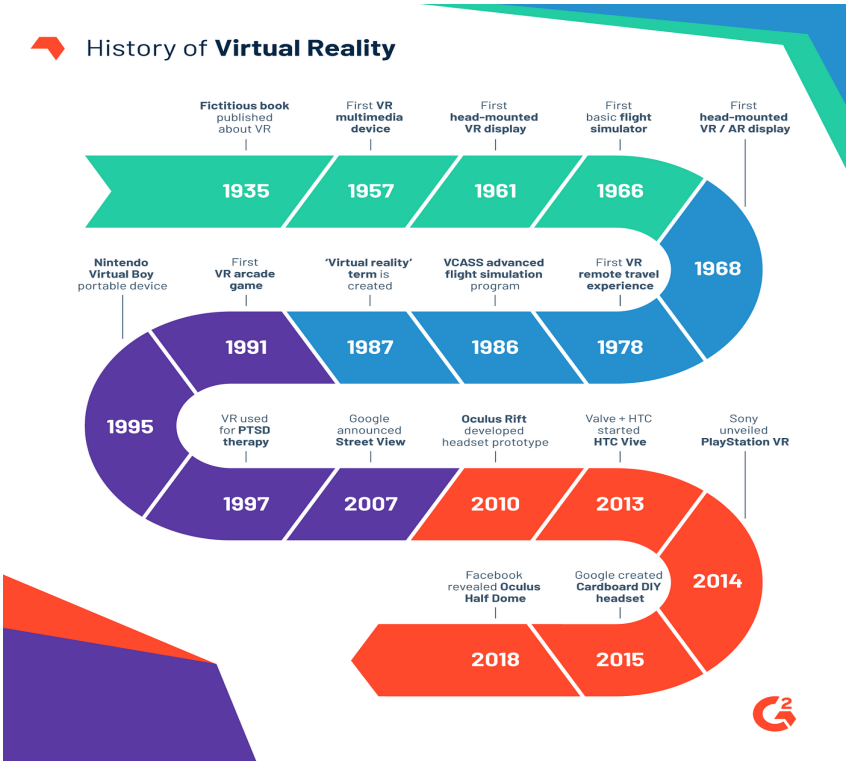
2.2 Virtual Reality

Virtual Reality is a simulated experience that intends to give the user an immersive feel of a virtual world through the employment of 3D near-eye displays and pose tracking. Currently, most VR systems use either virtual reality headsets with a small screen in from of the eyes or multi-projected environments that consist of multiple large screens in specially designed rooms. Of course, the generation of realistic environments that stimulate a user's physical presence needs 3D sound systems and ultimately, sensory and force feedback systems through haptic technology. Nowadays VR has many applications which vary from entertainment (games) and business (virtual meetings) to research (cheap

and safe simulations) and education (medical or military training).

2.2.1 Brief History of VR

There are plenty of definitions for Virtual Reality today, which all more or less overlap in key areas. For example, an early cinematic screening that showed a train heading straight to the camera is considered to be an urban legend of early virtual reality. That is because people in the attendance had no experience in films and ended up having a reaction to the footage as if the train was really headings towards them, than just being a picture. When we use the term VR now we specifically refer to computer generated experience developed with specifically designed hardware to have a viewing and sound outcome that is totally immersive. That being said, lets go back to the early attempts of VR and its evolution over the years 2.1.



Source: https://gdsgroup.com/insights/article/what-is-virtual-reality/

Figure 2.1: Timeline of VR history

In 1838 Charles Wheatstone invented the primary version of virtual reality headsets, the Stereoscope. His research showed that the human brain processes different two dimensional images from each eye into a single three dimensional object. The user gained a sense of depth and immersion by viewing two side by side stereoscopic images through a stereoscope. Even today, stereoscopic designs are still used, for example for the popular Google Cardboard as well as for low budget mobile phone VR HMDs.

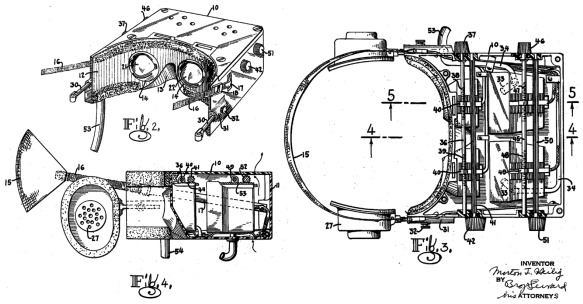
In 1849, Sir David Brewster, expert in optics, improved the earlier invention of Sir Charles Wheatstone. His innovation used a single lens cut in half so that two half-lenses acted as magnifiers as well as prisms when appropriately mounted. His stereoscope was considerably lighter and smaller than the original Stereoscope of Wheatstone. That being said, the Lenticular Stereoscope is considered to be his invention. [1]

Later on 1929, Edward Link created the Link Trainer, probably the first entirely electromechanical commercial flight simulator. Controlled by motors, with a steering wheel for pitch and roll and with turbulence and disturbance imitation from a small motor, the Link Trainer was the first "VR" tool used for human training. Pilot training is extremely dangerous, thus the US military bought and used many devices to train more than 500,000 humans for initial training and skill improving.

In 1930s Stanley G. Weinbaum (science fiction writer) derives with the idea of Pygmalion's Spectacles which let the wearer live through a fictional world by wearing goggles and using holographics, smell, touch and taste. He is called a true visionary of the VR field because the experience he described to those wearing the goggles is considered to be very close to the modern and emerging experience of VR.

William Gruber (stereoscopic photographer) and Harold Graves (president of the Sawyer postcard company) invented the Vie-Master at 1939. It was a hand held stereoscopic device that enabled multiple stereoscopic images (seven in particular) to be controlled by one viewer without single image exchanging. The device was designed to be an improvement of the single black and white image of the stereograph and the inventors hoped it could benefit the entertainment and education market. The View-Master imagery capitalized on Kodiak's new color transparency film which resulted in colorful images, making the device highly attractive. [2]

In the 1950s, Morton Heilig wrote the "Experience Theatre" that could encompass all the senses in an effective manner. Morton built a prototype of his vision called the Sensorama 2.2b, along with six short films, which he shot, produced and edited himself,



Source: https://martiancraft.com/blog/2023/06/evolution-of-vr-and-ar/

(a) Telesphere Mask

Source: https://sittinginsideanimage.wordpress.com/sensorama-telesphere-mask/

(b) Sensorama

Figure 2.2: Morton Heilig's inventions

in order to be displayed in it while including multiple senses like sight, sound, smell, and touch. Later in 1960 Heilig also developed the "Telesphere Mask" 2.2a. The patent application was the first example of a head-mounted display although it did not include any motion tracking. [3] As the user's head moved, the camera would also move thus providing stereoscopic 3D and wide vision with stereo sound.

In 1961 Comeau and Bryan, two Philco Corporation engineers, developed the Headsight, the first predecessor of today's HMD. A magnetic motion tracking system, linked to a closed circuit camera, sending the moving image to the incorporated video screen that was mounted for each eye. Head movements would allow the user to take a look around the environment by moving a remote camera. The Headsight was originally involved in immersive remote viewing of dangerous military situations and not for virtual reality applications. It is considered to be the first step in VR evolution HMDs but with the lack of computer integration and image generation.

Later on, in 1965 Ivan Sutherland, described the "Ultimate Display" concept which, in his words, could simulate reality so immersively that one could not tell the difference from the real world. His paper described VR as we know it today thus it became a blueprint for later on research on the field. [4] With the help of his student Bob Sproull, Sutherland created what was widely considered to be the first head-mounted display system for use in immersive simulation applications. The device was connected to a computer that created the simulation in real time and allowed the user to actually interact with objects in a

realistic manner. The first, technically defined, 'Virtual Reality' Device with the name Sword of Damocles designed an built by Ivan Sutherland in 1968 was scary looking, heavy and uncomfortable and was suspended from the ceiling.

In the following years of 1970 - 1987 VR devices have been mainly used for several purposes like flight simulations, automobile industry design, medical, and military training. Shortly, in 1969 Myron Krueger developed Artificial Reality which included computer-generated environments that enabled people to communicate with each other despite being miles apart. In 1975 Krueger Invented Videoplace which used Computer Graphics (CG), light projection, cameras and screens that could measure user's position. In 1977 MIT creates Aspec Movie Map which was a virtual sighting experience where the user walked through Aspen, Colorado which could be called a precursor of Google Street View. Furness Invents Super Cockpit in 1986 which featured CG and real time interactivity for pilots between movement tracking and aircraft control.

In 1987 Jaron Lanier, founder of the Visual Programming Lab (VPL), popularised the term "Virtual Reality". The research area now had a name. Through his company VPL research Jaron developed a range of virtual reality gear including the Dataglove (along with Tom Zimmerman) and the EyePhone HMD. They were the first company to sell Virtual Reality goggles and gloves.

In 1989 NASA, along Crystal River Engineering, creates the VIEW Project which was a VR simulation used to train astronauts. VIEW features gloves for fine simulation of touch interaction and looks like a modern example of VR.

In the early 90s Skip Rizzo started to work with VR to train people on how to use prosthetics and help others with rehabilitation. Today this known as Medical VR Therapy.

In the following years until 2010, a lot of different HMDs were developed and sold on the market with more and more people starting to engage with VR. Some examples are Medina's VR Mars Rover (1991), Sega VR-1 (1994) and Nintendo Virtual Boy (1995) as it comes to experience and game designed VR headset and then Google brings the Street View which contains a 360-degree street-level view of the whole planet.

In 2010 the first Oculus was designed and it wasn't until 2012 that Palmer Lucky launches a Kickstarter to fund the product that started its revolutionary path. From that time on, Facebook has bought the Oculus VR HMDs in 2014 and a lot of VR HMDs have been created and sold. Following the success Oculus had on developers and

technology enthusiasts, Microsoft, Sony, and Samsung followed up with releasing their own Virtual Reality HMDs. Google created a cardboard HMD case where users could put their smartphone inside and enjoy Virtual Reality applications.

After that, everyone turned their focus on VR and unleash products that are ready for prime time. Oculus Rift and HTC Vive lead the way, but the floodgates have truly opened and the devices keep on coming.

The cost of VR headsets has dropped dramatically and computer hardware capable of running VR is common to every-day us, with many headset options on the market. Today's HMDs dispose of extremely wide fields of view, hand scanning, eye tracking and other key developments. That is the reason why this thesis is implemented in a Virtual Reality environment. It can simulate real life conditions where users are immersed in the environment and conduct experiments with close, if not the same, results as if they were held in the real world. Moreover, Virtual Reality can guarantee the safety of the users, especially in the case of this thesis, where the experiment is held within an Autonomously Driven car, and the cost reduction of the whole procedure. Today Virtual Reality can be separated in two categories, Immersive VR and Interactive VR.

2.2.2 Immersion and Interaction in Virtual Reality

Immersion in artificial environments serves the purpose of making participants feel as immersed as they usually feel in their everyday life. It describes a technology and the extent to which the computer displays are capable of delivering an inclusive, extensive, surrounding and vivid illusion of reality to the senses of a human participant. [5] Virtual Reality offers to users, a unique and immersive experience that cannot be replicated with standard computer screens. This level of immersion can transform user's awareness of their physical self, creating a state of consciousness that feels as though they are truly present in an artificial world. Through a VR headset users can explore other worlds, be part of them and interact with them.

Interaction in VR, allows users to move inside the world and interact with objects inside it using hands (device controllers or gestures), eye-gaze or even their voice, depending on the hardware of the VR device being used. This interaction is not just about navigation but also about manipulating virtual objects and environments so that real world physics are represented and user inputs are handled dynamically.

According to Petersen [6], immersive and interactive VR environments can enhance the learning procedure by providing multimodal feedback and haptic experiences. This immersive interactivity can lead to deeper involvement, physically and mentally, which can result in better outcomes for research simulations. This kind of environments simulate a VR experience that can bear fruits, as it comes to the feedback for effectiveness, since the user tends to be more engaged. This is highly important for the research presented in this thesis, since immersion has a crucial role in the overall user experience (UX). Simulations that represent real-life situations need to be as realistic and immersive as possible, otherwise the evaluation results will be much different from what would occur in a real-life experiment.

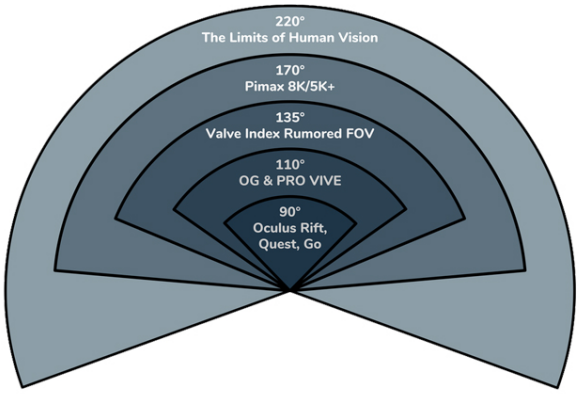
Immersion is a highly important factor in VR simulators. Specifically in this project, users need to be immersed in the environment in order to react and behave similarly to cruising in an AV in the real world. The more immersed the users feel inside the environment, the more accurate the results of our simulation and experiment stage will be.

2.2.3 Head Mounted Displays

Head Mounted Displays are imaging devices, which are meant to be worn on the head and have smaller or bigger visual display screens either on one eye (monocular HMD) or on both (binocular HMD). The image display technique may differ from one to another, as some may display images generated on the computer and others display images from the real and virtual worlds together (augmented reality-mixed reality). In the second case of mixed reality (MR), real and virtual worlds are combined, which is done by viewing the computer-generated image on a partially reflective mirror, through a direct at the real world. Commercially popular example HMDs of the first case (VR HMDs) are Oculus Rift S, Oculus Quest and HTC Vive Pro and for the MR HMD is the new Apple Vision Pro. All of them are commercial and common used headsets, while their prices vary from low to high price points. In this chapter we are going to analyze the specifications of each one of these HMDs and conclude on the rationale behind selecting the correct one for this research.

HMDs have many different use cases some of them being gaming, engineering, simulators, medical applications and personnel training. Depending on the use that needs

to be done, users have to do some research on the key characteristics and features of each of the devices. The main characteristics of the device as it comes to the display are resolution, refresh rate and field of view (FoV). Resolution and refresh rate are key aspects for an immersive VR experience and the FoV (observable area the user can see, described in angles) needs to be the highest possible in order to get closer to the human eye FoV 2.3.



Source: https://www.reddit.com/r/virtualreality/comments/b9lso6/comparison_of_the_horizontal_fov_of_different/

Figure 2.3: FoV Comparison

Wireless option in HMDs is also crucial when it comes to use cases that include moving around. The processor used is also highly important, because loading times may take too long if the use of the headset needs repeatability in high demanding tasks. Lastly, one of the most important aspects of a headset is the tracking features it provides, especially when it comes to research and simulators. Position movement, hand gestures, eye-gaze interaction and even speech may not be available or need extra external purchases if the headset does not have the tracking capabilities to implement the user's needs.

The Oculus Quest 2.4 is a popular standalone VR headset developed by Oculus, a division of Facebook (now Meta). Quest's wireless feature makes it highly convenient for users, since the elimination of cables and external sensors makes it a perfect candidate for use cases that need a lot of moving around. It is renowned for its ease of use, affordability and wireless capabilities and features a diamond Pentile OLED display for each eye, with an individual resolution of 1440 $\times$ 1600 and a refresh rate of 72 Hz (upgradable to 90 Hz

with software updates). It uses a Qualcomm Snapdragon 835 system-on-chip (SoC) with 4 GB of RAM and has a FoV of approximately 100 degrees. The software uses three out of the four 2.3 GHz CPU cores of the chip, while the remaining core and its four lower-power cores are reserved for motion tracking and other background functions. It has an Android-based operating system (OS) which enhances its performance in VR applications. It uses the second generation Oculus Touch controllers in order to enable Oculus Insight tracking. Oculus Touch controllers Gen2 relocated the tracking rings from the back of the controllers to the top, allowing them to be detected by the headset's cameras. However convenient due to wirelessness, its lower resolution and refresh rate compared to other high-end HMDs might limit the visual fidelity required for some applications, especially the ones that demand precise eye-tracking capabilities.

Source: https://www.amazon.com/Oculus-Quest-All-Gaming-Headset-android/dp/B07PRDGYTW?th=1

Figure 2.4: Oculus Quest

The Oculus Rift S 2.5, is a PC-bound VR headset (relies on computer processor) that offers improved optics and tracking over its predecessor, the original Oculus Rift. It has a Fast-switch LCD with a resolution of 1280×1440 per eye and 80 Hz refresh rate, slightly lower than other competitors, which might affect the clarity and smoothness of the overall visual experience. It's FoV is approximately 110 degrees. It offers a balance between high performance and ease of setup, thanks to its inside-out tracking system

with five built-in cameras. It includes accelerometers, gyroscopes and magnetometers, thus it removes the need for external sensors. It includes a pair of tracked controllers that provide intuitive hand presence and give the feeling that the virtual hands are actually your own. However, the lack of built-in eye-tracking capabilities makes it less suitable for research focused on gaze-based interaction which is a basic feature needed in this specific thesis.

Source: https://www.bhphotovideo.com/c/product/1590675-REG/oculus_quest_all_in_one_virtual_reality.html

Figure 2.5: Oculus Rift S

The HTC Vive Pro 2.6 is a premium VR headset known for its high resolution, robust tracking system, and advanced features suitable for professional and enterprise use. It is PC-bound like Oculus Rift S and has a Dual AMOLED 3.5" diagonal and features a resolution of 1440 x 1600 pixels per eye (2880 x 1600 pixels combined) and a refresh rate of 90 Hz. It has 110 degrees FoV, precise 360-degree controllers and headset tracking, realistic graphics, directional 3D audio and HD haptic feedback. Also, the optional wireless adapter provides flexibility while maintaining the performance of a wired connection. Additionally, it has an embedded eye tracker with 120Hz frequency for gaze data for both eyes (binocular). That is what makes it ideal for applications requiring detailed visual input and accurate user interaction for developing and evaluating gaze-based user interfaces.

Source: https://www.amazon.com/HTC-VIVE-Virtual-Reality-System-PC/dp/B07B9WPR7G?th=1

Figure 2.6: HTC Vive Pro

The Apple Vision Pro 2.7 represents Apple's entry into the high-end AR/VR market, combining clean design with advanced technology. It is the most recent commercial VR headset, released in February 2024. It has a 3D Micro-LED display with a resolution of 3660 × 3200 pixels (estimated 1800 x 1920 pixels per eye) with approximately 120 degrees FoV, and a supported refresh rate of 90hz. It is a standalone VR headset powered by Apple's highly efficient and powerful M2 and R1 chips and is completely wireless with optional tethering to other Apple devices. It does not include any hand-tracking controllers, however it relies on a comprehensive tracking system of external cameras and sensors, positioned on the headset, for precise spatial awareness and hand movement monitoring. It also features voice and eye input to navigate its own visionOS operating system. Other important technologies of the headset is a user authentication protocol based on the iris of the eye and spatial audio with dynamic head tracking. However, being a relatively new product, it might have limited compatibility with existing software and development environments compared to more established HMDs, not to mention the price range that places it in the high-end market.

HMDs differ from one another due to their specifications with each having both advantages and disadvantages, depending on the applications it is intended to be used in. While the Oculus Quest and Oculus Rift S have good features, the HTC Vive Pro features high resolution, precise tracking, and many software compatibility capabilities which make it the best choice for this project. The Apple Vision Pro is less established, much more expensive and might have some software compatibility problems. Specifically,

Source: https://gr.etoren.com/products/apple-vision-pro-1tb-us-ver

Figure 2.7: Apple Vision Pro

the HTC Vive Pro was chosen for this research due to its resolution, accurate eye-tracking capabilities and advanced features suitable for this specific research. After all, a driving simulator needs to be immersive and realistic, factors that can be achieved with high resolution, wide-range FoV and a high frequency refresh rate.

This device provides compatibility with many VR development tools and it can accurately track the eye-gaze which were some of the main reasons that we concluded to this decision. The high refresh rate, resolution and FoV provide an immersive experience within a VR-simulated environment that looks as realistic as it can, essential for evaluating how effective the AR user interface can be. Moreover, the HTC Vive Pro's wired and wireless capabilities ensure that the setup can be adapted to many different

needs without lowering the performance. These attributes make the HTC Vive Pro the most suitable HMD for the objectives of this research, which aims to develop and test a multimodal user interface for autonomous driving in a VR simulation.

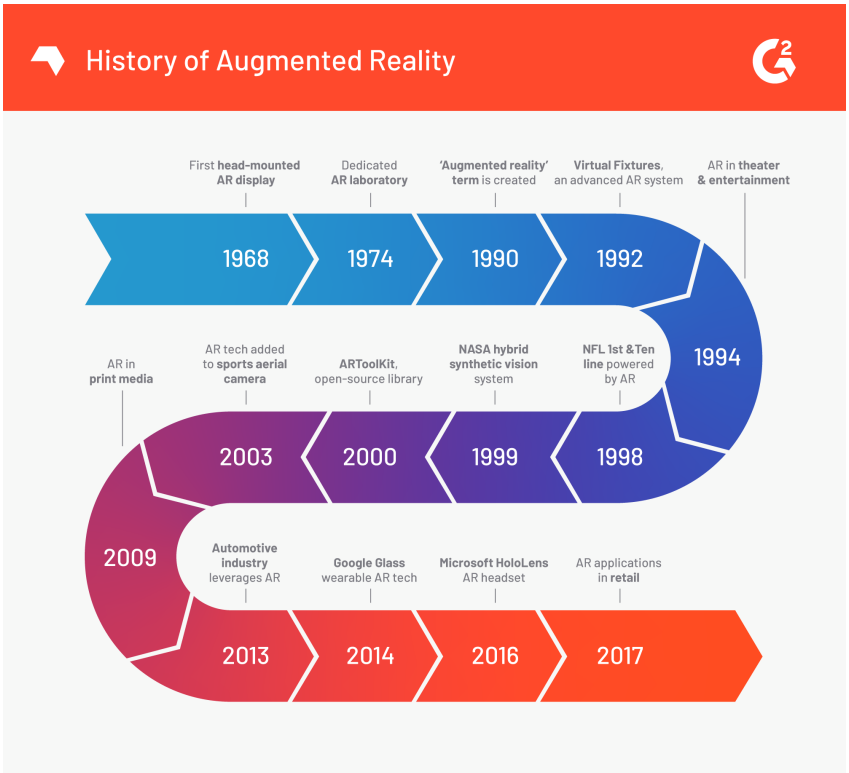
2.3 Augmented Reality

Augmented Reality is an interactive experience that combines the view of the real world with computer-generated 3D graphics. It can be defined as a system that fuses three basic features: a merge between the real and a virtual world, user interaction in real-time and accurate 3D entries of virtual and real objects. The content can affect different sensory modalities, or combinations of them, like visual, auditory, the sense of smell, haptic and somatosensory. This content can be either additive to the physical world or mask some or every aspects of it. An AR experience is perceived as an immersive aspect of the real environment, as it enhances what appears around us with 3D computer graphics. Nowadays, it provides users with more interactive and personalized experiences in every field, spanning its use from education to shopping, travel to gaming. This technology seems to be misconstrued as too "high tech" but the reality is that it is already used by everyday consumers without them knowing it is actually AR.

2.3.1 Brief History of Augmented Reality

Augmented Reality tech dates back more than 50 years, although its evolution has not come to the level we would expect today. Its history of development may intervene with that of VR in some cases, although their actual differences make them stand out from each other. That is because their specific terms have not been coined until the 90s, while their use case go back at least 30 years from that. To better understand that, let us have a look at the history of its first use cases and how it evolved until recently, in order to comprehend with its potential future impact 2.8.

Until the early 90s, the terms VR and AR were essentially the same concept, making it difficult for them to stand them out. Basically, anything that gave a different perception to human experience was considered to be virtual, no matter what it would be called in today's world. As mentioned in the VR history section, Ivan Sutherland, a Harvard professor and computer scientist, was the first to create a HMD called "The Sword of



Source: https://www.g2.com/articles/history-of-augmented-reality

Figure 2.8: Timeline of AR history

Damocles" in 1968. This device was enhancing the user's perception of the physical world using computer graphics, thus it is considered to be the first AR HMD.

Later in 1974, Krueger built a University Laboratory called "Videoplace" which became completely dedicated to artificial reality. As a computer researcher and artist, he developed an interactive yet entertaining experience. He used projection and camera technologies to emit onscreen silhouettes that were surrounding users and could be interacted with.

In 1990, Tom Caudell, a Boeing researcher, coined the term "Augmented Reality". That separation between AR and VR helped in further future research, and essentially started transitioning AR out of the lab and into various industries and business applications.

Later in 1992, Louis Rosenberg, a researcher in the US Air Force Armstrong's Research

Lab, created "Virtual Fixtures", which is one of the first fully functional augmented reality systems. It was mostly used for military personnel training and it allowed users to virtually control and guide machinery and perform various tasks on safer flying practices [7]. This system was the first time that physical and digital objects interacted with one another.

In 1994, AR was brought to the entertainment industry for the first time. Julie Martin, a writer and producer, came up with the theater production titled Dancing in Cyberspace. The show was featuring acrobats dancing alongside virtual objects on an actual stage, and the feedback from the viewers was very good.

The next technology is an example of how much AR or some forms of it are used for years and even today without people even knowing it is considered to be AR. In 1998, Sportsvision broadcasted the first live NFL game using the virtual 1st & Ten 2.9 graphic system, widely known as the yellow yard marker or the first down line. This technology displays a yellow line, overlayed on top of the field, so that viewers can quickly see where the team just advanced to get a first down. This system is still used until today, although much more advanced than it used to be in the late 90s, and also got adopted by other sports like football (or soccer) for the offside replay.

Source: https://en.wikipedia.org/wiki/1st_26_Ten_28graphics_system29

Figure 2.9: First down line Technology

In 1999, NASA creates a hybrid synthetic vision system of one of their spacecrafts, called X-38. The system used AR technology to provide better navigation during their test flights, by displaying map data on the pilot's screen 2.10.

Source: https://www.aviationtoday.com/2012/05/01/synthetic-vision-systems/

Figure 2.10: NASA X-38 Hybrid Synthetic Vision System

In 2000, Hirokazu Kato, developed an open-source software library called "ARToolKit". This package helps other developers build augmented reality software programs and it was the start to eventually roll out AR to everyday consumers. The library uses video tracking to overlay virtual graphics on top of the real world and is still used in many AR applications until today [8].

Later in 2009, Esquire Magazine used AR in print media for the first time, attempting to make pages come alive. Specifically, when readers scanned the cover, the AR equipped magazine featured Robert Downey JR. speaking to the readers. During the same year, ARToolKit brought AR to web browsers.

In 2013, Volkswagen launched the MARTA app (Mobile AR Technical Assistance) which could give users step by step instructions within the service manual. This was one of the most groundbreaking AR technologies, as it could be applied to various different industries to align and simplify many processes.

In 2014 Google unveiled its Google Glass devices, the first commercial pair of augmented reality glasses that users could wear for immersive AR experiences. Users wearing

the AR glasses, communicated with the Internet via Natural Language Processing (NLP) commands. NLP is an artificial intelligence (AI) technology that allows computer programs to understand human text or speech better. Google Glass devices had many applications, some examples of them are Google Maps, Google+ and Gmail. The device failed the commercial market, due to high price, safety and privacy reasons.

In 2016, Microsoft starts shipping the wearable AR technology called "HoloLens", which was much more advanced than Google Glass, although expensive too, it was not an everyday type of accessory. The headset runs on Windows 10 and can be thought of a wearable computer, with different interactions. The headset became widely known but due to its price, it was mostly used for research reason. One of the main reasons that appeared with this device is that its usefulness never grew over time, and that is what contributed to the device's commercial failure. Although, it is still used until today in research departments. Later in 2016, augmented reality was brought to the masses with Pokemon Go, changing the way an average consumer thought about the the emerging technology.

Source: https://www.wired.com/story/ikea-place-ar-kit-augmented-reality/

Figure 2.11: IKEA Place App

In 2017, retail industry changed forever, when IKEA released its AR app called IKEA Place 2.11. The app allows customers to virtually preview the potential decoration options of their house before actually having to decide for objects to be purchased.

Over the past years, the adoption of AR has begun to rise exponentially, and that became a viable option through our increased dependency on our mobile devices. Everyday consumers still consider AR as too "high tech", while using it constantly on social media apps through filters, in shopping apps through glasses and clothing fitting and so on. The biggest shift in AR will have to be how it is delivered to change the perception.

In this thesis, we explore the potential of an AR WSD screen, and the impact it can have on maintaining the road awareness of the user due to its transparency. Also, AR provides plenty of hands free interaction techniques, which has great advantages compared to hand interactions when it comes to lower levels of Vehicle Automation.

2.3.2 Interaction Techniques in Augmented Reality

Interaction techniques in AR is a fundamental aspect in creating an intuitive and immersive user experience. Through the evolution of AR technology, various interaction methods have risen, with each one of the having unique advantages but also challenges. The section analysis below provides an overview of the primary and most used interaction techniques in AR. This interaction techniques analysis includes touch, gestures and voice interactions, with a detailed analysis of eye-gaze interaction and multimodal interaction techniques.

Touch Interaction

Touch interaction is one of the most direct and intuitive methods for user interaction in AR. Its most common use cases involve using fingers or a stylus to interact with virtual objects overlaid on the physical world. This method benefits from users' familiarity with touchscreens and the direct manipulation of objects as research shows [9].

Touch interaction is a highly effective method when it comes to tasks that require advanced precision. It is commonly used in mobile AR applications in which users can interact with virtual objects overlaid on real-world environments through their smartphones [10]. However, touch interaction's limits are common when users have to interact

with objects at a great distance or in 3D environments and are caused by the size and resolution of the touchscreen [11].

Gesture Interaction

Gesture Interaction allows users to interact with AR environments through hand and body movements. This interaction technique needs motion sensors and cameras to detect and interpret gestures, giving the user an immersive experience. Gestures need research to understand which movements can be natural for users and training in order to get the users used to them. The more natural and trained the movements are, the more immersive the experience of the user will be [12].

Gestures range from simple hand movements, such as grabbing or swiping, to more complex sequences of movements. Over the past years, gesture recognition technology has advanced significantly, offering accurate interaction methods [13]. Some device examples that demonstrated the potential of gesture interactions in AR systems [14] are Microsoft HoloLens and Leap Motion. Gesture interaction also has limitations including its accuracy in various lightning conditions and the fatigue that is caused to users after long consecutive usage [15].

In this thesis, touch or gesture interactions have not been implemented in order to emphasize on eye-gaze interaction. More specifically, most people that use technological means for their everyday tasks, are used to touch and gesture interactions, meaning it would be the ones mostly used in the interface if it was a choice, due to user intuitiveness. Also, in this specific case, the use of hands needs to be as less as possible, due to potential Autonomous Driving malfunctions, needing the user to take immediate action.

2.3.2.1 Eye-Gaze Interaction

Eye-gaze interaction is a technique in AR that uses eye movements as input. This method tracks where the user is looking and can trigger actions based on gaze direction and extra inputs like duration (gaze dwell time) or eye blink. Eye-gaze interaction has unique advantages, especially for this specific thesis, including hands-free operation and quick input response, making it suitable for automated driving scenarios where users should have the less possible distraction from the road view and free hands in order to be able

to take control of the car at any time [16]. Eye-gaze interaction has many different gaze selection techniques each of them having its own benefits and limitations.

Gaze-Dwell Time

Gaze-dwell time is one of the most common eye-gaze interaction techniques, in which selection trigger happens when users focus their sight on objects for a specific duration of time. This is a method that is considered to be intuitive and easy for implementation, because it does not require much training and can be quickly adopted by users. Gaze-dwell time can easily lead to user fatigue [17] as well as unintentional selections (also known as "Midas-Touch") [18] if the dwell time duration is not carefully selected.

According to studies, optimizing dwell time duration can improve user experience. Short dwell times can speed up the triggering event but may also increase unintentional selections, while longer dwell times can reduce errors but will possibly slow down the interaction process and enhance user fatigue [19].

Gaze and Blink

This technique triggers the interaction when the user looks at an object and blinks one eye intentionally to select it. This technique can reduce the possibility of Midas-Touch as the user usually has a clear intent to close one eye for selecting an object [20]. The limitation of eye blinking technique is that it is much less intuitive and may require more training, as users need to adapt between the differences of natural and intentional blinks [21].

Gaze and Gesture

Another combination of eye-gaze with another input is gaze with hand gestures. This method can enhance the user interaction by using gaze for selection and gestures for handling. For example, a user can look at an object to select it and then user hand gestures to move, rotate or resize it. This multimodal approach can be precise and serve multiple actions [22]. The problems that appear for this selection technique is that it requires additional hardware component for gesture recognition and is more complex to

implement and use. Also, this technique lacks intuitiveness for users, thus it needs more training until users get used to the different gesture functions.

Gaze and Voice

Gaze and Voice is another multimodal interaction technique based on eye-gaze. Users need to look at an object and speak to perform actions from predefined voice commands like selecting, opening or modifying the object. This method combines the strengths of gaze and voice input methods, providing a hands-free and efficient interaction experience. Although, it relies on accurate voice recognition and its interaction ease can be significantly lowered by background noise and speech variations [23]. Research showed that Automatic Speech Recognition (ASR) performance is affected by memory and computational power, showing that it has specific hardware requirements [24].

Gaze and Head Movement

An integration of gaze combined with head movement can also enhance selection accuracy. In this technique, users need to stare a target and the use a slight head movement to confirm the triggering interaction. This method can reduce accidental selections but can also be physically demanding [25] and may not be suitable for all users, especially those with mobility disabilities.

Limitations of Eye-Gaze Interaction

Eye-gaze interaction may offer many advantages, although there are several limitations that need to be addressed for an effective outcome. The most important factor for accurate eye-gaze interactions is the eye-tracking technology. The eye-tracking must be highly accurate to ensure reliable interaction. Incorrect selections and user frustration is very often during eye-gaze developing and testing, due to tracking and eye calibration errors [26].

Unintentional selections is a very often example of what can go wrong in an eye-gaze based interface. It usually happens with the dwell-time selection technique (Midas-Touch) but can also happen if the system misinterprets natural eye movement as intentional

action. Multimodal interactions with eye-gaze as their base can help solve this issue but require careful fine tuning in order to avoid other issues [27].

Eye-gaze interaction techniques used for a long duration of time can cause eye strain and fatigue, especially if users need to focus intently for extended periods. Thus, it is important to design interactions that lower continuous gaze fixation as much as possible [28].

Lightning conditions and reflections have a big impact in eye-tracking system performance. Ensuring stability under different light conditions is essential for a reliable interaction system [29]. Thankfully, in the case of this thesis, the eye-tracking procedure is handled by the HMD, so the light of the virtual environment cannot affect its stability.

Interaction Techniques For this thesis

In this thesis, we chose to implement eye-blink combined with speech recognition, while also having eye-dwell time as a secondary choice, but not active if the user does not choose to activate it. Eye-blink and eye-dwell time allow for hands-free control, which is practically beneficial in immersive environments where hands may be needed for other, safety-lerated, reasons. These techniques utilize natural eye movements, making them intuitive and reducing the learning curve for users. When combined with speech recognition, these methods provide a multimodal interaction that can effectively improve user's ease of handling the interface. This combination enhances the usability and accessibility of the system, ensuring that interactions withing the virtual environment are both efficient and engaging.

2.3.2.2 Multimodal Interaction

Multimodal interaction techniques in Augmented Reality involve the integration of multiple input modalities such us eye-gaze, voice commands, hand gestures and touch inputs to create more flexible, efficient and natural user interfaces. By using the strengths of different interaction methods, multimodal interfaces allow users to select the most suitable method for any task to enhance their overall user experience. Research has shown that multimodal interaction can highly improve user performance and satisfaction in AR environments [30].

Moreover, multimodal interfaces can support more user needs and preferences, making AR system more inclusive, personalized and accessible. Research shows that multimodal interaction can improve task performance, reduce cognitive load and improve user satisfaction. Specifically, study participants [31] using multimodal interfaces completed tasks quicker and with fewer errors compared to those using single-modality interfaces.

However, multimodal systems face several challenges as it comes to multimodal interaction. The multiplicity of the input modalities increases the complexity of the system, requiring advanced hardware and bug-proof software to accurately capture inputs. As mentioned above, each of the modalities may have specific limitations, like voice input is gets worse when it comes to background noise or gaze-based systems from environmental lighting. Intuitiveness is also a drawback of this kind of systems which means that users usually need more training than single-modality systems that appear to be more straight forward [32].

In conclusion, multimodal user interfaces seem to hold significant promise for enhancing user experience in AR. More precisely, in applications like autonomous driving, where flexibility and efficiency are extremely valuable, multiple modalities can provide a more natural user experience. In this specific thesis, by combining eye-gaze and voice commands we intend to develop an interface that combines utmost task usability and the best possible road awareness simultaneously. To make that happen, we need to address the challenges of integration, user comfort and intuitiveness by designing an effective and usable multimodal user interface.

2.4 Autonomous Driving

The evolution of autonomous driving technologies marks currently one of the biggest transformation in transportation research and development. Compared with human driven vehicles, AVs offer many advantages such as reduction in the number of vehicle crashes by eliminating human error, increased mobility of disabled and elderly people, increased productivity of travel time by allowing the driver to engage in other activities, improved traffic flow and fuel efficiency. Many of the biggest companies in the world, either from the automotive industry or not, have entered the race of who will develop the first fully automated driving car. This section provides an overview of the key features involved in autonomous driving, the different levels of automation, state of the art

autonomous driving systems and technologies they use. Moreover, there will be an analysis which affects the thesis research, which includes the importance of Human-Vehicle Interaction (HVI) and the role of simulation and testing in advancing AD technologies.

2.4.1 Overview of Autonomous Driving

Autonomous driving technology aims to vehicle operation without the need for human intervention. In order for that to happen, AVs use various sensors, software algorithms and machine learning (ML) techniques. Society of Automotive Engineers (SAE) has defined six levels of driving automation, which range from level 0 (no automation) to Level 5 (full automation). In that way, SAE provided a framework for understanding the progression of AV capabilities [33].

Levels of Automation

- **Level 0** means the vehicle has no automated assistance and the human is responsible for driving. More specifically, the driver steers, brakes and accelerates without the support of assistant systems. Moreover, there are some electronic helpers that, according to SAE definition [33], exist under level 0. Examples of those helpers are Electronic Stability Control (ESC) which helps with the vehicle stabilization when it detects momentary loss of control, and Emergency Brake Assist (EBA) which detects situations that emergency brake is required by measuring the speed with which the brake pedal is depressed. These features are not considered to be parts of automated driving, since they only support the driver in certain situations.

- **Level 1** is about driving assistance, which means that the vehicle can assist with either steering or acceleration/deceleration but not both simultaneously. Adaptive cruise control (ACC) and lane-keeping assistance (LKA), with which the vehicle calibrates the speed and its distance from the vehicle in front, are examples of this level automation. The driver can override these advanced driving assistance systems (ADAS) at any time or even switch them off if that is an option (differentiates in each car). These ADAS are available in almost every vehicle class already.

- **Level 2** describes partial automation. Specifically, the vehicle can control both steering and acceleration/deceleration under specific conditions. Level 2 for example, is considered to be a vehicle that includes both LKA and ACC that are combined in one system. The responsibility for everything the vehicle does still remains solely with the driver who must monitor the system at all times and be able to intervene immediately if necessary.

- **Level 3** refers to conditional automation, which means the vehicle can perform all driving tasks under specific conditions. There is a big leap between level 2 and level 3, because the vehicle temporarily takes over the driving task from the driver without the constant need for human intervention. In this level of automation, the driver may pursue other activities withing certain limits. However, the human must be aware at all times and ready to take over because when the system reach their limits the driver only has a short amount of time until the intervention is mandatory.

- **Level 4** declares high automation. The AD can operate without human intervention in designated areas under certain conditions but may require a human driver outside these areas. At level 4, the human being no longer has to be ready to take control of the vehicle. During this time, humans can work, watch movies and even sleep. The vehicle may also drive alone, without passengers. However, the autonomy of the vehicle at this level is still linked to certain conditions such as defined routes, driving on the highway or in parking garages.

- **Level 5** is about full automation, which means that the vehicle can perform all driving tasks under every possible condition completely autonomously, without any human intervention. The vehicle can driver anywhere without limits, in road traffic and roadways and under all conditions without human beings. That being said, these vehicles do not need to have a steering wheel, gas or break pedals. At this stage, humans are turning into passengers.

Key technologies

Autonomous driving in no longer a distant vision of the future, with many examples of automated systems already on the roads. Moreover, in order to develop an AV there

are specific complex technologies that need to coexist.

Autonomous Vehicles rely on a combination of sensors, including cameras, radar, LiDAR and ultrasonic sensors in order to perceive their environment with precision. These sensors provide the data that the AD needs to detect objects of its environment in order to under what is surrounding it. The more precise the information about the environment, the more informed and correct the decision of the AD will be [34].

Afterwards, perception algorithms process these sensor data to identify and classify environmental objects of the AV. Perception algorithms are also responsible for predicting the movement of the objects, to fully understand the environment of the vehicle [35]. These algorithms use machine learning (ML), and particularly deep learning, to be able to advance their perception capabilities. The most common deep learning methodologies applied to AD are convolutional neural networks (CNNs), recurrent neural networks (RNNs), and deep reinforcement learning (DRL). Different neural network architectures are used to detect objects as 2D regions of interest, pixelwise segmented areas in images, 3D bounding boxes in LiDAR point clouds as well as 3D representations of objects in combined camera-LiDAR data.

The most popular architectures for 2D object detection in images are single and double-stage detectors. Double-stage detectors split the object detection process in two parts, region of interest candidates proposals and bounding boxes classification. Thus, double-stage detectors provide much better performance, but tend to be significantly slower than single-stage ones. Object detection on raw 3D sensory data, provide the 3D positions of the objects, however point clouds do not contain the rich visual information available in images. To overcome this, combined camera-LiDAR solves the accuracy problem, however the cost problem emerges. During the pixelwise segmentation process, the algorithm understands the driving scene using semantic segmentation **??**, which represents the labeling of each pixel in an image. In AD context, pixels can be marked with categorical labels which represent the different aspects of the environment like the drivable area, pedestrians, traffic participants, buildings and others [36].

Accurate localization and mapping are also essential aspects of autonomous navigation. Although this can be achieved with systems, such as GPS, it is mainly integrated with deep learning localization techniques. Localization algorithms aim at calculating the pose (position and orientation) of the AV as it navigates. Visual odometry (VO) is one of the possible localization technique, and is typically determined by matching keypoint

Source: https://sweta-nit.medium.com/how-semantic-segmentation-are-used-in-autonomous-vehicles-585d2bf404c9

Figure 2.12: Semantic Segmentation in an Autonomous Vehicle

landmarks in consecutive video frames. To improve the accuracy of VO, deep learning is used by directly influencing the precision of the keypoints detector. Another technique is LiDAR intensity maps, who are also suited for learning a real-time localization for autonomous cars. This method uses a deep neural network to construct a representation of the driving scene from LiDAR data and intensity maps [37]. These methods belong to the area of simultaneous localization and mapping (SLAM) [38].

Path planning algorithms come into action then, to determine the optimal path for the vehicle, considering factor such as obstacles, traffic and other road conditions like maintenance or events. During the path planning process, a self driving car considers all possible obstacles that are present in the surrounding environment and calculate a trajectory along a collision free route. Optimal path planning should operate at high computational speeds in order to achieve short reaction times, while having specific optimizations [39]. The state of the art literature has revealed increased use of deep learning technologies for path planning and behavior arbitration and two of the most representative examples are Imitation Learning (IL) and Deep Reinforcement Learning (DRL). IL is a technique during which the autonomous driver is trained to imitate the behavior of a human driver from recorded driving experiences, using CNNs [40]. During the DRL

technique, the autonomous driver is trained in a virtual environment which is a transform of a real environment [41].

Both path planning techniques have advantages and disadvantages. IL's advantage is that it can be trained with data collected from the real world. However, this means that this data may no include every possible corner case, making the trained driver's response uncertain when confronted with unseen data. One the other hand, the DRL technique seems to be able to explore different driving situations within a simulated environment. However, these models tend to have a biased behavior when they have to act and decide in the real world [36].

Another technique to be assessed in the future is the deployment cooperative driving through the connection of automated vehicles. For this to happen, the autonomous driving procedure needs to become a multiagent setting where the host vehicle must apply sophisticated negotiation skills with other autonomous drivers. The literature finds that communication through autonomous vehicle can resolve many autonomous driving problems, although for that to happen, all possible vehicles on the road should be under the same connection, which is not easily feasible [42].

Lastly, control algorithms ensure that the vehicle follows the planned path safely an smoothly and aims to the user's comfort and trust on the autonomous driver. The two main categories of control algorithms are Learning controllers and End2End learning controllers. Learning controllers make use of training information to learn their models over time. They mostly use Iterative Learning Control (ILC) and Model Predictive Control (MPC) methods to learn a dynamic model. ILC is a tracking control approach for systems that work in a repetitive mode such as path tracking or car parking. MPC is an advanced method of process control which is used to control a process though predicting the future behavior of a system based on a mathematical model. The major advantage of learning controllers is that they combine model-based control theory with learning algorithms, which makes a stable and predictive system possible [43].

An End2End control systems ignore all complex sensor data processing or control logic and process input values brought into images by using DNNs to output the control signals immediately. End2End learning can also be considered as backpropagation algorithm scaled up to complex models. An image based End2End autonomous system can be configured at low cost [44].

Throughout the whole procedure, a lot of data sources are required for training autonomous driving systems. The use of real world data is a key requirement for training a testing an effective autonomous system. Data collection on public roads has been turned into a valuable activity due to the high amount of data needed in the development stage of autonomous components. Over the past years, many driving data sets have been made public and documented due to the large and increasing research interest in AVs [45].

In this thesis, the autonomous driver is supposed to be at a SAE Level 3, which means that it is able to handle most of driving tasks and scenarios, however the user needs to be constantly aware of the surrounding environment in case of human intervention requirement. However, the AD does not use some proper technique of AVs as mentioned above, but on the other hand is developed to cruise around a city environment while following a predefined route.

2.4.2 State of the Art Autonomous Driving Systems

This section refers to some of the leading autonomous driving systems that exist today. Each of them uses state of the art technologies to achieve different levels of automation as defined by SAE.

Tesla Autopilot and Full Self-Driving

Tesla's Autopilot and Full Self-Driving (FSD) systems are probably the most popular in the autonomous driver consumer market. Current Tesla's Autopilot enables the car to steer, accelerate and brake automatically within its lane under certain conditions, requiring the constant supervision of the driver. The system can optimize navigate to the preferred destination, making adjustments so the car does not get suck behind slow cars, and also features smart summon which makes the car come find the driver even in complex parking spaces. FSD hardware is installed in every Tesla car but not yet available due to lack of training and regulatory approval, but software is constantly being updated automatically whenever a new capability is introduced. Current automation level of Tesla is SAE level 2. This system uses a suite of cameras, ultrasonic sensors and radar, without using LiDAR. The system is designed and trained through Deep Neural Networks for perception and control. [46]

Baidu Apollo

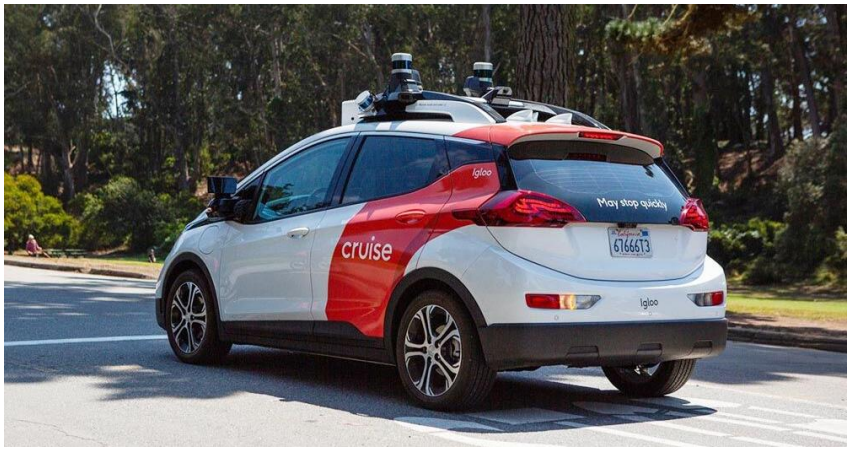
Baidu's Apollo is another significant autonomous vehicle candidate in the field. Its systems provide automation at SAE Level 3 and Level 4 in more than 10 cities in China with their Robotaxis already working in specific areas for the past year. Apollo uses a comprehensive sensor suite including LiDAR, cameras and radar, combined with AI perception and planning algorithms. The company has been testing its systems safety and effectiveness in various cities and has developed an open-source platform to accelerate autonomous driving development. [47]

Cruise

Cruise, owned by car manufacturer General Motors, was founded in 2013 to develop and test autonomous car technology. The company focuses on creating a reliable and safe driverless experience for consumer ride services in dense urban areas. It now features SAE level 4 autonomous vehicles designed for urban environments. The company has received a permit to launch plenty of Robotaxis 2.13 in several states of the US to provide driverless taxi rides. Due to a barrage of safety concerns in October 2023, the use of the company's Robotaxis was suspended for a few months, and activated again after a thorough investigation has taken place. Cruise's vehicles are equipped with a full sensor suite of LiDAR, radar, and cameras. Perception and planning algorithms are handled and trained by DNNs while prediction is being made by CNNs. [48]

Waymo

Waymo, formerly known as the Google Self-Driving Car Project, operates one of the most advanced autonomous driving system, featuring SAE level 4 automation. Waymo offers robotaxi services in Phoenix, Arizona and San Francisco, with plans to expand to Los Angeles, California. The company manufactures a suite of self-driving hardware, which includes cameras, sensors, radar and LiDAR and a hardware-enhanced vision system. Waymo's deep learning architecture VectorNet is used as a path planning algorithm for complex traffic scenarios and it's graph neural network as a prediction algorithm which has demonstrated state-of-the-art performance on several benchmark datasets for trajectory prediction. Waymo Carcraft is the company's virtual world where Waymo



Source: https://www.davisenterprise.com/news/state-agencies-ground-cruise-driverless-cars-for-public-safety/article_9eaf4c6a-7847-11ee-8adb-4bfa927e8ebb.html

Figure 2.13: Cruise Robotaxi

simulates driving conditions to train its autonomous drivers through the navigation of several existing cities of the US. [49]

2.4.3 Human-Vehicle Interaction

Human-Vehicle Interaction (HVI) is a critical aspect of autonomous driving, especially at automation levels that human intervention is still required. An effective HVI design ensures that the driver can understand the vehicle's actions and intentions, affecting the trust and psychological status of the driver and take control when necessary.

Trust and Acceptance

Building user trust in autonomous systems is one of the most important factors for their acceptance and every day use. Research shows that showing the autonomous driver's intentions and route information can highly affect user's trust and mental state throughout the transportation process. [50] As a matter of fact, the way of displaying this information as well as the time of displaying it, affect the whole experience. Specifically, the way of displaying this kind of information should be simple and intuitive, so it can

be read and understood fast, without the need of training. Also, the study showed that users mostly prefer to have the information available a little prior to the actual action compared to real-time, since it can be easily frustrating. Transparency in system actions and clear communication of the vehicle's capabilities and limitations are essential. Driver's intervention, in situations that are not properly informed, is not optimal in both time and quality of response [51].

Situation Awareness

Drivers must maintain an adequate level of situation awareness to effectively intervene when necessary. This requires the system to provide relevant information without overwhelming the driver. The design of the system's interface can significantly affect the situation awareness of the user [52]. AR is the perfect candidate for in car displays when it comes to non driving related tasks, since it enhances perceptual and cognitive of the natural environment, thus increase continuous awareness. However, that does not change the fact that the users might take their sight off the road for a few moments, but unfortunately that is something that could also happen in of the context of manual driving as well. Research shows that situation awareness, during the time that the AR display is on or off, is not changed or affected. Familiarity with the interface is a factor that decreases the awareness of the user, and it can be highly affected by training or age since it usually decreases the intuitiveness [53].

Handover Process

The transition of control between the autonomous system and the human driver must be smooth and intuitive. This includes clear cues and sufficient time for the driver to regain situational awareness and control [54]. Research shows that the analysis of driving data by the user can reflect the actions he is able to take depending on the situation. That means that the analysis of the eye tracking data can give clues about what the driver has possibly detected and analyzed. The result of this analysis can give insights about how far the driver can understand and anticipate the situation, thus calculate the needed time for the driver to regain control [55].

Our implementation tries to implement all of the basic aspects of an effective HVI as mentioned above. The Interface is designed and implemented in a way the integrates

transparency for user's constant road awareness, while maintaining the sight of the user in specific spots like the area that the user would look at if he was driving. Moreover, the system tries to improve user's trust and acceptance by informing for every possible move that is being made, some frames prior to the start of this movement.

2.4.4 Simulation and Testing in Autonomous Driving

One of the most fundamental aspects of developing autonomous driving technologies as well as interfaces to enrich the experience of automated driving, is the process of simulation and testing. VR can have a significant role in creating realistic and immersive environments that represent real world situations and test the effectiveness of the automated vehicle over a wide range of driving scenarios. Moreover, having user's safety guaranteed, testing and evaluating different design styles of AR interfaces can be accelerated.

Importance of Simulation

- **Safety**: Simulation allows for the testing of autonomous systems in a controlled and safe environment, reducing the risk of accidents during the development phase. VR can also support the AV's Verification and Validation (V&V) process safely, which is a standard procedure for checking if a system meets the requirements and specifications that fulfills its intended purpose [56].

- **Scalability**: Developers can simulate endless scenarios, including rare and hazardous situations that would be difficult and dangerous to test in the real world [57]. Another state-of-the-art virtual test architecture proposes the Collaborative Virtual Environment concept, in which diverse contributors belonging to different organizations design and execute tests for autonomous driving software [58].

- **Cost**: Simulation reduces the need for physical testing and accelerates the iteration process [56]. Furthermore, there is no need for the construction of physical models, of the automated vehicles to be tested, solely for the scope of testing. Also, testing an automated driver in the real-world, always bares the risk of car crashes and human injuries.

Role of VR and AR

- **Immersive Testing Environments**: VR provides immersive environments where developers can test the interaction dynamics between autonomous systems and human drivers thoroughly. These environments can be developed with real-world conditions which gives the chance for detailed analysis of system performance. They accelerate driver training to interact with autonomous systems confidently, building trust and reducing hesitation towards AVs [59].

- **Augmented Reality Overlays**: AR can be used to overlay digital information on real-world environments, helping with the visualization of sensor data, planned paths, and potential hazards. Overlaying all type of preferred not only enhances the understanding of the system behavior, but can also significantly improve and accelerate the debugging process [60].

2.4.4.1 Mobile Office Concept

The advancement of AVs is not only transforming transportation, but also the NDRTs that users can go through while navigating to their destination. As mentioned in automation levels analysis 2.4.1, above SAE level 3, users are able to complete some tasks without the need for constant awareness of the driver's actions, and from SAE level 4 and above, the car does not need user's attention at all. One implication that emerges with the increase of SAE levels, is the concept of the "future mobile office". In this concept, AVs serve as workspaces that enable productivity on the move, providing unique benefits as well as some challenges.

Concept and Potential Benefits

The future mobile office, converts AVs to mobile workspaces equipped with toold and technologies that implement office tasks. This transformation potentially provides increased productivity, since travel time is utilized more effectively. AVs can offer flexible workspace that can be adapted to various professional needs, from individual work to collaborative meetings. Hybrid and work from home scenarios are very common nowadays, so the car could be one more work space for that purpose. Getting some tasks done

while navigating to your destination could lessen the work hours, since, otherwise, this transportation time would be wasted, when it come to daily work hours [61].

Technological Requirements

Undeniably, to implement the concept of mobile office, several technological advancements and integrations are necessary. Reliable high-speed internet connection is essential for communication tasks and access to cloud services, and collaborative tools. The interior of the AV should be designed to support ergonomic seating when it comes to interactions with an interface, which is supposed to be developed according to the interactions techniques. Furthermore, effective noise cancellation technologies are required to create a quite environment, so users can be focused on their tasks, especially when it comes to urban driving.

Challenges and Considerations

Implementing a concept like that presents many challenges, especially since the automation level of AVs is not yet ready for commercial use, thus the evaluation and testing are still confined to research departments. A critical factor that will never change in the concept of AD is the safety of the passengers. Ensuring that office functionalities do not distract passengers from safety related information coming from the interface, is an absolutely critical factor. The interface should balance work related tasks and situational awareness, and make sure that the user is properly informed about the driving state and conditions. When it comes to accessing sensitive work related information, privacy and security are mandatory, while using a mobile environment. Lastly, a concept like mobile office differs from traditional office settings and may not be intuitive to users. That probably means that training may be unavoidable to help ease this transition and adapt to working efficiently.

In this thesis, some basic functions of the Mobile office concept have been implemented. Specifically, the user can accept and start voice calls, write and read messages and handle notifications. In this scope, those functions are already enough, due to the level of the AD not being high enough for the user to engage more complex tasks.

2.5 User Interface Design

User Interface (UI) design is an important factor of developing effective and intuitive systems, especially when it comes to VR and AR technologies. In this section there will be an analysis of the essential principles for UI design, specifics for VR and AR UI interfaces, design challenges for autonomous vehicle UIs and methods for evaluating usability and effectiveness.

2.5.1 Principles of UI Design

Effective UI design is based on specific principles that aim to make the user experience better by enhancing usability, accessibility and user satisfaction. **Consistency** is a major factor of effective and user friendly interfaces because it helps users learn and predict how the system will behave. As a term in design, it refers to uniformity in visual design and terminology, and interaction patterns across the interface [62]. Similar components throughout the interface should have similar looks, uses and operate the same way. Specifically, the same action should always yield the same result, the function of elements should not be interchangeable and the position of standard elements should not change for the aspect of consistency to be intact [63].

Another basic aspect of an intuitive UI is **feedback**. That is, because providing immediate and informative feedback to the user, in any form, visual, textual, or even auditory can help with understanding the results of their actions and maintain a sense of control [64]. Visual feedback can be provided with many different methods, for example changing the color, transparency or size of a button when the cursor is hovering it, or showing the loading icon when a function needs a few seconds for the result to appear, showing that way that the interface is still working. Textual feedback is a technique that is mostly based on functional behavior, for example when a search condition is not met and a text appears to show that there are no results for this specific search keywords. Some auditory feedback examples are short audios that inform the user that the triggered action was successful or not with positive or negative tone of music respectively [63].

Simplicity is a fundamental principle of UI design. A simple and clean interface reduces cognitive load and helps users focus on their tasks. The user interface should be easy to understand and navigate, with the lowest possible distractions. The simpler the design, the easier it is for users to accomplish their goals without frustration or confusion

[64]. This involves minimizing unnecessary elements and using clear, concise language which also reduces time needed for task performance and minimizes user frustration [65].

The factor of **visibility** of important information and controls is crucial to an effective UI and pleasant user experience. Main aspects that the UI needs to show like information or controls should easily accessible and visible. This ensures that users can find what they need quickly, without time consuming search or navigation [66]. This can happen either with toolbars that are always on display, constantly showing important information or through shortcuts of the most important aspects of the interface.

User-centered design refers to UIs that always put the user's needs and preferences first. The design of UIs that try to implement that should be tailored to the target audience and make the user experience intuitive, enjoyable and efficient. UIs should also be accessible to all users, including those with disabilities [64].

Lastly, **affordance** is a main aspect of UIs to improve intuitiveness. It means that the elements consisting an interface should suggest their functionality. For instance, buttons should look clickable, sliders should indicate they can be dragged and search inputs should clearly show the message that they can receive texts [67].

2.5.2 UI Design for VR and AR

Specifically in the process of designing UIs for VR and AR environments extra challenges and considerations arise, compared to 2D interfaces. Fundamental principle standards of UI design still need to be met precisely for the effectiveness of the basic functions of the interface, but there are some extra parameters that appear in order to improve intuitiveness and user experience.

Immersion and Presence

One of the primary goals in VR and AR UI design is to maintain a high level of immersion and presence. This involves creating interfaces that feel natural and integrated into the virtual or augmented environment [68]. Disruptions to immersion, such as hard-to-handle controls or intrusive UI elements can lead to a worse user experience.

Spatial Interactions

UI interactions in VR and AR includes users to interact with elements in a 3D space. This requires to design interfaces that implement spatial interactions such as gaze-based interactions, hand gestures and spatial audios in order to take advantage of the 3 dimensions. By leveraging this kind of interactions, the accessibility and intuitiveness of the interface can be enhanced [69].

Ergonomics and Comfort

The use of VR and AR system for a long period of time can lead to physical discomfort and fatigue. The design of the UI should be considered in a way that it minimizes the need for extensive head and hand movements which are the most common of fatigue in extensive UI use. Also, if the interactive elements need some kind of movement to be interacted with, the design should be implemented that way, so they remain within a comfortable reach [70].

Visibility and Readability

Ensuring that UI elements are visible and readable in various lighting conditions and viewing angles is crucial in VR and AR interfaces. For example, the visibility of AR elements under interior lightning conditions during the daylight is much better compared to outside conditions. Designers need to use appropriate contrast, text size, and positioning to maintain readability and reduce eye strain [71].

2.5.3 UI Design for Autonomous Vehicles

The design of UIs for AVs has specific challenges, since the interface has to support both the operation of the autonomous driver and potentially, human intervention if it is required.

Road Awareness

User's road awareness should never be off the road since the current SAE levels of automation may need human intervention at any time. This is possible when implementing

an AR UI since the display is transparent enough to be see-through. In the process of designing the UI, the time needed to complete a task should be they lowest possible in order to take user's awareness off the road for the least possible time duration. Also, the UI components should be placed in a way that the the user's eyes look in the center of the windshield for most of the time compared to the corners for example [72].

Situational Awareness

The difference between situational and road awareness is that situational awareness is a result of driver's awareness and information coming from the AV through the UI. The UI should provide relevant information about the driving environment, such as nearby vehicles, road conditions and navigation updates [52].

Trust and Transparency

A UI implemented in an AV display should try to gain user's trust which is an essential factor of user acceptance. For that to happen, the UI has to communicate the vehicle's current status and intentions, and any required actions from the user. The user has to feel comfortable and in control of the situation which can only happen through transparent information about the vehicle's decisions and behavior [50].

Handover Procedures

Effective handover procedures are critical in situations where control transitions between the autonomous system and the human driver. When it is required, the AV should inform the user that he needs to take control, but not in the last possible moment, while it would be even better to have given some hints before having to completely pass the control to the user, ensuring a smooth and safe transition [73].

Interaction Modalities

The way of interacting with the interface is the most fundamental factor that affects effectiveness, task completion speed, less need for training and user satisfaction in a system. The design of the interface needs to be according to the input modalities that are allowed. Multimodal interaction methods, such as voice commands, touchscreens,

eye-gaze and physical controls, can enhance the usability and flexibility of a UI developed for an AV. Combining different modalities allows users to interact with the system in the most convenient and effective manner [32].

2.5.4 Evaluation of User Interfaces

Evaluation of a system designed for commercial use in general, affects its scalability and accelerates the evolution of the end product. Particularly in UIs, evaluating their usability and effectiveness is essential to make sure that they implement what users need and support their intended interactions. There are several methods that can be used to evaluate UIs in VR, AR and AV scenarios.

Usability testing

Usability testing is a method to get feedback through observing real users interaction with an interface. This method helps designers to identify usability issues, and understand how users perceive and interact with the interface. Identifying areas of improvement through experiments with real users can be harder than it seems, because the way of designing the experiment can affect the quality of the results. User-based studies should employ user tasks that are representative but not so specific that the findings cannot be applied to the target interface. The experiments should be conducted using the equipment and the environment that is most likely to be used in the actual application setting. Evaluation process should be quick, as it is quicker to design, develop and run but also easier to analyze, which helps designers to focus on the most important factors of the UI [74].

Heuristic Evaluation

Heuristic evaluation involves experts reviewing the interface using established usability principles in order to identify potential problems. This method is quick and cost-effective and can identify issues that may not be found in user-based testing. Another advantage of heuristic evaluation is that it does not require advanced planning and that most of the times, it can be conducted in the early development stages. One factor that makes heuristic evaluation not so common to use is that, in order to have accurate results,

the more the evaluators, the best are the results, and they should probably conduct it independently [75].

User Experience Metrics

Quantitative UX metrics, refer to task completion time, error rate, and user satisfaction scores and provide an objective and accurate view of the UI's performance. These metrics can be collected during the usability testing or through extra surveys and are very helpful to improving the interface. They are used to measure user's behavior and capability while interacting with the interface and their outcomes can extensively improve the system and user experience when they are solved [76].

Cognitive Walkthrough

A cognitive walkthrough involves a detailed walking through the interface, in order to understand the user's perspective and identify potential usability issues at each step. This method focuses on understanding how new users learn and navigate an interface they have not used before. The results of this method intend to make the interface more intuitive for new users. A cognitive walkthrough has two phases, preparation and evaluation. In the preparation phase, the evaluators come up with tasks that represent some basic functions of the interface. For each of the tasks, they describe the initial state of the interface, the action sequence used to accomplish the task and the user's initial goals. During the evaluation phase, the interaction between the user and the interface is analysed in depth [77].

In this thesis, we test the effectiveness of the UI using User Experience Metrics evaluation for task completion speed and error rates for both of the available eye gaze selection techniques. Then we conducted a Usability Testing study through letting the users test every aspect of the UI and answer the questionnaires which analyse their preferences between interaction techniques, and their trust towards the AV as well as the productivity within the mobile office concept.

2.6 Human Factors and Usability

In the context of technologies like VR, AR and AVs, it is crucial to research the human factor and usability, in order to develop interactive systems. The following section explores user experience (UX) design on user satisfaction and performance, techniques and metrics for usability testing, along with factors that need to be considered for immersive autonomous systems.

2.6.1 User Experience (UX) in Immersive Technologies

UX design is an essential factor for the success of VR, AR and AV systems. UX aims to create meaningful and satisfying experiences, by combining all of the aspects of the user interacting with the system.

Impact of UX Design on User Satisfaction

An interactive system that combines intuition and enjoyment in use with accurate responsiveness, enhances user satisfaction and can be considered as a well-designed UX. The key factors of such experiences in immersive technologies, need to keep the fundamentals of UI design as standard 2.5.1. Specifically, immersion and presence are crucial factors for a realistic experience, and the feedback and responsiveness of the system helps users navigate intuitively and maintain their engagement in the whole experience. Furthermore, intuition can not only be obtained through a good feedback because, in the first place, a system needs to be easy to learn and use to minimize the cognitive load of the user which highly contributes to a positive UX [78]. A system that about commercial use should be accessible by everyone. People with disabilities should be able to go through the whole experience without having to skip parts or go through high physical or mental fatigue. This involves considerations about visual, auditory, and physical impairments.

Impact of UX Design on User Performance

UX design can also impact user performance, especially in task-oriented applications like AR interfaces found in AVs. One of the key considerations user task performance is about task efficiency. A system should be able to have its tasks efficiently completed by

providing clear directions to achieve goals. The steps between the completion of a task should be minimized as much as possible [78].

Effective UX design should try to minimize user errors, or the possibilities of their appearance, and provide easy ways to recover from user mistakes. Some ways for the implementation of that is error messages, so users are aware of the error and try to recover from them, and undo functionalities in order to continue their navigation prior to the error. Moreover, focus on the task completion is mandatory for the effectiveness improvement of the system, so the cognitive load should be reduced as much as possible. Simplifying the visual appearance of the interface elements and the complicity of the interactions should be enough to reduce the mental capacity needed, and result to a satisfying UX [64].

During the implementation of this project, there has been a big section of research over the placement of the interactable components throughout the UI. Through research, trial and error, and think aloud methodology with other Experienced VR users of the SURREAL team lab, we concluded in a layout that minimized the error rate and task completion speed as much as possible. Having pointed out the drawbacks and weak points of the hardware and the tools used, we build a UI based on them, by trying to minimize their effect.

2.6.2 Usability Testing

It is essential to evaluate the effectiveness and user-friendliness of interfaces, especially those in VR and AR environments or AV interfaces, and that is something that can be done by usability testing. There are various techniques and metrics used to achieve the evaluation of usability, identify issues and end up to design improvements.

Techniques for Usability Testing

- **Think-Aloud Protocol**: Users are asked to express their thoughts and actions while interacting with the system. This can help to identify usability issues while understanding the thought process of the user simultaneously. Thinking aloud to understand the end users' thought processes in interaction with a system can support the development of usable systems by identifying system deficiencies. One

drawback of this testing technique is that it might disturb the cognitive process of the user [79].

- **A/B testing**: It is a method during which, different versions of the interface are tested with users to compare performance and preferences. This method can help determine which design version is more effective. One important drawback of that technique is the large amount of possible version that are needed to be thought and implemented in order for the testing to take place [80].

- **Remote Usability testing**: In this testing technique, users interact with the system in their own environment, and their actions are recorder for later analysis. This method can provide information about how the system performs in real-world settings and is very effective for intuition evaluation. [65]

- **Eye tracking**: Eye-tracking technology can be used to analyze where the users look on the screen during their interaction with the system, which helps in identifying the areas of interest and potential issues with visual navigation. The continued growth of eye-tracking and its use in Human Computer Interaction (HCI) studies seems to continue as the technology is more affordable, less invasive and easier to use [81].

Metrics for Usability Testing

The metrics that play a crucial role during usability testing and highly affect the final UX are specific and need to be fine tuned as much as possible. Task completion rate is the percentage of the tasks that are completed successfully by users, reflecting the effectiveness of the interface. Furthermore, completing a task successfully is essential, but the time on task completion also matters. In fact, the amount of time users need to complete tasks, reflects the efficiency of the interface. While doing tasks, there may be some errors during the process. The error rate, is about the number and types of errors made by users, and the higher it gets, the more possible for users to feel fatigued. Error rate is a valuable metric during usability testing, because it can highlight areas where the interface may be confusing or misleading, thus needing more fine tuning. Lastly, since the purpose of UIs is user centered, subjective measures of user satisfaction are necessary

to improve UX. The most often way of obtaining feedback for user satisfaction is through surveys or questionnaires [76].

Human Factors in VR and AR

Human factors in VR and AR experience involves understanding how users interact with immersive environments. There are some unique considerations that need to be taken into account for human factors, especially when it comes immersive and autonomous systems. These considerations need to ensure that users interactions and experience is comfortable, intuitive and effective.

When it comes to VR and AR interfaces, the interaction in 3D environments seems to be inherently difficult. Therefore, during the design process needs to be very careful for the correct interaction. Users must be able to maintain a sense of spatial awareness to navigate and interact with 3D environments effectively. This involves clear visual feedback and consistent spatial mapping for an overall satisfying UX [69].

Motion Sickness

A troublesome problem that appear since the early stages of VR development is the tendency for some users to feel symptoms for classical motion sickness during and after their experience in the virtual environment. This type of cybersickness, is distinct from motion sickness because the user remains motionless but has a continuous sense of moving through the visual moving inside the virtual environment. While designing a UX, developers should consider factors like frame rate, latency and FoV to minimize the effects of sickness. Unfortunately, there are is no foolproof method for eliminating the problem, since every virtual environment is built under different circumstances [82].

Chapter 3

Technological Background and Definitions

3.1 Introduction

This chapter provides an overview of the technological components and tools used in the development of this thesis. It delves on software and hardware technologies like the technological platform used in the thesis, Unity 3D and the components needed to implement project functions. We take a closer look to various Unity components that are needed to create a 3D application, like prefabs, scripts, scenes and audio. Afterwards, the structure and the architecture in Unity is analyzed, which explains basic components needed as a starting point in development. Afterwards, Unity components that are used for the implementation of this specific thesis needs, like speech recognition and dictation, and API web requests, are covered. Lastly, a closer look to HTC Vive Pro HMD's connection to Unity through OpenXR, and its eye tracking system as well as the software needed to connect it with Unity, are presented. The goal is to provide information about the technologies used and their roles in the project, setting the stage for the terminologies that will be used in implementation Chapter 5.

3.2 Unity Game Engine

Unity is a powerful cross platform Game Engine, offering a complete 3D programming environment for developers and creators. It is a versatile development platform that provides many tools to develop 2D, 3D, VR and AR environments and interactive experiences. The wide range of tools and features in Unity, give developers the chance to create games and interactive experiences in its integrated development environment (IDE). Some are these tools include unity's physics engine, scripting tools, and an extensive library of assets and external plug-ins that can be implemented.

3.3 Basic Structure of Unity

Unity's workflow builds around the structure of its components. Games or applications build in Unity, called projects, consist of the Assets that are being used, and one or more Scenes. The Scenes consist of GameObjects and Prefabs, and each one of them has one of more Components and Scripts attached to it.

3.3.1 Assets

An asset is any item that someone can use to create a Unity 2D or 3D project. It can refer to any file that will be used to create a game such as models, textures, sounds, scripts, etc. Assets can be bought or used for free, and they are found from many sources, such as unit's asset store or other external sources. Assets can also be created using various applications like Blender and others.

3.3.2 Scenes

In Unity, the Scenes function as individual levels for the environment and game objects. Some developers create entire games in a single scene, such as puzzle games, loading dynamic content through code. Building a game in many scenes, gives the developer the choice to allocate loading and testing times separately. Scenes can be used to organize and manage different parts of a project, like in this thesis case, the development of the UI mostly took place in a test scene where only the car existed. That happened because the end-scene, with the city environment, was much more demanding in processing power so

the process of the UI developing would be much slower. At any time there is only one Scene that can be open, which is the one we are working on, as it is not possible for two scenes to work simultaneously.

3.3.3 GameObjects

Every object in the scene is called a GameObject. All GameObjects contain at least one Component which is the Transform Component. The Transform component includes the position, rotation and scaling of an object and each of them is described by the Cartesian coordinates X, Y, Z. The component can determine or change the coordinates of the object through code.

3.3.4 Components

Components have many different forms and are attached to GameObjects. They are used to create behavior, determine appearance, and affect other aspects of the operation of an object. By placing Components in an object, its functions can be changed. Common components of the game come integrated with Unity, such as the Rigidbody component, lights, cameras and more. Scripts, are also treated as components that extend or modify the existing functionality of the GameObject.

3.3.5 Scripts

A scripting language is a programming language that gives to opportunity to control one or more applications. Every game or app needs scripts to implement the user's desired input and handle gameplay events, that need to take place. Beyond that, scripts can be used to create visual effects, control the physical behavior of objects, or even implement a custom AI system for game characters or even environment animals.

Scripts in Unity are Components that extend or modify the behavior of GameObjects. They allow developers to implement game logic, handle user input, and interact with the Unity API. They are created and placed as Components in GameObjects, and they do not have any restrictions on how many Scripts can be used. For instance, scripts are used to manage the 2D UI, handle speech recognition, and control the car's movement. Unity

also offers a scripting code editor, Visual Studio, allowing developers to program using C # programming language.

3.3.6 Prefabs

Prefabs are prefabricated and stored versions of an object that can be reused in various parts of the game or app as assets. They allow users to create, configure, and store a game object complete with all its components and properties. Prefabs are useful for creating repeated elements, such as UI buttons, cars, and buildings, ensuring consistency in appearance and functionality while also saving development time.

3.4 Unity Architecture

Unity's architecture is based on its Components. In order to use these Components, it is necessary to understand the architecture on which it was built and the way it uses these Components. The most important Component is the Transform, determining the coordinates either Local or in World Space. Then there is Mesh Component and Rigidbody Physics, which are used to replicate real world collisions and gravity. To determine the way an Object is rendered, Textures, Materials and Shaders are used. For Menu and 2D interfaces, we need the User Interface (UI) Components. Lastly, all these are rendered in the computer screen (or VR headset in our case) by using the Camera Component and to accompany the image with sound, Audio Components are used.

3.4.1 Coordinates - Transform

The way of representing the coordinates varies on if the component is placed in 2D or 3D. At a 2D, each point is described by a pair of numerical coordinates X (horizontal axis), Y (vertical axis). In 3D Unity applications there is a third axis, Z, which represents the depth. This format is known as the Cartesian coordinate system. All objects have a transform component, which is used to store and treat the position, rotation and scaling of an object. Every transform component can have a parent, which allows the developers to move the object, rotate it, and scale it hierarchically.

3.4.2 Local Space vs World Space Coordinates

In any 3D environment, there is a point of origin which is represented in position (0.0.0). All positions of the objects in the 3D worlds have zero as their reference point. However, to make things simpler, we use local space to define object positions relative to other objects.

3.4.3 Mesh Component

3D Meshes are the main graphics of Unity Components. They define the shape of an object and contain the visual information of the objects, like colours and textures.

3.4.4 Rigidbody Physics

Rigidbody component controls an object's position through physics simulation. Unity's physics machine allows developers to simulate real-world responses to objects. Unity uses the Nvidia's PhysX engine, which is a popular and accurate physics machine. In Game Engines there is no assumption that an object should be affected by gravity, firstly because it requires a lot of processing power and secondly because there is no reason to do so. The physics machine uses the Rigidbody dynamic system to create realistic motion. This means that instead of objects being static, they can have properties such as mass, gravity, velocity and friction. As far as processing power increases, so the Rigidbody physics system applies to games, as it allows for more realistic simulations.

3.4.5 Materials, Textures & Shaders

Materials define how a surface should be rendered, by including references to the Textures it uses, tiling information, Color tints and more. The available options for a Material depend on which Shader the Material is using.

Textures are bitmap images. A Material can contain references to textures, so that the Material's Shader can use the textures while calculating the surface color of a GameObject. In addition to basic Color (Albedo) of a GameObject's surface, Textures can represent many other aspects of a Material's surface such as its reflectivity or roughness.

Shaders are small scripts that contain the mathematical calculations and algorithms for calculating the Color of each pixel rendered, based on the lighting input and the Material configuration.

3.4.6 Camera

The camera is one of the most important elements in a 3D game. Cameras are the devices that capture and display the world to the user. The number of cameras in a scene can be unlimited, but in most of the cases only one camera can render per time. They can be set to render in any order, at any place on the screen, or only certain parts of the screen letting the the participants see the game world from different points of view.

3.4.7 Audio

Audio components in Unity are used to add sound effects, background music, and voice feedback to the project and it consists of two basic components which are described below:

• Audio Source is the source of the sound and reproduces an audio clip which can be either 2D or 3D. This component contains settings for volume, repeatability, tone, priority over other sources, and a variety of other settings and effects.

• Audio Listener works like a microphone device, receiving input from the audio sources of the scene and playing sound from the computer speakers or headphones. There should always be a Listener activated per time. It is usually placed on the main camera as the participant hears and sees from there.

3.4.8 User Interface (UI)

Unity's User Interface (UI) system allows the developers to build interactive interfaces. The User Interface system consists of a Canvas, in which all UI elements are placed. These include buttons, text fields, sliders, panels and others. In this project, the 2D UI within the car uses these components to create an interactive and user-friendly interface which can be handled through voice and eye inputs.

3.5 Speech Recognition

Speech recognition functionalities are implemented to allow users to interact with the system using their voice. These features have many ways of implementation but in this specific thesis we take advantage of built-in Unity phrase recognition and dictation systems.

Phrase Recognition system enables users to issue commands and interact with the UI without using their hands. It is responsible for managing phrase recognizers and dispatching specific events on them. Specific phrases need to be predefined in order for the system to be able to understand them and implement the dedicated event.

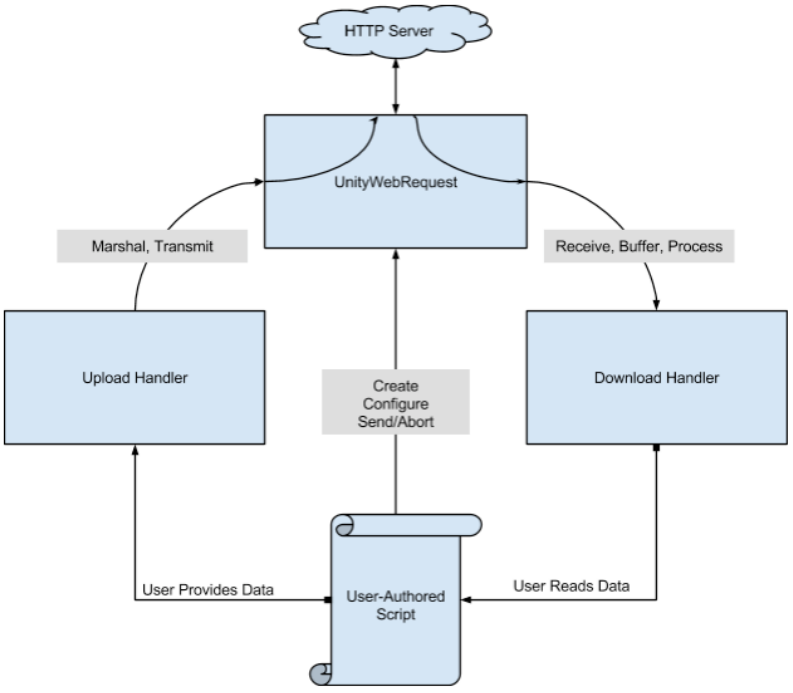
Dictation allows users to dictate text, which is not included in the predefined phrases. It is particularly useful for writing messages, notes, or entering data while driving. Dictation recognizer listens to speech inputs and attempts to determine what phrase was uttered.

At its core, speech recognition software works by breaking down a speech recording into individual sounds. This technology then analyses each sound and uses an algorithm to find the most probable word fit for that sound.

3.6 Unity Web Requests

Unity provides a modular system called UnityWebRequest, which is used for composing HTTP requests and handling HTTP responses. The goal of this system is to allow unity games and apps to interact with back-ends of the web browser. It is also able to support high demand features such as handling of large data transfers, full control of request settings and streaming data uploads. The systems consists of two layers of external API requests. A Low-Level API (LLAPI), which provides maximum flexibility (for more advanced users), and a High-Level API (HLAPI), which wraps the LLAPI and provides a convenient interface for performing more common operations.

The architecture of UnityWebRequest ecosystem divides an HTTP transaction into three operations. Those operations include the supply of the preferred data to the server, receiving the corresponding data from the server, and controlling the HTTP flow like error handling or redirecting 3.1.



Source: https://docs.unity3d.com/Manual/UnityWebRequest.html

Figure 3.1: Unity Web Request System Architecture

In order to provide a better interface for advanced users, each one of these operations are controlled by their own objects which consist of an UploadHandler, a DownloadHandler and a UnityWebRequest. An UploadHandler object is responsible for the transmission of data to the server. DownloadHandler objects obtain, buffer and postprocess the data that are received from the server. UnityWebRequest objects are responsible for the management of the other two objects, while also handling HTTP flow control, which means that errors and redirect information are stored there. Also, this is the object where the URLs of the server as well as the custom headers are defined.

3.7 VR Integration with HTC Vive Pro

The HTC Vive Pro is a high end VR headset used in this project for its advanced features and compatibility with eye-tracking technologies as mentioned in HMDs analysis

in Section 2.2.3. In order to create or transition a unity project into a VR experience, specialized tools, libraries and VR interaction components are needed. These components include VR-specific libraries and tools, which enable VR input and spatial interaction in 3D space, and and facilitate VR rendering and performance.

The necessary components for seamless VR experience and interaction, manage various aspects of VR integration. Some of those aspects include headset tracking, controller interaction, user interface optimizations and interaction systems, specialized for immersive VR experiences. Specifically in this thesis, the HMD (HTC Vive Pro) integration with Unity environment, plug-ins like OpenXR and SteamVR platform have been used used and will be analyzed below.

3.7.1 OpenXR

OpenXR is an open-source plug-in, developed by Khronos Group Inc that provides access to XR platforms and devices. It aims to simplify AR and VR development by allowing developers to access a wide variety of AR/VR devices. Unity OpenXR, as a set of tools and APIs provided by Unity, helps developers to create immersive experiences in AR and VR by connecting the platform with the devices. This plug-in provides an API that helps developers create XR applications and be able to use them in various XR devices and platforms. Essentially, OpenXR allows developers to make their applications compatible with various XR hardware, like HMDs and AR glasses, without the need of integration for each one of the devices or platforms used separately. It promotes collaboration between XR ecosystems, by providing a common framework that handles XR features such as tracking, input handling, device interactions and rendering.

3.7.2 XR Plug-in Management

In order to be able to use OpenXR, developers need to upgrade a project, since it is a plug-in in Unity's XR plug-in architecture. Unity XR plug-in Framework is a unified framework that enables direct interactions for multiple platforms. It consists of an API that exposes common functionality across the platforms that Unity supports and enables XR hardware and software providers to develop their own Unity plug-ins. Unity recommends using the XR Interaction Toolkit, easily downloaded from Unity Package Manager, which is used for input and interactions in AR/VR applications.

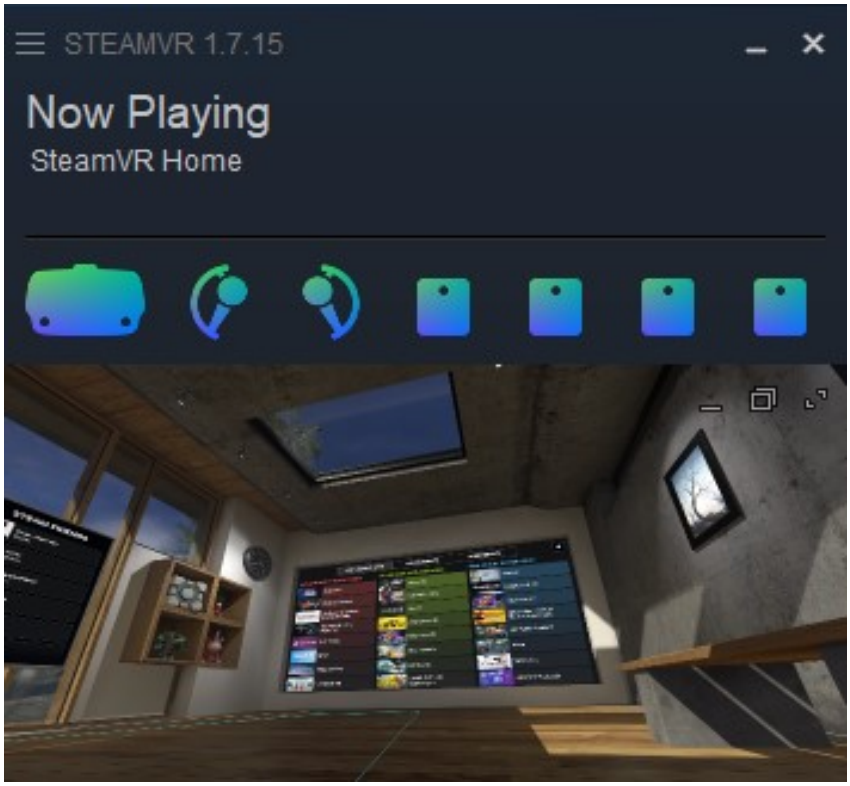
The XR Interaction Toolkit package, is a high-level interaction system, which provides a framework that makes 3D and UI interactions available from Unity Input events. The core of this system is a set of base Interactor and Interactable components, along with an Interaction Manager that merges the two components together. This toolkit is specifically developed for VR applications, which allows developers to use a wide range of tools and components essential for implementing VR interactions. These tools include a wide range tasks that allow user interactions, like object manipulation and menu navigation, as well as engaging spatial experiences like initiating VR animations. This system allows developers to use a wide range of pre-arranged prefabs, scripts and other resources to implement features like hand gesture recognition and object interaction.

3.7.3 SteamVR

Developed by Valve Corporation, SteamVR provides a platform for a wide range of immersive VR experiences. SteamVR is a full-featured, 360-degree room-scale VR experience that provides a VR interface (SteamVR Home) 3.2, as a desktop Home page, for many HMDs. Essentially, it is an API, or a suite of systems that allow VR headsets to play games and other VR applications. It allows users to navigate through a VR graphical interface to access any apps or options the user needs. SteamVR is also responsible for the connectivity between the HMD and its bases, or controllers. It displays when the bases are able to see the HMD and its controllers, to show the users when a connection problem appears. Also, users need to set up their position inside a game or app through SteamVR's Room setup, in order to be able to navigate with more accuracy about the height of their head, and their movements. Lastly, in order to implement eye-gaze interactions, users need to calibrate their eye movements, via VIVE Pro Eye, to make sure they have accurate eye-tracking.

3.7.4 XR Origin

XR Origin in Unity is an essential component used in XR Interaction Toolkit designed for VR and AR applications. It provides a setup for position and orientation tracking of the player in virtual environment, and can support both stationary and room-scale experiences. This component, manages the player's head and hand tracking by maintaining



Source: https://www.uploadvr.com/steamvr-update-1-7/

Figure 3.2: SteamVR Home

a reference point for the XR camera. It ensures an accurate and consistent spatial relationship between the user's movements in the physical and virtual world, enhancing immersion and minimizing motion sickness. Using XR Origin, developers are able to align the user's physical location and orientation with that in the VR environment and create effective movements and interactions during the VR experience.

3.8 Eye Tracking Software

Applications that enable eye-gaze interactions need to implement eye-tracking accurately for a better user experience, effective UIs, and less user fatigue. In order to achieve that, software that has been used include TobiiXR SDK and VIVE SRanipal SDK, to achieve eye-tracking integration into Unity.

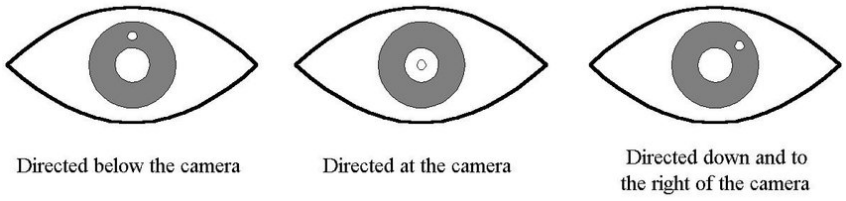
3.8.1 Tobii XR SDK

Tobii provides tools and libraries for integrating eye-tracking capabilities into Unity applications. It calculates gaze direction and fixation points, allowing for precise interaction based on where the user is looking. TobiiXR API is used to access eye-tracking data through script in order to implement gaze-based interactions. Interaction logic based on gazed data can also be implemented, like checking if the gaze ray intersects with a virtual object in order to perform actions like highlighting or selecting the object.

3.8.2 VIVE SRanipal SDK

The VIVE SRanipal SDK is designed to work with the HTC Vive Pro eye-tracking hardware, in order to provide eye-tracking data to enable intuitive interactions. SRanipal API is used to access eye-tracking data, such as gaze direction and origin, as well as eye opening and closing. These eye-tracking data output are a combination of hardware and software in order to accurately track user's gaze.

3.8.3 Tracking the eye-gaze



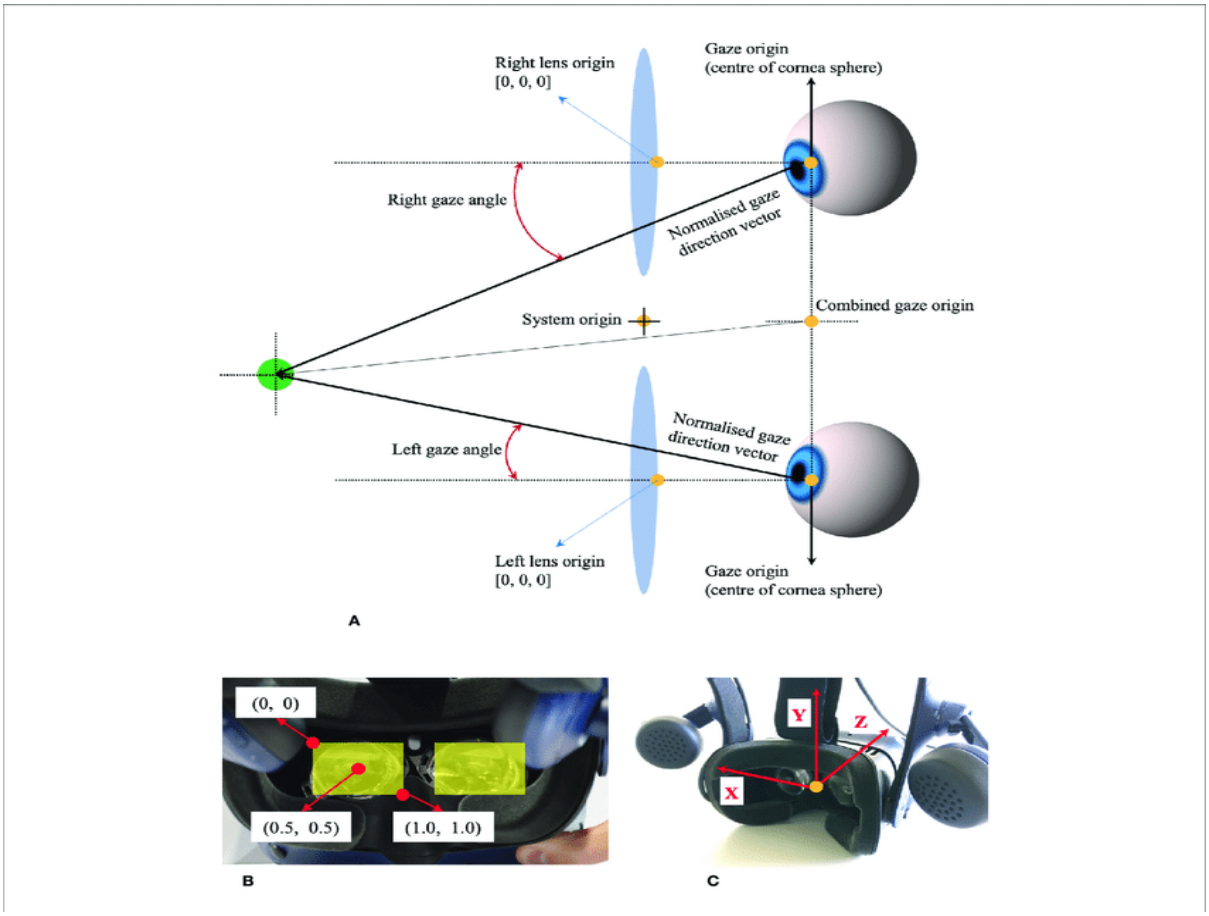
Source: https://www.researchgate.net/figure/Using-the-corneal-reflection-pupil-centre-method-to-determine-point-of-regard-when-the_fig1_322406088

Figure 3.3: Corneal Reflection

Hardware components include infrared illuminators which emit invisible infrared light to illuminate the eyes and cameras that capture high-speed images of the user's eyes reflecting the infrared light. Afterwards, the images are processed in order to identify key features like the center of the pupil and reflections of the corneal. Pupil detection happens through detecting the dark circular shape of the pupil which helps to determine

the center of the eye. Corneal Reflections are visible by detecting infrared light on the cornea, which appears as bright spots in the images 3.3.

Furthermore, a 3D eye model is created in order to estimate the eye-gaze. This model calculates the gaze direction, which accounts for properties of eye anatomy, like the curvature of the cornea and the position of the pupil 3.4 (Figure A). The system then, calculate the gaze vector through analyzing the relative positions of the pupil center and corneal reflections 3.4 (Figure B). Then, a point of gaze is determined, by finding where the gaze vector collides with a surface (2D or 3D) in the VR environment 3.4 (Figure C).



Source: https://www.researchgate.net/figure/Coordinate-system-of-HTC-VIVE-Pro-Eye-based-on-the-manual-of-SRanipal-SDK-A_fig1_346058398

Figure 3.4: (A) Coordinate system of eye tracking on VIVE Pro Eye. (B) Coordinate system of pupil position data from user's view. (C) Coordinate system of gaze direction vector from user's view.

3.8.4 3D collision detection

To mathematically check if a gaze ray penetrates an Axis-Aligned Bounding Box (AABB), a typical technique used is called the Ray-AABB Intersection. A bounding box is simply a bounding parallelepiped whose faces and edges are not parallel to the basis vectors of the frame in which they're defined. This test determines whether a given ray intersects with the bounding box by calculating the intersection along each axis (x, y, z) and checking if the intersection intervals overlap. The given ray is defined by an origin point O and a direction vector D and an AABB is defined by minimum Bmin and maximum Bmax points. The mathematical equation of the ray can be represented parametrically as $R(t) = O + tD$, where t is a scaling parameter that moves the point along the ray. The Intersection Intervals need to be calculated for each axis. The key idea is to find the values of t where the ray intersects the segments of the AABB, which refers to the space between the min and max boundaries for each axis. The equations for t values for each axis are:

For x-axis: $t_{minx} = \frac{B_{minx} - O_x}{D_x}$ $t_{maxx} = \frac{B_{maxx} - O_x}{D_x}$

For y-axis: $t_{miny} = \frac{B_{miny} - O_y}{D_y}$ $t_{maxy} = \frac{B_{maxy} - O_y}{D_y}$

For z-axis: $t_{minz} = \frac{B_{minz} - O_z}{D_z}$ $t_{maxz} = \frac{B_{maxz} - O_z}{D_z}$

If the direction component Di (i is for x,y or z) is zero, the ray is parallel to the corresponding axis and if the origin O lies outside the AABB segment, the the ray intersect the collider. The ray intersects the AABB if there exists an overlap in the intersection intervals across all axes. This overlap can be found by computing the $t_{min} = max(t_{minx}, t_{miny}, t_{minz})$ and $t_{max} = min(t_{maxx}, t_{maxy}, t_{maxz})$.
The condition for intersection is $t_{min} \leq t_{max}$ and $t_{max} \geq 0$. If $t_{min} > t_{max}$ the intervals do not overlap, and the ray dot not intersect the AABB. If $t_{max} < 0$, the intersection happens behind the ray's origin, so there is no visible intersection.

This method is computationally efficient and works well for real-time applications like gaze-based object interaction in VR/AR systems, where it's critical to quickly determine if the user is looking at an object within the 3D space.

Chapter 4

Users View

4.1 Introduction

In this chapter, the view of the user's experience 4.1 within the application will be presented, along with the interaction capabilities during playtime. Specifically, the main menu will be explained, and the various interactions, available to the user, through the UI, will be illustrated through diagrams, and shown through pictures.

Figure 4.1: User's View

4.2 Application Main Menu

When the application starts, a dark screen with 2 interactable buttons appears 4.2. In this panel, the user has to interact using the mouse of the computer. These options, both initialize the main scene of the project, with the difference that the first option is the default application simulation. The second option simulates the evaluation process, which compared to the default scene, initializes some extra elements that help with metric evaluation.

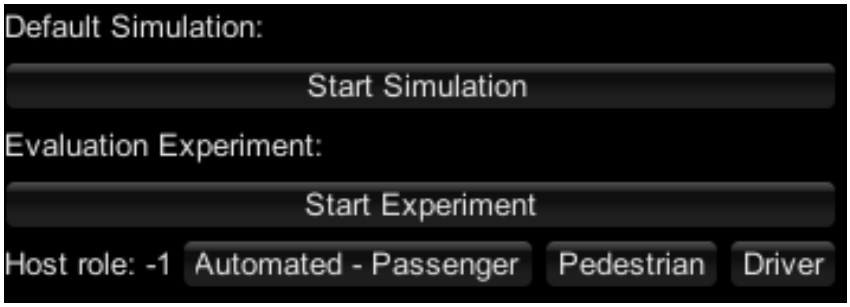


Figure 4.2: Application Main Menu

When user chooses one of the two options, the view is directed to the next panel, which only has one option, to start the simulation 4.3. The use of this second panel, between the choice of the preferred simulation and its runtime, is to set all of the simulation parameters ready, and load the scene. Once the Start Simulation Button appears, the scene is set and the preferred parameters are ready to continue with the simulation after the button is pressed.

Figure 4.3: Start Simulation Panel

4.3 Interconnection between Windows

The interface includes many different windows, each one containing multiple interactive elements. In this section, the potential interactions in each one of these windows, and the interconnection between them, will be described. The UI consists of the main window and 6 additional windows, each serving its own purpose and functionality. The diagram below 4.4 shows the possible interactions between the windows.

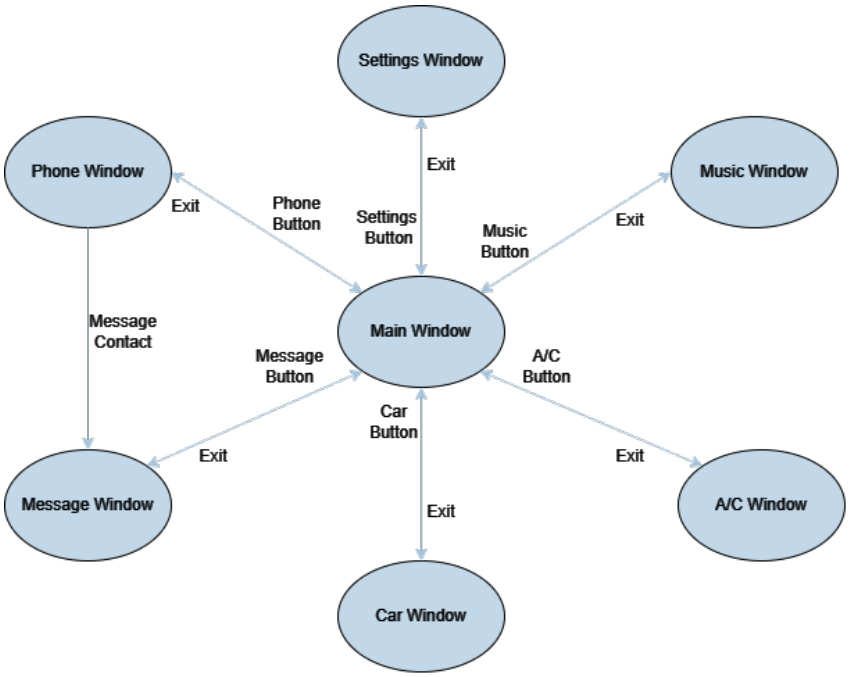


Figure 4.4: Interconnection between UI windows

4.3.1 UI Main Window

The main window of the UI, which is the menu of the interface, consists of 2 informative - non interactable- widgets and 6 interactable buttons 4.5. Those widgets include a weather forecast interface, right side of the main window, which is designed to show the live weather forecast for the capital of Greece, Athens. Also, there is an informative widget on the left side of the main window, which includes information about the vehicle, like

the traveling speed (in km/h), the remaining fuel range (in km) , and if the vehicle is driving autonomously or not.

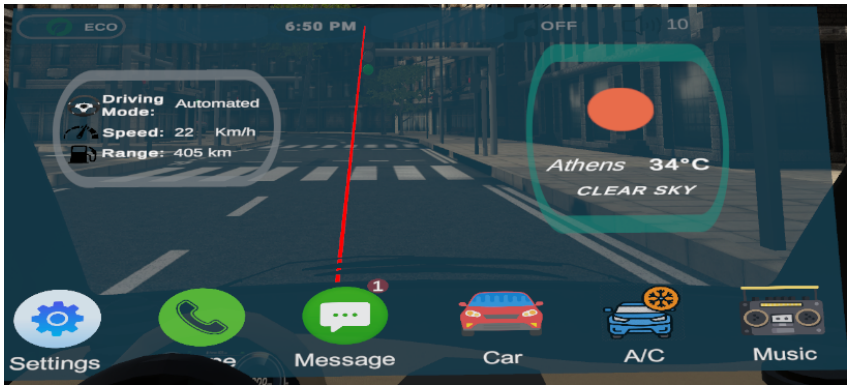


Figure 4.5: User Interface Main Window

The interactable buttons in the main menu can access every possible window of the UI. They are placed in the bottom side of the screen and they include a Settings Button, a Phone button, a Message Button, a Car Button, an A/C button and a Music Button. The possible interactions in each one of those windows will be explained below. On the top side of every window, there is a tool bar for quick access and information about many different aspects of the application.

4.3.2 Settings Window

Settings window consists of some main features that handle the way the user can interact with the interface 4.6. More specifically, it includes 3 toggle switches, 2 of which affect the choice of the eye selection technique. First toggle is about the dwell time selection technique which is deactivated by default. The user can activate it, through the Settings window or a by using the corresponding voice command. Activating that toggle will allow the user to interact with the interface through staring in any interactable object of the scene for half a second.

The second toggle activates and deactivates the eye blink selection technique. It is activated by default and can be deactivated through this canvas or by using the corresponding voice command. The third toggle is also activated by default and is the toggle

Figure 4.6: Settings Window

responsible for the activation and deactivation of voice commands. It can also be deactivated using a voice command but the only way to activate it is through the Settings window. Last, there is a single button on the right side of the settings window, which is responsible for minimizing the UI, when pressed. When that happens, a small button appears in the bottom left side of the windshield, leaving the car windshield viewing area empty of virtual elements 4.7. The UI can also be minimized by the voice command "screen off", and maximized by saying "screen on".

Figure 4.7: Minimized Interface

4.3.3 Phone Window

This window is responsible for the user's interaction with contacts and calls 4.8. It consists of 2 different panels, one for Recent Calls and one for Contacts. The user can choose the panel to interact with from the left side of the window. The chosen panel is highlighted with a white outline, so that the user can constantly be informed about the panel in used.

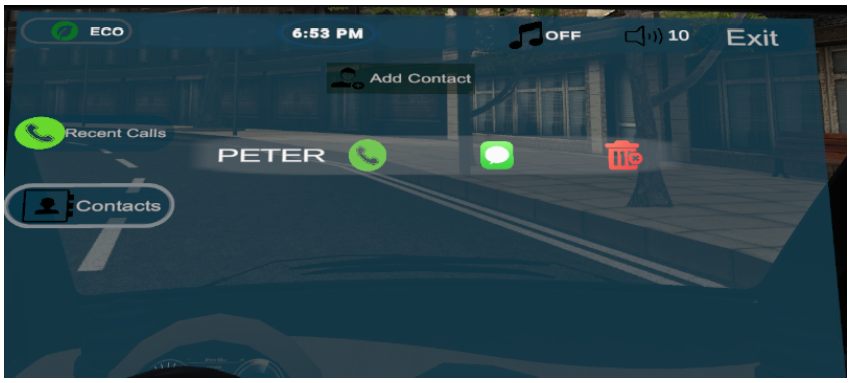


Figure 4.8: Phone Window. Contacts Panel

In the Contacts panel, the user can add new contacts by selecting the button "Add Contact". Then, an Input field appears, along with a Save Button 4.9.

Figure 4.9: Contacts Panel. Add New Contact

The user needs to speak in order to write a name for the new contact and then press the Save Button. Then, a new contact is added to the Contacts Panel. Users can call,

message or delete any of those contacts through the contact panel. If the user calls a contact, a calling panel appears in the bottom left side of the Phone window and a new outgoing recent call appears in the log of recent calls panel. The Contact Panel is initialized with one contact by default.

The Recent calls panel 4.10 saves the incoming and outgoing calls during the UI usage. Whenever the user makes a phone call, a new outgoing call is added in the recent calls panel. In each recent call instance in the panel, the user has the choice to call or message the appearing name of the Recent call.

Figure 4.10: Phone Window. Recent Calls Panel

The user can also make a phone call by saying the keyword "call" and then the name of the contact to be called. This can only happen with already existing contacts, otherwise no call can and will take place.

4.3.4 Message Window

In this window, users can receive and send messages to and from others. This window has two different panels, one for inbox messages and one for outbox messages 4.11. Users can change panels from the inbox and outbox buttons appearing in the left side of the window, and they know with which of those two panels they interact at any moment since it appears highlighted with white outline.

In both of those panels, there is a button on the top middle part of the screen which users can select to send new messages. When a user selects the "New Message" Button,

Figure 4.11: Messages Window. Inbox Panel

the new message panel appears and the user has to select one name from the contact list, as the receiver of the message. If the contact list is empty, the user cannot send a new message through the Message Window, but only through the recent calls window. That is possible, because from the recent calls, users can directly send messages to the contacts that appear, without the need for the contact choosing step, during the new message procedure.

When the users have chosen the contact they want to send their message to, the receiver's name is registered in the receivers position, and the send button appears. After that, the Message Input Field is activated, and the user is called to speak in order to write down the context of the message. Then, the user just has to select the "Send Message" Button in order for the message to be sent. When that happens, the new message panels closes and the outbox panel appears, showing the new sent message.

Each message, either in the inbox or the outbox panel, has a button that can open it. Opening a message gives the user the choice to either forward or delete it. Deleting a message leaves the open message panel and goes back to the panel that the message was opened from (inbox or outbox), without this specific message existing anymore. When a user forwards a message, the same contact choosing procedure as in the new message case starts. If the are no contacts available, the user is informed about it with a warning and cannot continue in the forwarding procedure. If the contacts are more than zero, the users has to choose one of the as the receiver of the forwarded message. Then the name of the contact appears in the receiver spot and the send button appears. If the user

selects the send button, the message is being sent and the view is being directed back to the panel it came from.

While writing or forwarding a message, the user has the choice to leave the panel. If this is done, the UI is developed to hold the receiver and message context information as a draft message. The next time the user opens the new message, the previous unsent new message will be intact, as long as the contact still exists.

4.3.5 Car Window

The car window is responsible for some specific information about the car. Specifically, it includes 3 different panels, the Preset panel 4.12, the Tire Pressure panel and the Car Maintenance panel. Each of the panels can be chosen by selecting the corresponding button in the left side of the Car window, and the panel being used is highlighted.

Figure 4.12: Car Window. Engine Presets Panel

The Preset panel consists of 3 different buttons, Eco, Normal and Sport. By default, Eco mode is chosen, but the users can change that through the Car Window. In the Tire pressure panel, a 2D model of the car with the number in each of the four tires appears. This number corresponds to the pressure of each tire measured in BAR. The Maintenance panel consists of a button which checks the car for malfunctions and informs the user with the search results.

4.3.6 A/C Window

This Window is responsible for handling the air conditioning unit of the vehicle 4.13. The window consists of an on/off toggle switch, 2 Buttons for higher or lower temperature. The temperature can also be controlled through voice commands. Also, there are 2 buttons that control the air flow, one of them letting the air come from the outside of the car, and the other one recycling the air inside the car. Last, there are five buttons that control the direction of the air as shown in the image below.

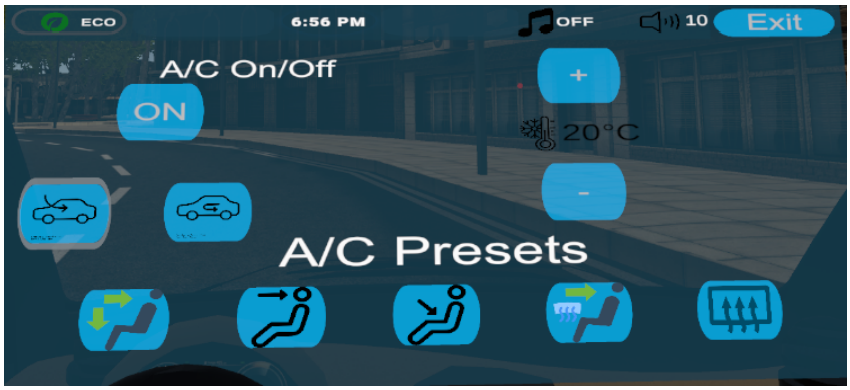


Figure 4.13: A/C Window

One of the two air flow buttons is always highlighted so the users are constantly aware where the user is coming from. Also, one of the five air direction buttons is highlighted whenever the A/C is on. If one of the five buttons is selected, the A/C is activated and the buttons gets instantly highlighted. If the On Button is pressed, the first of the five air direction presets is selected and highlighted by default.

4.3.7 Music Window

The music window is responsible for handling the music played inside the car 4.14. In this window, there is a On/Off Toggle Switch that opens or closes the music. On or off can also be handled through voice commands. On the right side of the music window there are two buttons that control the volume of the music, which can also be handled by using voice commands.

Figure 4.14: Music Window

In the bottom side of the window, there are five buttons that represent five radio stations. By pressing one of these buttons, it is highlighted and the music of that station starts playing. If the On button is pressed, the default radio station is FM1, if it has not been changed. If the radio station is changed, the next time the on button is selected, the system will remember which was the last selected station. This can also be controlled with voice commands if the user say "next" or "previous station". Last, there is the mute button which mutes the volume if it is on, or unmutes it if it is off.

4.3.8 Toolbar

On the top side of the screen, there is a horizontal, thin toolbar screen, working as an informative top side bar. This toolbar is open in all windows and all panels, always showing the preferred information to the user.

On the left side of the toolbar, the first information shown, is the car engine Preset. Set default at Eco, but can be changed through the Car window, it is an important information to be constantly shown on the toolbar of the UI. Next, there is a an empty space, serving as a calling panel. This panel is used whenever a call takes place, informing the user if the call is incoming or outgoing, writing the name of the caller or receiver of the call. In this panel, there also is an "end call" button which the user can select in order to end both incoming and outgoing calls.

Next, in the middle of the toolbar, there is a clock which is constantly writing the time of Athens (UTC + 3). On the right side of the clock, there is another empty panel

which serves as a notification panel. More specifically, this panel sends notification of incoming messages to the user, writing the name of the sender and a small part of the message context. Also, this is the panel that warnings coming from the actions of the AV appear. Whenever the car is about to turn or break, a respective message appears in this panel, to warn the user for the action that the AV is about to do.

On the right side of the notification panel, there is a panel that shows information about the music inside the car. Specifically, it shows if the music is on or off and if it is on, which radio station is currently streaming. Also, it shows the volume of the music, either if it is on or not, and lastly shows if the sound is muted or not.

4.3.9 Help panel

Outside of the UI screen, there is a panel on the right side of the steering wheel, mentioning the available voice commands 4.15. This panel can be opened and closed using voice commands, saying "help" or "close help" will open or close it respectively. It is activated by default when the application starts.

Figure 4.15: Help Panel

Chapter 5

Implementation

5.1 Introduction

Chapter 5 focuses on explaining the implementation details of the project, providing insights of how various components of the project were developed and integrated to create the final application. From the implementation of the immersive Virtual Environment to the developing of the User Interface, and all the in between stages, this chapter offers a thorough technical analysis of the project aspects. Most of the script will analyzed in detail as well as screenshots of some script will be provided.

The project analysis consists of a break down of 5 stages, each one implementing a crucial aspect of the application's functionality. These stages include the Creation and Loading of the Simulation Environment, the Autonomous Vehicle integration, the Eye Gaze Interaction, the Voice Recognition and the User Interface Implementation. Each of these processes will be addressed individually, explaining their development stages, as well as their interconnection with the rest of the project.

The project to be explained has been entirely developed within the Unity Game Engine using C# scripting. From loading the virtual environment to interacting with the UI, most of the functionalities of the project have been implemented through scripts. These scripts are responsible for implementing the preferred behavior of each virtual element or interaction between elements withing the application. Even though Unity's Scene Editor plays a crucial role in the interconnection between project components, scripting allows for parameterized elements which open the door for future updates with less changes.

5.2 Simulation Environment

As explained in the Research Overview Chapter 2, the Virtual Environment of a Simulator that scopes for realistic behavior and testing, needs to be as immersive and realistic as possible. That being said, the creation of the environment, including every aspect of the scene, need to surpass a standard level of visual resolution and high-quality 3D graphics in order to seem realistic. That can happen by choosing and combining detailed 3D models for every visible virtual component, including the buildings, roads, cars, pavements, lights and everything. Scenes filled with many virtual components tend to be heavy computing tasks, so loading needs to be handled efficiently.

5.2.1 Scene Creation

Figure 5.1: Main Scene Virtual Environment

The main scene of the project, is one of the most important factors when it comes

to fidelity in realistic simulations. Each part of the virtual environment needs to be of high quality in order to combine into a realistic outcome. That is the reason why a predesigned driving simulator **??**, already used for research in the past, has been found online 5.1. The simulator includes a realistic city design with different buildings, roads, traffic lights and car prefabs. The change needed, towards using this environment for the project needs, was converting it to a VR ready environment, specifically using the HTC Vive Pro HMD.

The first step to converting a desktop simulation to a VR ready environment is to change the rendering camera. Having the XR package installed, Unity offers the option to create an object called VR Origin. This object, essentially represents the player when the application is running. This object includes a camera with an offset to enable stereoscopic rendering and ensure an immersive and realistic VR experience. Additionally, XR Origin aimed to be used for devices like the Vive Pro HMD that implement eye tracking, contain the object EyeGazeOrigin. This GameObject represents the place of the user's eyes and needs to be initialized in the position of the XR Origin camera, since that is where the head and eyes of the user will be initialized inside the virtual environment.

The XR Origin's camera also implements tracking options from the HMD, allowing the camera to move in sync with the user's head movements. This ensures that the virtual environment updates in real-time, providing a seamless and immersive experience as the user navigates through the application.

AI Car Circuit

The Autonomous driver of the project is programmed to follow a specific circuit within the roads of the city, through following a predefined route of waypoints placed on the city roads. This route is created by placing waypoint objects in the main scene of the project which are automatically inserted in a list of waypoints. These objects need to be placed in a way that their order within the list (from first to last) reflects the order that the car needs to follow in order to complete the circuit. These waypoints are empty gameObjects without a mesh renderer, only having a box collider and a script named SpeedSettings attached 5.2. SpeedSettings script has public variables like speed, acceleration, causeToYield boolean and others that describe the behavior of the vehicle after it touches each waypoint.

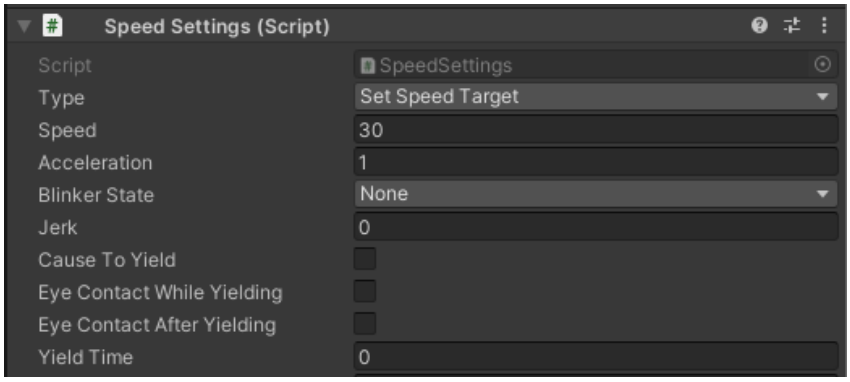


Figure 5.2: SpeedSettings Variables

The collider of these objects essentially calls the OnTriggerEnter() method 5.3 whenever the vehicle touches any of these waypoints. Then, the AICar script which will be explained later, derives information held by the waypoint that has just been touched, which holds information about the target speed and acceleration of the vehicle, in case the causeToYield boolean is false. If this boolean is true, the car is set to perform a full stop, gradually setting its speed to 0, and setting the target speed and acceleration to a specific value that will be used after finishing Yield coroutine that counts the time interval between the full stop and resetting vehicle's speed.

```
if (other.GetComponent<SpeedSettings>().causeToYield)
{
    set_speed = 0;
    set_acceleration = other.GetComponent<SpeedSettings>().brakingAcceleration;
    speedAfterYield = other.GetComponent<SpeedSettings>().speed;
    accAfterYield = other.GetComponent<SpeedSettings>().acceleration;
    yieldingTime = other.GetComponent<SpeedSettings>().yieldTime;
    shouldYield = true;
    state = CarState.BRAKING;
}
else
{
    set_speed = other.GetComponent<SpeedSettings>().speed;
    set_acceleration = other.GetComponent<SpeedSettings>().acceleration;
}
```

Figure 5.3: OnTriggerEnter() method

5.2.2 Loading the Environment

The first two OnGui screen layouts that appear during project runtime, as explained in Chapter 4 4, are responsible for setting the simulation parameters. More specifically, when the application starts, two OnGui buttons ("Start Simulation" and "Start Experiment") 4.2 appear on the screen. Both buttons initialize the same main scene with the difference that the experiment option sets the value of experimentOn boolean true. When that boolean is true, the simulation experiment takes place. This experiment initializes some visual indicators that guide users to select specific buttons in a specific order, completing tasks that measure the time needed until completion, while counting the total errors during that procedure. The experiment procedure will be explained thoroughly in the next chapter of the thesis.

When the user selects one of the two OnGui buttons, the main Scene of the project starts loading along with the predefined circuit. When it does, another black screen with an OnGui button "Start Game" appears, which immediately starts the game when pressed. This way of loading the scene allows the project to initialize without immediately loading heave assets or running complex scripts. When one of the two buttons is pressed, the main scene is loaded asynchronously, allowing the application to load necessary resources without freezing the UI or causing noticeable frame drops. The second black screen ensures that everything in the main scene is fully loaded and ready before starting the simulation.

Gradual loading scenes in a staged manner prevents Unity from overwhelming the system with asset loading, initialization of scripts, physics, and other heavy processes all at once. This approach avoids Unity's typical synchronous loading, which can cause the application to freeze temporarily, especially if the main scene has a lot of assets or complex scripts as it does in this project. When a complex scene starts loading, user might experience initial stutters, low frame rates, or see assets appearing into view as they load. This can be prevented with asynchronous OnGui load. Also, all scripts, including those triggered in Start() and Awake() methods, execute as soon as their objects are loaded. If loading takes longer, scripts might start running with missing references or uninitialized states. By splitting the loading process into stages, it is ensured that all resources are loaded gradually, optimizing memory usage and processing power. Scripts that control the simulation will only start once all dependencies are loaded, reducing runtime errors.

Once the "Start Game" button is pressed, the simulation starts, placing the car in a predefined position, heading to the first waypoint, which is already initialized.

5.3 Autonomous Driver

The functionality of the Autonomous vehicle derives from the combination of 3 scripts that set up the circuit the car will follow, check for the correct progress while following it, and integrating the movement of the vehicle between the waypoints of the circuit. These 3 scripts are WaypointCircuit, WaypointProgressTracker and AICar that handle the described functionalities respectively.

During the spawn of the car prefab, Awake() method of WaypointCircuit script is triggered, which checks if there are waypoints in the waypointList. If there are more than one waypoints, CachePositionsAndDistances() method 5.4 is called. This method stores the positions of the waypoints and the distances between them in arrays in order to optimize lookup speed during runtime. The number of the waypoints is also recorded, which is essential for looping through waypoints during the AV's operation, through the WaypointProgressTracker script. After the initialization of the waypointCircuit, Reset() method of WaypointProgressTracker script is called in order to reset the waypoint tracker to its initial state, setting the car's progress along the route.

```
private void CachePositionsAndDistances()
{
    // transfer the position of each point and distances between points to arrays for
    // speed of lookup at runtime
    points = new Vector3[Waypoints.Length + 1];
    distances = new float[Waypoints.Length + 1];

    float accumulateDistance = 0;
    for (int i = 0; i < points.Length; ++i)
    {
        var t1 = Waypoints[(i) % Waypoints.Length];
        var t2 = Waypoints[(i + 1) % Waypoints.Length];
        if (t1 != null && t2 != null)
        {
            Vector3 p1 = t1.position;
            Vector3 p2 = t2.position;
            points[i] = Waypoints[i % Waypoints.Length].position;
            distances[i] = accumulateDistance;
            accumulateDistance += (p1 - p2).magnitude;
        }
    }
}
```

Figure 5.4: CachePositionsAndDistances() method

Update() method of WaypointProgressTracker script is called every frame and updates the car's progress along the waypoints based on its speed and position. The speed is calculated based on the distance traveled since the last frame (lastPosition-transform.position) divided by the time elapsed. Math.Lerp is a linear interpolation function that is used to gradually adjust the speed to match the given speed by the current waypoint smoothly. Next, to determine the waypoint target position and rotation, GetRoutePoint() method of WaypointCircuit script is called.

GetRoutePoint() method 5.5 is used so the AV can follow the route by retrieving specific route points and positions based on the distance traveled along the route. This method calculates the current position of the vehicle (p1) and a position slightly ahead of p1 (p2, which has an offset of 0.1 units). These variables (p1 and p2) are returned by GetRoutePosition() method, which calculates the exact position on the route for a given distance. Specifically, the method smooths the route using linear interpolation between waypoints using the Mathf.Lerp function. Finally, the GetRoutePoint() method returns a RoutePoint object, which includes the position p1 and the direction from p1 to p2.

```
public RoutePoint GetRoutePoint(float dist)
{
    // position and direction
    Vector3 p1 = GetRoutePosition(dist);
    Vector3 p2 = GetRoutePosition(dist + 0.1f);
    Vector3 delta = p2 - p1;
    return new RoutePoint(p1, delta.normalized);
}
```

Figure 5.5: GetRoutePoint() method

Returning to WaypointProgressTracker's Update() method 5.6, the position of the target is calculated using GetRoutePoint() method, based on the progressDistance plus some additional factors. These factors handle the number of the substeps needed between the waypoints, in order to smooth the vehicle's movement, based on the current speed of the vehicle. Also, using GetRoutePoint(), the rotation of the target is calculated, so the vehicle's rotation aligns the forward direction to match the path's direction at the

calculated point. The purpose of this calculation substeps is to ensure the car rotates smoothly along the path, aligning itself with the intended direction of travel.

```
private void Update()
{
    if (progressStyle == ProgressStyle.SmoothAlongRoute)
    {
        // determine the position we should currently be aiming for
        // (this is different to the current progress position, it is a certain amount ahead along the route)
        // we use lerp as a simple way of smoothing out the speed over time.
        if (Time.deltaTime > 0)
        {
            speed = Mathf.Lerp(speed, (lastPosition - transform.position).magnitude/Time.deltaTime,
                               Time.deltaTime);
        }
        target.position =
            circuit.GetRoutePoint(progressDistance + lookAheadForTargetOffset + lookAheadForTargetFactor*speed)
                .position;
        target.rotation =
            Quaternion.LookRotation(
                circuit.GetRoutePoint(progressDistance + lookAheadForSpeedOffset + lookAheadForSpeedFactor*speed)
                    .direction);

        // get our current progress along the route
        progressPoint = circuit.GetRoutePoint(progressDistance);
        Vector3 progressDelta = progressPoint.position - transform.position;
        if (Vector3.Dot(progressDelta, progressPoint.direction) < 0)
        {
            progressDistance += progressDelta.magnitude*0.5f;
        }


        lastPosition = transform.position;
    }
```

Figure 5.6: WaypointPogressTracker Update() method

Afterwards, the progress point of the route is calculated, with the use of progressDelta variable, which continuously checks if the current position of the car has surpassed the target position of the next substep. Constantly updating the lastPosition variable with the current position of the vehicle, can inform the AD if the current position of the vehicle has surpassed the progressPoint so the process described above starts again for the next substep.

```
void FixedUpdate()
{

    // Every physics calculation involves the orientation and speed of the object.
    Vector3 new_position = transform.InverseTransformPoint(target.position);
    float psi = Mathf.Asin(new_position.x / (Mathf.Pow(new_position.x * new_position.x + new_position.z * new_position.z, 0.5f) + 0.001f));
    //  Update all required informations
    rotationAxis.rotation = Quaternion.Euler(0, rotationAxis.rotation.eulerAngles.y, 0);                    //heading

    //  Change of Ambient Traffic rotations based on current heading and target position
    theRigidbody.angularVelocity = new Vector3(0f, psi * turn_rate_degree * Mathf.PI / 360f, 0f);
```

Figure 5.7: AICar FixedUpdate() method

In AICar script, FixedUpdate() method 5.7 handles the AI car's behavior including steering, speed adjustments, braking and resetting its state when it comes to a stop.

FixedUpdate() methods are crucial in Unity's physics engine because it is called at a fixed time interval, making it ideal for handling physics-related calculations such as velocity, rotation and forces. In this specific script, it uses the car's current position and speed relative to target point to determine the necessary adjustments.

First, the target's world position is converted into the car's local coordinate system. The target is not the next waypoint but the next substep which has been calculated by the Update() of the WaypointProgressTracker script. Then, psi variable is calculated, which is the steering angle that the car needs to adjust towards target. It measures the angle between the car's current forward direction (new_position.z) and the line connecting the car to the target. This line is Euclidean distance (hypotenuse) from the car to the target position using the Pythagorean theorem. Then, in every iteration of the FixedUpdate(), the rotation of x and z axes are reset, keeping the rotation only around the y-axis (yaw). Then, the angularVelocity of the rigidBody is set in order to control how quickly the car rotates towards the target waypoint, to ensure that the car can smoothly follow the circuit.

```
// This statement is applied when the car is just driving.
if ((braking == false) && (reset == false))
{
    if (jerk != 0)
    {
        acceleration = set_acceleration;
        t += Mathf.Abs(jerk) / Mathf.Abs(set_acceleration) * Time.fixedDeltaTime;
        pitch = acceleration / 3 * HasPitch;
        model.localRotation = Quaternion.Slerp(model.localRotation, Quaternion.Euler(-pitch, 0, 0), 0.5f);
    }

    if (set_acceleration > 0f && set_speed > speed || set_acceleration < 0f && set_speed < speed)
    {
        speed = speed + set_acceleration * Time.fixedDeltaTime * conversion;
        pitch = acceleration / 3 * HasPitch;
        model.localRotation = Quaternion.Slerp(model.localRotation, Quaternion.Euler(-pitch, 0, 0), 0.5f);
    }

    if (acceleration != 0 && Mathf.Abs(speed) < Mathf.Abs(set_speed) * (1 + tolerance) + tolerance * 10 && Mathf.Abs(speed) > Mathf.Abs(set_speed) * (1 - tolerance) - tolerance * 10)
    {
        jerk = 0;
        speed = set_speed;
    }

    theRigidbody.velocity = rotationAxis.forward * speed / conversion; // Application of calculated velocity to Rigidbody

}
```

Figure 5.8: AICar Driving State

If the boolean braking and reset are set to false 5.8, it means that the car is driving normally, in which case the script adjusts the acceleration and jerk (the rate of change of acceleration) and the car's speed is updated according to the set_acceleration variable. To smoothly accelerate the car towards its target speed, the speed is increased by the set_acceleration value multiplied by the time step between physics updates (Time.fixedDeltaTime). When the vehicle accelerates, the localRotation of the model is gradually adjusted so the car's pitch (tilt) changes to visually simulate the change of

speed. More specifically, a higher acceleration, causes the front of the car to tilt upwards and braking cause it to tilt downwards, with boundary values of -0.5 to 0.5.

If the braking boolean is true while reset boolean is false 5.9, the cars starts braking, eventually coming to a full stop. Delta_distance variable is calculated as the absolute difference between the car's position and braking triggerLocation. This variable measures how far the car is from a braking trigger point in order to set the descending speed value until the full stop. Based on Newton's 3rd equation of motion, $v^2 = u^2 + 2as$, *where v is final velocity, u is initial velocity, a is the acceleration and s is the distance covered (delta_distance)*, the speed is calculated in order to smoothly decelerate the car as it approaches the target stop distance. Also, the pitch is gradually changing based on the car's speed and distance to simulate the car's nose dipping when slowing down or leveling out when speed is low or zero. The boolean shouldYield is set to true in this case, since the waypoint that causes the car to break enables it. In this case, when the car stops completely, the Yield coroutine is called and the timer counts the time needed to pass in order to start speeding again. This time interval comes as an input from the waypoint collider which starts the process of braking.

```
// This statement is applied when the car starts braking
else if ((braking == true) && (reset == false))
{
    // Compute delta distance for deceleration
    if (WaitInputX == true)
    {
        delta_distance = Mathf.Abs(this.gameObject.transform.position.x - triggerlocation);
    }

    else if (WaitTrialZ == true)
    {
        delta_distance = Mathf.Abs(this.gameObject.transform.position.z - triggerlocation);
    }

    else if (WaitTrialX == true)
    {
        delta_distance = Mathf.Abs(this.gameObject.transform.position.x - triggerlocation);
    }
    // Apply delta_distance for deceleration
    // Formula: v = sqrt(u^2 + 2*a*s) with v = final velocity; u = initial velocity; a = acceleration; s = distance covered.
    speed = Mathf.Sqrt(900 + 2 * set_acceleration * Mathf.Pow(conversion, 2) * delta_distance); // Application of conversion of km/h to m/s which needs to be squared
```

Figure 5.9: AICar Braking State

If the braking boolean is false and reset value is true 5.10, it means that the car starts accelerating after a full stop. Again, using Newton's 3rd equation of motion, but this time without the initial velocity term (u), the speed is calculated based on the car's acceleration. Specifically, if the car's speed is less than 2f, the script makes sure to gain sufficient speed (2f) when it starts moving again, gradually increasing the speed until it is back to the cruising speed.

```
// This statement is applied when the car stood still and is resetting its speed.
else if ((braking == false) && (reset == true))
{
    // Accelerating in the X direction
    if (WaitInputX == true)
    {
        delta_distance = Mathf.Abs(this.gameObject.transform.position.x - startlocation);
    }

    // Accelerating in the Z direction
    else if (WaitTrialZ == true)
    {
        delta_distance = Mathf.Abs(this.gameObject.transform.position.z - startlocation);
    }

    // Accelerating in the X direction
    else if (WaitTrialX == true)
    {
        delta_distance = Mathf.Abs(this.gameObject.transform.position.x - startlocation);
    }

    speed = Mathf.Sqrt(2 * set_acceleration * Mathf.Pow(conversion, 2) * delta_distance);
    if (speed < 2f)
    {
        speed = 2f;
    }
    theRigidbody.velocity = rotationAxis.forward * speed / conversion; // Application of calculated velocity to Rigidbody

    if (speed >= 10f)
    {
        reset = false;
        WaitInputX = false;
        WaitTrialZ = false;
        WaitTrialX = false;
    }
}
```

Figure 5.10: AICar Reset State

5.4 Eye Gaze

Eye gaze interaction using Tobii SDK involves tracking where a user is looking in a virtual environment and using that information to enable interaction with the UI or objects within that environment.

5.4.1 Setting Up Tobii SDK

To start using eye tracking, integrating Tobii SDK into the development environment is needed. In Unity, this typically involves installing and configuring the SDK. Installing the SDK can be done via Unity's Package Manager or by downloading the SDK from the Tobii Website.

Configuring the SDK involves setting up the necessary Tobii component inside the Unity Scene including the TobiiXR_Initializer component, which manages the eye tracking data and TobiiXR_TrackingSpace, which defines the coordinate space for tracking. Withing TobiiXR_Initializer component, G2OM Layer Mask can be set 5.11, which determines what layers should be searched for focusable candidates.

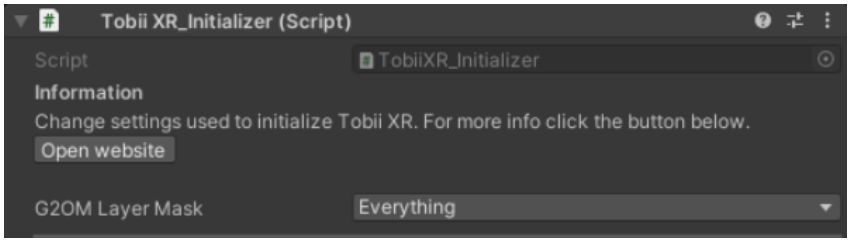


Figure 5.11: Tobii XR Initializer

In this project, objects intended to be focusable are given the EyeGazeFocusable layer input along with the G2OM Layer Mask value. Active objects that belong in this layer will be searched within the scene and become gaze candidates. This decreases computational power needed, since the objects to be searched for eye gaze interaction are much less than the existing objects in the scene.

TobiiXR_TrackingSpace refers to the space being tracked to report eye tracking data in. The tracking space can have two different versions, local and world space. World tracking space corresponds to the cartesian coordinates that consider the center of the environment as point zero. Local tracking space point zero is the center of the parent object that the component belongs to.

5.4.2 Initialization Parameters

The core of eye-gaze interaction lies in capturing and interpreting the eye-tracking data provided by the Tobii SDK. The SDK provides a GazeRay object, which contains information about where the user is looking. The GazeRay has two main properties, its Origin, which is he starting point of the gaze in world coordinates and the direction vector indicating where the user is looking. The Origin point is determined by the coordinates of a GameObject named EyeGazeOrigin, strategically placed exactly where XROrigin camera is.

At the start of project runtime, TobiiXR_Initializer prefab initializes the layer to be checked for gaze focusable objects inside TobiiXR_Initializer script, and then calls Start function 5.12 of TobiiXR script. Start function configures the settings, loads any required licenses, and establishes the connection with the Tobii eye tracker. Afterwards, it creates

the G2OM (Gaze-2-Object-Mapping) object which is a machine-learned selection algorithm, accurately predicting what the user is looking at, during TobiiXR SDK runtime. Additionally, TobiiXR Updater object is created, which is responsible for constantly updating the data input for G2OM.

```
// Setup G2OM
if (settings.G2OM != null)
{
    Internal.G2OM = settings.G2OM;
}
else
{
    Internal.G2OM = G2OM.Create(new G2OM_Description
    {
        LayerMask = settings.LayerMask,
        HowLongToKeepCandidatesInSeconds = settings.HowLongToKeepCandidatesInSeconds
    });
}

// Create GameObject with TobiiXR_Lifecycle to give us Unity events
_updaterGameObject = new GameObject("TobiiXR Updater");
var updater = _updaterGameObject.AddComponent<TobiiXR_Lifecycle>();
updater.OnUpdateAction += Tick;
updater.OnDisableAction += Internal.G2OM.Clear;
updater.OnApplicationQuitAction += Stop;

return true;
```

Figure 5.12: Tobii XR Start method

5.4.3 Eye Tracking Data

TobiiXR.Tick() 5.13 function is called every frame to update and process eye tracking data. This method Calls Internal.Provider.Tick() to update the eye-tracking provider and then fetches new eye-tracking data in local space to transform it to world space. Then, it applies filters to smooth or refine the gaze data and that is how G2OM_DeviceData structure is created. Lastly, it calls Internal.G2OM.Tick(), giving the G2OM_DeviceData previously created as input, to process the gaze data against the scene and determine which objects are being focused on.

G2OM is responsible for refreshing the gaze focusable candidates at a given refresh rate, determined by the value of DefaultCandidateMemoryInSeconds variable which is set at 1 second by default. More specifically, in every cycle of refresh, the function GetRelevantGazeObjects 5.14 is called, which finds relevant objects in the active scene, that the

```
private static void Tick()
{
    Internal.Provider.Tick();
    Internal.Provider.GetEyeTrackingDataLocal(_eyeTrackingDataLocal);
    EyeTrackingDataHelper.CopyAndTransformGazeData(_eyeTrackingDataLocal, _eyeTrackingDataWorld,
        Internal.Provider.LocalToWorldMatrix);

    if (Internal.Filter != null && Internal.Filter.enabled)
    {
        var worldForward = Internal.Provider.LocalToWorldMatrix.MultiplyVector(Vector3.forward);
        Internal.Filter.Filter(_eyeTrackingDataLocal, Vector3.forward);
        Internal.Filter.Filter(_eyeTrackingDataWorld, worldForward);
    }

    var g2omData = CreateG2OMData(_eyeTrackingDataWorld);
    Internal.G2OM.Tick(g2omData);
}
```

Figure 5.13: Tobii XR Tick method

user could potentially focus on based on gaze direction, by calling the FindObjects function. FindObjects determines which of the active scene components can be distinguished as focusable through Unity layer compatibility. Afterwards, it adds new candidates to the list of potential gaze targets and removes old ones that are no longer relevant. This method always ensures to shrink the list to fit G2OM capacity by giving each one of the components a timestamp and an id, and then removing the objects with the oldest timestamp by their corresponding id, if the list exceeds the predetermined capacity.

```
public void Tick(G2OM_DeviceData deviceData)
{
    _deviceData = deviceData;
    var now = _deviceData.timestamp;
    _objectFinder.GetRelevantGazeObjects(ref _deviceData, _newCandidates, _distinguisher);

    AddNewCandidates(now, _internalCandidates, _newCandidates);

    RemoveOldCandidates(now, _internalCandidates, MaxHistoryInSeconds, _keysToRemove, _internalCapacity);

    // Ensure capacity is enough
    if (_internalCandidates.Count > _nativeCandidates.Length)
    {
        _nativeCandidates = new G2OM_Candidate[_internalCandidates.Count];
        _nativeCandidatesResult = new G2OM_CandidateResult[_internalCandidates.Count];
    }

    // Prepare structs to be sent to G2OM
    _colliderDataProvider.GetColliderData(_internalCandidates, _nativeCandidates);

    var raycastResult = _objectFinder.GetRaycastResult(ref _deviceData, _distinguisher);

    _context.Process(ref _deviceData, ref raycastResult, _internalCandidates.Count, _nativeCandidates, _nativeCandidatesResult);

    // Process the result from G2OM
    UpdateListOfFocusedCandidates(_nativeCandidatesResult, _internalCandidates, _gazeFocusedObjects);

    _postTicker.TickComplete(_gazeFocusedObjects);
}
```

Figure 5.14: G2OM Tick method

The G2OM_PostTicker 5.15 script is responsible for handling the logic that occurs

after the G2OM system has determined which objects are being focused on. It manages gaze focus changes and triggers appropriate actions on gaze focusable components. TickComplete() method is called after the G2OM system has completed its processing for the current frame and it updates the focusedObjects withing the FocusedCandidate list, representing the objects currently being focused on. UpdateFocusableComponents() method handles the logic for when the focus shifts from one object to another. If the focused object is different from the previous frame, it calls GazeFocusChanged(false) on the components of the previously focused object to indicate that the object has lost focus. It then clears the list of focusable components and adds the components of the newly focused object.Lastly, it calls GazeFocusChanged(true) on the new components to indicate that they have gained focus.

```
public void TickComplete(List<FocusedCandidate> focusedObjects)
{
    GameObject focusedObject = focusedObjects.Count == 0 ? null : focusedObjects[0].GameObject;

    UpdateFocusableComponents(focusedObject, ref _previousGazeFocusedObject, _gazeFocusableComponents);
}

private void UpdateFocusableComponents(GameObject focusedObject, ref GameObject previousFocusedObject, List<IGazeFocusable> gazeFocusableComponen
{
    if (focusedObject == previousFocusedObject) return;
    if (previousFocusedObject != null)
    {
        foreach (var focusableComponent in gazeFocusableComponents)
        {
            focusableComponent.GazeFocusChanged(false);
        }
        gazeFocusableComponents.Clear();
    }

    if (focusedObject != null)
    {
        focusedObject.GetComponents(gazeFocusableComponents);

        foreach (var focusableComponent in gazeFocusableComponents)
        {
            focusableComponent.GazeFocusChanged(true);
        }
    }
    previousFocusedObject = focusedObject;
}
```

Figure 5.15: G2OM PostTicker method

5.4.4 Object Colliders

The UIGazeCollider script is designed to create and manage a BoxCollider component for UI elements in Unity. This collider is essential for detecting when a user is gazing at a UI element, enabling gaze-based interactions. This script is designed to ensure that UI elements have correctly sized and positioned colliders for gaze interaction. It is particularly useful in scenarios where gaze-based selection or activation of UI elements is needed. The Update() of the script, ensures that the collider is correctly initialized and updated based on the UI element's properties. InitComponents() method initializes the required

components (RectTransform, Graphic, and BoxCollider) if they are not already set. If the BoxCollider does not exist, it generates a new one using GenerateRectCollider(). UpdateCollider() adjusts the collider's size and position based on the UI element's current state. It uses the CalculateCenter() and GetSize() methods to compute the collider's size and position.

More specifically, GenerateRectCollider() creates a new BoxCollider and sets its size and position based on the UI element. CalculateCenter() method, computes the center position of the collider, taking into account the UI element's pivot, offset, and depth. GetSize() computes the size of the collider based on the RectTransform and padding of the component.

The G2OM_ColliderDataProvider script is part of the Gaze-2-Object Mapping (G2OM) system. It provides collider data for game objects that can be focused on by the user's gaze. This data is used to determine which objects are being gazed at in the scene. By supporting various collider types, it ensures that a wide range of objects can be tracked. GetColliderData(), the main method of this script, retrieves collider data for a list of game objects. The method checks for different types of colliders and computes their axis-aligned bounding boxes (AABBs). For each collider type, it calculates the minimum and maximum bounds in local space. The method retrieves the object's localToWorldMatrix and worldToLocalMatrix to transform the bounds from local space to world space. Lastly, it calculates the collider data to be stored in the G2OM_Candidate structure, which is then used by the G2OM system to detect gaze focus.

5.4.5 Visual Feedback

The implementation of visual feedback scripts demonstrate an approach to handling gaze-based interactions in the VR environment. By providing real-time visual feedback, these scripts make the virtual environment more responsive and intuitive. The UIGazeButtonGraphics script offers a complex and customizable interaction experience, while the UIHighlightAtGaze script provides a straightforward but effective way to highlight UI elements. These scripts manage the visual feedback of UI elements in response to the user's gaze, enabling a more immersive and intuitive interaction experience.

The UIGazeButtonGraphics script is responsible for handling the visual feedback of a UI button when it is interacted with through the user's gaze. This script handles how

the button looks when it is in different states: Idle, Focused, and PressedDown, which are the enumerations of the ButtonState. Idle is the default state of the button, when it is not being interacted with, Focused is the state when the button is being gazed at and PressedDown is the state when the button is clicked while being gazed at. The button's appearance changes based on these states to provide the user with visual cues about their interaction with the UI.

The script handles the visual feedback of the buttons through several methods. Awake() method initializes the button's default color, scale, and other properties. This method stores the original appearance of the button so that it can be reverted after the interaction. AnimateButtonPress(ButtonState currentButtonState) triggers the button press animation by starting a coroutine that changes the button's scale and color based on the currentButtonState which comes as an input. AnimateButtonVisualFeedback(ButtonState currentButtonState) handles the visual feedback when the button is focused or loses focus, similarly to the AnimateButtonPress() method.

```
private IEnumerator AnimateButton(float duration, AnimationCurve animationCurve, ButtonState currentButtonState)
{
    var startBackgroundColor = _buttonDefaultColor;
    var startButtonScale = _buttonRect.localScale;
    var endBackgroundColor = _buttonDefaultColor;
    var endButtonScale = _buttonDefaultScale;

    // Updates the end values of the animation depending on the button state.
    switch (currentButtonState)
    {
        case ButtonState.Focused:
            endBackgroundColor = endColor;
            endButtonScale *= _buttonFocusScale;
            break;
        case ButtonState.PressedDown:
            endBackgroundColor = _backgroundPressColor;
            endButtonScale *= _buttonScaleOnPress;
            break;
    }

    // Lerp the colors and scale.
    var progress = 0f;
    while (progress < 0.05f)
    {
        progress += Time.deltaTime * (1f / duration);
        var animationProgress = animationCurve.Evaluate(progress);
        _buttonRect.localScale = Vector3.Lerp(startButtonScale, endButtonScale, animationProgress);

        Color buttonColor = Color.Lerp(startBackgroundColor, endBackgroundColor, animationProgress);
        this.GetComponent<Button>().GetComponent<Image>().color = buttonColor;
        yield return null;
    }
    Color afterFocus = new Color32(255, 255, 255, 200);
    if(currentButtonState!=ButtonState.Focused) this.GetComponent<Button>().GetComponent<Image>().color = afterFocus;

    _buttonAnimationCoroutine = null;
}
```

Figure 5.16: AnimateButton method

AnimateButton(float duration, AnimationCurve animationCurve, ButtonState currentButtonState) 5.16 is the coroutine that handles the actual animation of the button's

appearance. It changes between the start and end values of the button's color and scale based on the given animation curve. The animation runs for a specified duration and smoothly transitions the button between different states. Put more simply, while the button is at Focused state, the coroutine sets its scale and color values to the preferred ones, until it changes to Idle or PressedDown state, in which case these values change back to normal.

During the simulation runtime, if a gameObject gets deactivated, the execution of its methods stop immediately. Thus, in some cases, when there is a UI canvas change, a Focused button might get deactivated before it changes its focused state back to Idle, in which case the color and scale will not change back to normal. For that reason , AnimationReset(GameObject canvas) 5.17 method checks each one of the button objects of the newly opened canvas and changes their color to the original if they are not focused.

```
private void AnimationReset(GameObject canvas)
    {

        var ret = new System.Collections.Generic.List<GameObject>();
        foreach (Button b in canvas.GetComponentsInChildren<Button>())
        {
            if (b.gameObject.layer.Equals(3)) {
                ret.Add(b.gameObject);
            }
        }

        Color ogColor = new Color32(255, 255, 255, 200);

        for (int i=0; i<ret.Count; i++) {
            ret[i].GetComponent<Button>().GetComponent<Image>().color = ogColor;
        }

    }
```

Figure 5.17: AnimationReset method

5.4.6 Selection Techniques

After tracking the eye correctly, and giving feedback where the user is looking at, UITriggerGazeButton script was designed to manage a gaze-aware button's input methods to trigger actions withing the VR environment. Triggering actions include button presses, and have two different selection techniques, as explained in Chapter 2 2.3, dwell-time and

eye blink. An object of this script is set as a component in each of the focusable buttons, checking the state of the button constantly in the Update() method. If a button is focused, ClickButton() method 5.18 is called. Both selection techniques are implemented within ClickButton() method.

```
private void ClickButton(ButtonState currBtnState, Transform btnCords) {

    newButtonCords = btnCords;
    /// <summary>
    /// Implements dwell time selection technique
    /// </summary>
    if (blinkScript.eyeDwellTimeBool && helper.oldBtnCord != null)
    {
        if (newButtonCords == helper.oldBtnCord && currBtnState == ButtonState.Focused)
        {
            helper.timeElapsed += Time.deltaTime;
            if (helper.timeElapsed >= gazeThreshold)
            {
                OnButtonClicked.Invoke(this.gameObject);
                thisButton = this.gameObject.GetComponent<Button>();
                helper.timeElapsed = 0f;
                blinkScript.audio[vents.PlayOneShot(blinkScript.buttonClickAudio);
                thisButton.onClick.Invoke();
            }
        }
        else
        {
            timeElapsed = 0f;
        }
    }


    /// <summary>
    /// Implements eye blink selection technique
    /// </summary>
    if (blinkScript.eyeBlinkBool && currBtnState == ButtonState.Focused && (TobiiXR.GetEyeTrackingData(TobiiXR_TrackingSpace.World).IsLeftEyeBlinking || TobiiXR.GetEyeTrackingData(TobiiXR_TrackingSpace.World).

        thisButton = this.gameObject.GetComponent<Button>();
        StartCoroutine(blinkScript.BlinkStallCoroutine(thisButton,blinkScript.blinker));
    }

    OnButtonClicked.prevFocusedButton = newFocusedButton;
    helper.oldBtnCord = newButtonCords;

}
```

Figure 5.18: ClickButton method

During the initialization of the script 5.19, OnButtonClicked event is initialized to avoid null reference errors.

```
private void Start()
{
    // Store the graphics class.
    blinkScript = GameObject.Find("UI").GetComponent<BlinkCoroutine>();
    _uiGazeButtonGraphics = GetComponent<UIGazeButtonGraphics>();
    helper = GetComponent<GazeTriggerHelper>();

    // Initialize click event.
    if (OnButtonClicked == null)
    {
        OnButtonClicked = new UIButtonEvent();
    }
}

private void Update()
{
    if (_currentButtonState == ButtonState.Focused) ClickButton(_currentButtonState, this.transform);
    // When the button is being focused and the interaction button is pressed down, set the button to the PressedDown state.
    if (_currentButtonState == ButtonState.Focused &&
        ControllerManager.Instance.GetButtonPressDown(ControllerButton.Trigger))
    {
        UpdateState(ButtonState.PressedDown, _uiGazeButtonGraphics._buttonNumber);
    }
    // When the trigger button is released.
    else if (ControllerManager.Instance.GetButtonPressUp(ControllerButton.Trigger))
    {
        // Invoke a button click event if this button has been released from a PressedDown state.
        if (_currentButtonState == ButtonState.PressedDown)
        {
            // Invoke click event.
            if (OnButtonClicked != null)
            {
                OnButtonClicked.Invoke(gameObject);
            }

            ControllerManager.Instance.TriggerHapticPulse(HapticStrength);
        }

        // Set the state depending on if it has focus or not.
        UpdateState(_hasFocus ? ButtonState.Focused : ButtonState.Idle, _uiGazeButtonGraphics._buttonNumber);
    }
}
```

Figure 5.19: UITriggerGazeButton script Initialization

Also, an object of UIGazeButtonGraphics class is initialized in order to be able to interact with a button's color and scale when it is pressed. BlinkCoroutine and GazeTriggerHelper are also initialized, which are used as helper scripts. More specifically, GazeTriggerHelper holds the information of the previous focused button and the time that each button is Focused continuously. Finally, BlinkCoroutine helps with a problem that occurs with eye blink technique which will be described in the next paragraphs.

Dwell-time selection technique relies on focusing on a single button for a specific duration of time which, in this project, is 0.8 seconds. The script constantly checks if the focused button changes, through GazeFocusChanged() method 5.20, updating the previously focused button object in GazeTriggerHelper script every time it changes, and updates the focused duration on the script's timer, until it reaches the threshold of 0.8 seconds. If the focus changes, the timer, which is a different object for each one of the UI buttons, set its value back to 0. If the timer reaches the threshold while the button is still Focused, the timer gets zeroed and the functionality of the pressed button is invoked.

```
public void GazeFocusChanged(bool hasFocus)
{
    // If the component is disabled, do nothing.
    if (!enabled) return;

    _hasFocus = hasFocus;
    if (hasFocus == true) {
        this.helper.timeElapsed = 0f;
    }

    // Return if the trigger button is pressed down, meaning, when the user has locked on any element, this element shouldn't be highlighted when gazed on.
    if (ControllerManager.Instance.GetButtonPress(ControllerButton.Trigger)) return;

    UpdateState(hasFocus ? ButtonState.Focused : ButtonState.Idle, _uiGazeButtonGraphics._buttonNumber);
}
```

Figure 5.20: GazeFocusChanged method

Eye Blink selection technique refers to selecting a button when the users blinks with one eye. Specifically in this project, when a button is Focused, the user can blink an eye to press the button. This is possible, since TobiiXR.GetEyeTrackingData can give the information about a blinking eye in real time. In this project, the user can click a button by looking at it and closing one eye, no matter if it is the left or right. After the eye blink, BlinkStallCoroutine(Button pressedButton, Boolean blinkOrNotBlink) of BlickCoroutine script 5.21 starts. In this script, if the boolean blinkOrNotBlink is true, the button is clicked and its functionality is invoked. If that happens, the boolean sets its value to false for 0.4 seconds and then back to true. That is happening because TobiiXR cannot recognize if the user just closed one eye or if the eye was already closed, which can lead to significant selection errors, if the user close one eye for more time than needed.

BlinkCoroutine makes sure that the user has enough time to open the closed eye after the button press. This is taking place in a different script, because UITrigerGazeButton script is a component in each one of the UI buttons, which means that its methods will stop executing after the button is pressed, if it gets deactivated. Thus, BlinkCoroutine is only set as an object in one UI component that never gets deactivated.

```
public IEnumerator BlinkStallCoroutine(Button pressedButton, Boolean blinkOrNotBlink)
{
    if (blinkOrNotBlink == true)
    {
        blinker = false;
        audioEvents.PlayOneShot(buttonClickAudio);
        pressedButton.onClick.Invoke();
    }

    yield return new WaitForSecondsRealtime(0.4f);
    blinker = true;
}
```

Figure 5.21: BlinkCoroutine Script

5.5 Voice Recognition

Voice recognition features in this project include voice commands, which implement specific functionalities of the UI, and speech to text recognition for writing contact names and message content. It allows for interaction with the Unity environment through spoken commands and dictation, primarily using the Unity Windows Speech API. The main script that implements the functionality of voice recognition is SpeechRecognition, which is a component of the main UI object in the hierarchy. That is mainly happening because voice commands need to be available at any given moment, no matter in what window the user currently is, which is only possible if the script is a component of an object that never gets deactivated.

5.5.1 SpeechRecognition Initialization

During the initialization of the script 5.22, the set of available voice commands need to be declared in a dictionary list of strings that maps spoken keywords to specific actions. Then, the keywordRecognizer is initialized and started, since voice commands

are available when the simulation starts. The dictationRecognizer is initialized but not started, since keywordRecognizer and dictationRecognizer use the same resource, which means that they cannot work at the same time. Finally, the automated stopping of the dictationRecognizer (InitialSilenceTimeoutSeconds and AutoSilenceTimeoutSeconds deactivating dictation if there is silence at the start of the dictationRecognizer or after the user has spoken) is set to 30 seconds. That is a long period of time, but the dictation is deactivated in the methods which use it after its use has ended his task, so the automated deactivation works just as a reassurance in case the user stops interracting with the UI.

```
void Start()
{
    keywordActions.Add("play music", StartMusic);
    keywordActions.Add("stop music", StopMusic);
    keywordActions.Add("volume up", VolumeUp);
    keywordActions.Add("volume down", VolumeDown);
    keywordActions.Add("next track", NextTrack);
    keywordActions.Add("previous track", PreviousTrack);
    keywordActions.Add("sound off", Mute);
    keywordActions.Add("sound on", UnMute);
    keywordActions.Add("end call", EndCall);
    keywordActions.Add("call", StartDict);
    keywordActions.Add("blink on", BlinkOn);
    keywordActions.Add("blink off", BlinkOff);
    keywordActions.Add("dwell on", DwellOn);
    keywordActions.Add("dwell off", DwellOff);
    keywordActions.Add("voice commands on", VoiceCommandsOn);
    keywordActions.Add("voice commands off", VoiceCommandsOff);
    keywordActions.Add("screen on", ScreenOn);
    keywordActions.Add("screen off", ScreenOff);
    keywordRecognizer = new KeywordRecognizer(keywordActions.Keys.ToArray());
    keywordRecognizer.OnPhraseRecognized += OnKeywordsRecognized;
    keywordRecognizer.Start();
    dictationRecognizer = new DictationRecognizer();
    dictationRecognizer.InitialSilenceTimeoutSeconds = 30;
    dictationRecognizer.AutoSilenceTimeoutSeconds = 30;
    dictationRecognizer.DictationResult += OnDictationResult;
    dictationRecognizer.DictationComplete += OnDictationComplete;
    dictationRecognizer.DictationError += OnDictationError;
}
```

Figure 5.22: SpeechRecognition Initialization

5.5.2 Keyword Recognition

OnKeywordsRecognized 5.23 method is called when a keyword is recognized. It checks if the recognized phrase exists in the keywordActions dictionary and invokes the corre-

sponding action if voiceCommandsBool is true. Available voice commands include music and volume handling, interaction techniques handling, ending calls and also one voice command for phone calling. More specifically, if the keyword "call" is recognized, the keywordRecognizer stops, and the dictationRecognizer starts in order to listen to the given contact name. If that spoken name exists in the contacts, an outgoing call to the specific contact starts.

```
private void OnKeywordsRecognized(PhraseRecognizedEventArgs args)
{
    string recognizedPhrase = args.text.ToLower().Trim();
    Debug.Log("Recognized Keyword: " + recognizedPhrase);
    if (keywordActions.ContainsKey(recognizedPhrase) && voiceCommandsBool)
    {
        keywordActions[recognizedPhrase].Invoke();
    }
    else
    {
        Debug.Log("Unknown keyword: " + recognizedPhrase);
    }

}
```

Figure 5.23: OnKeywordsRecognized method

5.5.3 Dictation

Dictation implements speech to text functionality when the user wants to write a new message or give a name to a new contact. StartDict() 5.24 method is called every time the dictationRecognizer is needed. This method checks if the keywordRecognizer is currently running, to stop its runtime, since phrase recognition and dictation recognition cannot work at the same time. After stopping the keywordRecognizer, it shuts down the phraseRecognition system in order to be able to start dictating. Normally, when stopping the keywordRecognizer, the dedicated resources need to be disposed for less memory allocation and less computational power, but in this case, these resources are still needed for the dictationRecognizer. After PhraseRecognitionSystem.Shutdown() the StartDictationRecognizer coroutine starts. In this routine, the first step is to wait until the PhraseRecognitionSystem.Status is Stopped, otherwise the dictation cannot start. When that happens, the dictation starts and the user can talk and write when needed.

```
public void StartDict()
{
    // Stop keyword recognizer to avoid conflicts
    if (keywordRecognizer.IsRunning)
    {
        keywordRecognizer.Stop();
    }
    PhraseRecognitionSystem.Shutdown();
    StartCoroutine(StartDictationRecognizer());

}

public System.Collections.IEnumerator StartDictationRecognizer()
{
    // Wait until the PhraseRecognitionSystem has completely shut down
    yield return new WaitUntil(() => PhraseRecognitionSystem.Status == SpeechSystemStatus.Stopped);
    spRecIsOff = false;
    // Start dictation recognizer
    dictationRecognizer.Start();
}
```

Figure 5.24: Start Dictation methods

OnDictationResult method 5.25 gets the dictated phrase, trims the extra empty spaces and the writes the result on the corresponding InputField.

```
private void OnDictationResult(string text, ConfidenceLevel confidence)
{
    dictatedPhrase = text.ToLower().Trim();
    Debug.Log("Dictation Recognized Phrase: " + dictatedPhrase);

    if (contactTextInput.activeSelf) {
        contactTextInput.GetComponent<TMP_InputField>().ActivateInputField();
        contactTextInput.GetComponent<TMP_InputField>().Select();
        contactTextInput.GetComponent<TMP_InputField>().text = dictatedPhrase.ToString();
        contactTextInput.GetComponent<TMP_InputField>().DeactivateInputField();
    }

    if (newMessageCanvas.activeSelf)
    {
        newMessageTextInput.GetComponent<TMP_InputField>().ActivateInputField();
        newMessageTextInput.GetComponent<TMP_InputField>().Select();
        newMessageTextInput.GetComponent<TMP_InputField>().text = dictatedPhrase.ToString();
        newMessageTextInput.GetComponent<TMP_InputField>().DeactivateInputField();
    }
    if (contactList.Contains(dictatedPhrase))
    {
        MakePhoneCall(dictatedPhrase);
        RestartKeywordRecognizer();
    }
    else
    {
        Debug.Log($"Contact '{dictatedPhrase}' does not exist in contacts.");
    }
}
```

Figure 5.25: OnDictationResult methods

Specifically, this method checks which window is activated in order to write the dictated phrase to the correct gameObject. There are 3 cases, either to call an existing contact by saying the word "call" and the name of the contact as described above, write the name of the new contact entry, which happens if the system detects that the con-

tactTextInput is activated, or write the message content if the newMessage Window is activated.

OnDictationComplete() method 5.26 is triggered when the system considers that the dictation needs to stop, which varies by the DictationCompleteCause which is the input of the method. The possible caused include successful dictation session completion, failure by bad audio quality, cancelled dictation session, timeout caused by the variables explained above, network failure or unavailable microphone. When the session is over, the dictation status is checked, and if it is still running, RestartKeywordRecognizer() method handles the resources used. Otherwise, StartKeywordRecognizer() coroutine is called which immediately starts the voice commands operation.

```
public void OnDictationComplete(DictationCompletionCause cause)
{
    Debug.Log("Dictation completed.");
    if (dictationRecognizer.Status.Equals(SpeechSystemStatus.Running))
    {
        RestartKeywordRecognizer();
    }
    else
    {
        StartCoroutine(StartKeywordRecognizer());
    }
}
```

Figure 5.26: OnDictationComplete method

5.5.4 Recognizer Management

StartKeywordRecognizer() coroutine 5.27 simply waits until the PhraseRecognitionSystem status is running and then starts the keyword recognizer.

```
private System.Collections.IEnumerator StartKeywordRecognizer()
{
    // Wait until the PhraseRecognitionSystem has completely started
    yield return new WaitUntil(() => PhraseRecognitionSystem.Status == SpeechSystemStatus.Running);
    // Start the keyword recognizer
    keywordRecognizer.Start();
}
```

Figure 5.27: StartKeywordRecognizer coroutine

If the keywordRecognizer gets started without the system status running, the voice command functionalities do not work correctly.

RestartKeywordRecognizer() method 5.28 has the main difference that it is mostly called after using dictation, in order to enable the voice commands again. This method starts by checking if the dictation recognizer is running, and then calls OnDisable method.

```
public void RestartKeywordRecognizer()
{
    if (dictationRecognizer.Status.Equals(SpeechSystemStatus.Running))
    {
        OnDisable();
    }
}
```

Figure 5.28: RestartKeywordRecognizer method

OnDisable method 5.29 can be called either for stopping dictation to start keyword recognition or the opposite. That is the reason why this method checks which of the speechRecognition systems is running to stop it. If the keywordRecognizer is running, it stop its process and calls StopSpeechRecognition() coroutine. If the dictationRecognizer is running, it starts by stopping it and the calls StopDictation() coroutine.

```
private void OnDisable()
{
    // Stop and dispose both recognizers when the object is disabled
    if (keywordRecognizer != null && keywordRecognizer.IsRunning)
    {
        keywordRecognizer.Stop();
        StartCoroutine(StopSpeechRecognition());
    }

    if (dictationRecognizer != null && dictationRecognizer.Status.Equals(SpeechSystemStatus.Running))
    {
        dictationRecognizer.Stop();
        StartCoroutine(StopDictation());
    }
}
```

Figure 5.29: OnDisable method

These coroutines 5.30 are used in order to wait until the corresponding speech recognition system has stopped running, otherwise the other system would not start its function correctly.

```
private System.Collections.IEnumerator StopSpeechRecognition()
{
    yield return new WaitUntil(() => PhraseRecognitionSystem.Status == SpeechSystemStatus.Stopped);
}


private System.Collections.IEnumerator StopDictation()
{
    yield return new WaitUntil(() => dictationRecognizer.Status.Equals(SpeechSystemStatus.Stopped));
    PhraseRecognitionSystem.Restart();
    StartCoroutine(StartKeywordRecognizer());
}
```

Figure 5.30: StopPhraseRecognition and StopDictation coroutines

5.6 User Interface

During the implementation of the UI the primary focus was towards the design, in order to facilitate intuitive interactions through eye-gaze. During research and testing, the outcome was that the eye-gaze interactions were easier with buttons closer to the eye origin point. Specifically, buttons on the top side or the right side of the UI were not as easy to select, ending up in frequent selection errors. As described in Chapter 4 4, the UI consists of the main window which includes one weather widget and some vehicle information and 6 buttons at the bottom side of the window which lead to the other 6 windows when selected. In this section, the process of the UI implementation will be described, starting with the thought process of its content.

The content of the UI includes basic functions of the vehicle that can be found in infotainment systems, like the speed and mileage that appear in the main window, tire pressure, maintenance and engine presets that appear in the car window. Also, there is a window to handle the music playback and another window to handle the Air Conditioning unit. Most cars nowadays can also connect with the user's phone in order to use the vehicle's systems for calls, so the phone window can also be considered as a part of a normal car's infotainment system, but also a step towards the mobile office. The message window is an essential part of a mobile office, since the user needs to be able to interact through phone messages or mails. Undoubtedly, even in an automated vehicle, the hands of the user need to be unoccupied, which leads to the use of text to speech algorithms for inserting texts either in messages or during contact adding.

5.6.1 Main Window

The main window includes 6 interactable buttons, belonging in the gazeCandidate layer as described earlier. Selecting one of these buttons will lead the user's view to a new window, depending on which one was selected. The background color chosen was light blue, which is a color that does not strain the eye (e.g. red) and does not cover elements from the external environment. That is also possible, because the opacity of the color (value of a in RGBA color values) was set to 200. The alpha parameter is a number between 0 (fully transparent) and 255 (fully opaque). After experiments and research, 200 was found to be an appropriate value for the UI opacity. Lower values would lead to more transparency than needed, making the interactable elements harder to be distinguished inside the UI, while higher values lead to lack of transparency, making users unable to maintain road awareness.

The informative panel in the main window includes information about the cars functions. More specifically, this panel holds information about the autonomous driver, being active or not. It is set to active by default, beucase in this project there is no need for manual driving. Next, there is a text indicator which constantly updates the speed of the vehicle so the user does not have to watch the speedometer. The speed text is constantly updating its value since it is held in the Update() method of the main script of the UI functionality UIPrinter(). The value is received directly from the speedometer, converted to km/h (multiplied by 1.6) and then converted to an integer 5.31. Last, there is a mileage counter which refers to the number of kilometers the car travel with the current level of fuel. The number written is 405, and is set by default, since the car does not actually have fuel consumption.

```
//Main Window speed
GameObject.Find("UI").GetComponent<UIPrinter>().speedText.GetComponent<TextMeshProUGUI>().text = (Convert.ToInt32(GameObject.Find("UI").GetComponent<UIPrinter>().speed.text)*1.6).ToString();
```

Figure 5.31: Main window vehicle speed

On the right side of the main window, appear a weather widget which informs the user about the weather conditions and temperature of Athens. This panel shows real-time data about the weather in Athens, received from openweathermap.org API, a functionality handled in WeatherWidget script. WeatherWidget is a script which handles the communication between the application and the API server. During its initialization, the

apiUrl and apiKey are important information in order to communicate with the correct server and get access to it. Then, the city name needs to be given as input, and during the script's Start() the coroutine GetWeatherData() is called.

This coroutine starts by creating a string.Format by the name url, inserting the information to be sent to the server, which include the initialized url, apiKey and city name. Then, using UnityWebRequest, it sends an HTTP GET to the server, using the specified url, and waits for its response. When the server responds to the request, if it does not result to connection or protocol error, it calls the ProcessWeatherData method 5.32 which gets as an input the bytes from data interpreted as a read-only UTF8 string. This method processes the JSON response from the weather API, by parsing the JSON string into the WeatherInfo object, which is a custom class that represents the structure of the weather data returned by the API. The temperature is extracted from the WeatherInfo.main.temp and then cast to an integer to be displayed in the UI as a string. The weatherCity text is the name of the city displayed on the UI and the weatherDescription text is also extracted from WeatherInfo object and displayed in the UI.

```
IEnumerator GetWeatherData()
{
    string url = string.Format(apiUrl, city, apiKey);

    using (UnityWebRequest www = UnityWebRequest.Get(url))
    {
        yield return www.SendWebRequest();

        if (www.result == UnityWebRequest.Result.ConnectionError || www.result == UnityWebRequest.Result.ProtocolError)
        {
            Debug.LogError("Error: " + www.error);
        }
        else
        {
            ProcessWeatherData(www.downloadHandler.text);
        }
    }
}

void ProcessWeatherData(string json)
{
    WeatherInfo weatherInfo = JsonUtility.FromJson<WeatherInfo>(json);

    temperatureText.text = ((int)weatherInfo.main.temp) + "°C";
    weatherCity.text = city;
    weatherDescriptionText.text = weatherInfo.weather[0].description;
```

Figure 5.32: Weather Data retrieved from the API

The coroutine GetWeatherIcon 5.33 is responsible for fetching and displaying the weather icon image from the internet using the icon code provided by the API. A string named iconUrl constructs the URL for the weather icon using the iconCode. The icon is fetched from OpenWeatherMap's image repository with the "@2x.png" suffix indicating

a higher resolution version of the icon. Another UnityWebRequest is used again, but this time to download a texture from the constructed iconUrl. When the request is complete, the newly created Sprite is assigned to a UI Image component weatherIcon, which displays the weather icon in the user interface.

```
IEnumerator GetWeatherIcon(string iconCode)
{
    string iconUrl = "http://openweathermap.org/img/wn/" + iconCode + "@2x.png";

    using (UnityWebRequest www = UnityWebRequestTexture.GetTexture(iconUrl))
    {
        yield return www.SendWebRequest();

        if (www.result == UnityWebRequest.Result.ConnectionError || www.result == UnityWebRequest.Result.ProtocolError)
        {
            Debug.LogError("Error: " + www.error);
        }
        else
        {
            Texture2D texture = DownloadHandlerTexture.GetContent(www);
            weatherIcon.sprite = Sprite.Create(texture, new Rect(0, 0, texture.width, texture.height), Vector2.zero);
        }
    }
}
```

Figure 5.33: Weather Icon retrieved from the API

5.6.2 Settings Window

The settings window handles the interaction techniques input. It consists of 3 components for activating and deactivating dwell selection, blink selection and voice commands each of them including a set of On and Off toggle buttons. By default, the blink selection technique and the voice commands are activated, and the dwell time selection is deactivated.

If the user clicks the Off button for eye blink, the EyeBlinkSelectionTechnique() method 5.34 is called, which deactivates the Off button and activates the On button, for later change of the selection technique. Also, a boolean called eyeBlinkBool is set to false, which does not allow for blink selection, since the implementation of ClickButton() method in the UITriggerGazeButton script checks if the eyeBlinkBool is true in order to Invoke the button functionality. The eyeBlinkBool is a boolean component of the BlinkCoroutine script which, as mentioned previously, is a script attached to a gameobject that never gets deactivated.

```
public void EyeBlinkSelectionTechnique() {
    if (blinkScript.eyeBlinkBool)
    {
        blinkScript.eyeBlinkBool = false;
        eyeBlinkOff.GetComponent<Button>().GetComponent<Image>().color = afterFocus;
        eyeBlinkOff.SetActive(false);
        eyeBlinkOn.SetActive(true);
    }
    else
    {
        blinkScript.eyeBlinkBool = true;
        eyeBlinkOff.SetActive(true);
        eyeBlinkOn.GetComponent<Button>().GetComponent<Image>().color = afterFocus;
        eyeBlinkOn.SetActive(false);
    }
}
```

Figure 5.34: EyeBlinkSelectionTechnique() method

If the user click the Off button for voice commands, the method VoiceCommandsEnabler() 5.35 is called, which deactivates the Off button and activates the On button, so the user can later enable voice commands again, if needed. A boolean called voiceCommandsBool, which also is an object of the BlinkCoroutine script, is set to false. Whenever a voice commands is recognized by the SpeechRecognizer, the OnKeywordsRecognized() method in SpeechRecognition script checks if the voiceCommandsBool is true, in order to invoke the use of the spoken voice command.

```
public void VoiceCommandsEnabler() {
    if (spRec.voiceCommandsBool)
    {
        spRec.voiceCommandsBool = false;
        voiceCommandsOff.GetComponent<Button>().GetComponent<Image>().color = afterFocus;
        voiceCommandsOff.SetActive(false);
        voiceCommandsOn.SetActive(true);
    }
    else
    {
        spRec.voiceCommandsBool = true;
        voiceCommandsOff.SetActive(true);
        voiceCommandsOn.GetComponent<Button>().GetComponent<Image>().color = afterFocus;
        voiceCommandsOn.SetActive(false);
    }
}
```

Figure 5.35: VoiceCommandsEnabler() method

If the user clicks the On button for dwell time, the DwellTimeSelectionTechnique() method 5.36 is called, deactivating the On button and activating the Off button. A boolean from BlinkCoroutine script, called eyeDwellTimeBool is set to true, and the

user is able to press buttons by dwelling on them for 0.8 seconds as mentioned earlier. ClickButton() method in the UITriggerGazeButton Script checks if this boolean is true every time the user tries to achieve a selection through the dwell time technique.

```
public void DwellTimeSelectionTechnique() {
    if(blinkScript.eyeDwellTimeBool)
    {
        blinkScript.eyeDwellTimeBool = false;
        dwellTimeOff.SetActive(false);
        dwellTimeOff.GetComponent<Button>().GetComponent<Image>().color = afterFocus;
        dwellTimeOn.SetActive(true);
    }
    else
    {
        blinkScript.eyeDwellTimeBool = true;
        dwellTimeOff.SetActive(true);
        dwellTimeOn.GetComponent<Button>().GetComponent<Image>().color = afterFocus;
        dwellTimeOn.SetActive(false);
    }
}
```

Figure 5.36: DwellTimeSelectionTechnique() method

Also, settings window includes a button with the label minimize UI. If this button is pressed, the method MinimizeUI() is called and the component of the UI is deactivated and another canvas, which appears on the bottom left side of the windshield. This button maximizes the UI and if the user selects it, the method MaximizeUI() is called and the view of the user is directed back to the main window of the UI.

All of these interactions can also be achieved through the corresponding voice commands, as long as voiceCommandsBool value is set to true. If the system recognizes one of these commands, the method of the keyword is invoked, which directs to the methods mentioned above to implement their functionality. The methods mentioned above, are a part of UIPrinter script.

5.6.3 Phone Window

The phone window appears if the user selects the phone button from the main window. Then, the canvas of the main window gets deactivated and the phone canvas is enabled. This canvas consists of two different panels, one for recent calls and one for contacts. By default, the recent calls panel is activated when the user first enter the phone window.

In order to implement the functionality of phone canvas, two new prefabs were made, one for recent calls objects and one for contacts objects. Both of these prefabs include

a name (for contact or recent call), a call button and a message button. The difference between those prefabs is that the contact prefab also includes a delete button so the user can delete a specific contact, while that is not available for recent calls. A script called ContactsType with the attributes of name, call, message and delete button is a component of both of these prefabs in order to save the information of for each one of their instances. By default, the recent calls button has an instance of a recent call prefab with the name Panos and the contacts panel has an instance of a contact prefab with the name George. That serves the purpose of being able to write messages without having to create a new contact first, because as mentioned before, a new message cannot be written if there are no existing contacts.

Either in the recent calls or the contacts panel, the user can press the call button of the instances of the prefabs that appear there. When that happens, the MakePhoneCall() method of the UIPrinter script is called, and the calling procedure starts. A calling dial panel appears in the bottom left side of the phone window, with the name of the contact or recent call appearing there. Also, a calling dial appears in the top side of the screen, within the Toolbar, writing that there is an outgoing call, the name it calls, and also has a red 'end call' button. The call dial panel of the phone window has two different labels, one for outgoing and one for incoming calls. That appears to be the the only difference of this panel, while the name remains to be the name of the caller or the contact to be called, and the red 'end call' button is always there if the user wants to end it.

When a call is taking place, whether it is incoming or outgoing, the recent calls log needs to be updated. Inside the recent calls panel, 4 empty gameobjects were created, with names recent_calls 1-4, which serve as local positions that the recent call prefab instances can take. When the user makes the call, the MakePhoneCall() checks if one of the 4 positions is empty 5.37. A gameobjects called positionsChild was created and attached as a compoment for each of these 4 positions. If the position is not empty, the positionsChild gameobject takes the value of the prefab that appears in that position. So, when the method check the positionsChild of each of the 4 positions, it is aware of which one is empty. Those positions start from the upper side of the panel and end up in the bottom from numbers 1 to 4 respectively. Each time a new recent call object is saved in the log, if there are more recent calls, they slide down by one position. So, each time the method checks for available positions, it starts by searching position1, and if it is empty, it saves the prefab there instantly. If not, it checks for the rest of the positions

and slides each of them by 1 position, and then saves the recent call in position1. Since there are only 4 positions, if number 4 is not empty, it is deleted, by destroying the prefab of this spot, and delete it from positionsChild gameobject.

```
if (callPosition2.GetComponentInChildren<ContactsType>() != null) {
    if (callPosition3.GetComponentInChildren<ContactsType>() != null) {


        if (callPosition4.GetComponentInChildren<ContactsType>() != null) {
            Destroy(callPosition4.GetComponent<UIPrinter>().positionsChild);
        }
        callPosition3.GetComponent<UIPrinter>().positionsChild.transform.SetParent(callPosition4.transform, false);
        callPosition4.GetComponent<UIPrinter>().positionsChild = callPosition3.GetComponent<UIPrinter>().positionsChild;

    }
    callPosition2.GetComponent<UIPrinter>().positionsChild.transform.SetParent(callPosition3.transform, false);
    callPosition3.GetComponent<UIPrinter>().positionsChild = callPosition2.GetComponent<UIPrinter>().positionsChild;
}
```

Figure 5.37: Recent Calls position logic in MakePhoneCall() method

The instantiation gets the recent call prefab as an input, the position1 and the worldPositionStays: false which means that it appears in local position, because it needs to be instantiated in this specific position of recent calls panel. After the method manages the positions, it instantiates an object of the recent calls prefab in position 1, with the correct name from the ContactsType script. Since unity prefabs cannot get values in the inspector, if these values are not also prefabs, the values of the new recent call instance are initialized in the script.

If the user presses the Contacts button located at the left side of the window, the view is directed to the Contacts panel of the phone window. There, the user can add new contacts by pressing the respective button which calls the NewContact() method 5.38 of the UIPrinter script. This method, deactivates the add contact button and activates the save button. Also, it shuts down the phraseRecognizer and activates the dictationRecognizer so the user can write the wanted name of the new contact. After the user speaks the preferred name, the script activates the input field to write it, and also selects it, otherwise it will not accept the input. Then, the user can select the save button, if the input field text is not empty, which calls the SaveContact() method and the opposite procedure starts, deactivating the input field and dictationRecognizer, restarting the phraseRecognizer, deactivating the save button and activating the add contact button. Using the same procedure as in recent calls, the method searches the contactPositions and instantiates the contact prefab in position 1, in local coordinates. A list of strings

called contactList is also updated every time a new contact is added, saving the name of the new contact to the list.

```
public void NewContact() {
    addContactButton.SetActive(false);
    contactName.SetActive(true);
    saveContactButton.SetActive(true);

    if (contactName.GetComponent<TMP_InputField>().text.Length != 0) {
        contactName.GetComponent<TMP_InputField>().text = null;
    }

    if (spRec.keywordRecognizer.IsRunning)
    {
        spRec.keywordRecognizer.Stop();
    }
    PhraseRecognitionSystem.Shutdown();

    contactName.GetComponent<UIPrinter>().ContactNameWriter();
}
```

Figure 5.38: NewContact() method

Deleting a contact calls the DeleteContact() method, which removes the name from the contactList, destroys this contact prefab and sets this positionChild to null. DeleteContact() method searches the below positions of the deleted contact and if they are not empty, it slides their contents one position up.

5.6.4 Message Window

The message window consists of 2 main panels, one for inbox and one for outbox messages. Inbox panel is active by default when the user is opening the message window for the first time. There are 2 time of message prefabs, one for inbox messages and one for outbox messages. Both have a name attribute (receiver or sender), a message content part and a button to open the message. A script called OutboxMessage was created to hold the information for each one of these prefabs instances. This script has the attributes of receiver name, message content, open button, and unreadSign for inbox messages. By default, the inbox contains one instance of a message.

When the user presses the new message button, OpenNewMessageCanvas() method is called. In this method, the inbox or outbox canvas gets deactivated and the new

message canvas is enabled. The method checks the contactList for available contacts and if it is empty a text gameobject "No available contacts" appears, without being able to proceed to message writing. If the list is not empty, it retrieves the names of the contacts and places them in instantiations of a button prefab that appears so the user can choose in which contact he wishes to send the message. After choosing the receiver, the ChooseMessageReceiver() method is called and sendMessage button is activated, and the text to speech procedure starts by calling the WriteMessage() method, with the user being able to talk to write the wanted message. When the sendMessage button is pressed, if the message content is not empty, the message is sent, saving an instance of the outbox message to the outbox panel, and directing the user's view back to it. Saving the instance of the message happens with the same procedure with the gameobject positions as in the contacts and recent calls.

The user is able to open a message to view its content by clicking the open button on its object. When that happens, the openMessageCanvas is activated, showing the information of the OutboxMessage script, within the openMessageCanvas prefab. There, the user has the choice to forward a message by pressing the corresponding button, which calls ForwardMessage() method, which instantiates the prefabs for the receiver choice as explained in the new message procedure. When the user chooses the receiver, the send button appears and the message can be send by calling the SendForwardMessage() which handles the positions that the new forwarded message will appear in the outbox panel.

Also, in the openCanvas, the user has the option to delete the opened message. In this case, the message is deleted from the corresponding canvas with DeleteMessage() method implementing the same procedure as deleting a contact. Inside the message window, the user can choose the inbox or outbox panel from the left side of the window, each of the calling OpenInboxCanvas() and OpenOutboxCanvas() respectively. In these functions, except for activating and deactivating the panels, a sprite which appears around each of these panels is handled. This sprite is made as a button wrapper so the user is constantly aware of the currently activated panel.

5.6.5 Car Window

The car window holds information about basic functions of the car and is divided in 3 main panels. As mentioned before, those panels include Engine Presets, Tire Pressure

and Maintenance. Users can change panels through selecting their respective buttons from the left side of the car window. Each button has a white wrapper gameobject which is activated when the each of the buttons is selected 5.39, and serves as an indicator of the panel currently working. When a button is pressed, one of the methods ShowEnginePresets(), ShowTirePressure() and MaintenanceUI() is called, and the respective wrapper gets activated, and deactivates the previously enabled one. By default, engine presets panel is enabled when the user opens the car window for the first time.

Figure 5.39: Car Window panels

Engine presets panel has 3 available engine presets that can be chosen (Eco, Normal and Sport). By default, eco mode is on. Each of these presets also has a wrapper so

the enabled one can be distinguished. The difference between these wrappers are that they are not white but colored. When one of the preset buttons is selected, ActivatePreset() method is called, which enables the colored wrapper for the chosen preset, and updates the preset text indicator below the buttons. Also, it updates the value of the activeEnginePreset text which is the text value shown in the toolbar.

If the tire pressure button is pressed, ShowTirePressure() method is called, which activates a picture showcasing the pressure of the tires. Maintenance button, calls MaintenanceUI() method which deactivates the TirePressure or EnginePreset panels if needed, and activates Maintenance panel. In this panel, appears a button that detects car malfunctions if pressed. By pressing DetectProblems button, ProblemDetection() coroutine is called, which activates a rotating Cog for 3 seconds, as an indicator of searching car problems. After 3 seconds of rotating, the Cog is deactivated and a message appears, showcasing the malfunctions detected in the car.

5.6.6 A/C Window

Selecting the A/C button from the main window, directs to the Air Conditioning window. There, the user can handle various functions of the air conditioning unit, including the air flow direction, air origin and temperature, handled in the AcPreset and AirFlowPreset scripts. Specifically, the air origin has 2 options, one for air coming from the outside of the car and one for recycling the air existing inside the car. Pressing one of those buttons calls ChangeAirFlow() method from AirFlowPreset script, which highlights the pressed button with a white wrapper. As it comes to air flow direction, there are 5 options: head, legs, head and legs, head and windshield, windshield. By choosing one of those A/C presets, ChangeAcPreset() method of AcPreset script is called, which highlights the chosen preset with a white wrapper and deactivates any other activated wrappers from the rest of the 4 presets. Also, if the A/C was previously closed, ChangeAcPreset() deactivates the On button and ativates the Off button, so that the user can deactivate the air conditioning unit at any time. The set of toggle buttons for opening and closing the A/C unit, call the methods OpenAc and CloseAC respectively. If On button is pressed and the air conditioning unit starts working for the first time, the air flow origin coming from the outside environment is chosen by default, as well as the first air flow direction which stands for air flowing towards head and legs. If a different preset has been chosen either

for origin or direction of air, when the on button is pressed, the previously chosen presets are maintained. When the user selects the Off button, CloseAc() method deactivates the wrappers around the enabled A/C presets. Last, there is a set of buttons that handle the air temperature. The user can handle the temperature through selecting the Plus (+) or Minus (-) buttons to increase or decrease the temperature, invoking TempUp() or TempDown() method of UIPrinter respectively. Those methods increase or decrease the temperature of the air by 2 Celsius degrees, updating the text value shown inside the A/C window. The default value of the temperature is 20 Celsius degrees, the lowest temperature is 12 Celsius and the highest 28 Celsius degrees.

5.6.7 Music Window

The music window can open through pressing the Music button in the main window. It consists of 5 presets of FM radios, a set of On/Off toggle buttons, a set of volume handling buttons and a mute button. Selecting one of the 5 FM radio buttons invokes PlayClip() method 5.40 of UIPrinter script, which enables the audioSource, and feeds it with the audioClip component that the pressed button has as an object (.wav file) in UIPrinter.audioClip variable. Then, the audioSource is set to Play() and the music playback starts. Each of the buttons has a different audioClip attached, and the audioSource is set to stream them on loop, if the user does not stop it.

```
public void PlayClip() {
    if (!audioSource.isPlaying)
    {
        audioSource.clip = this.audioClip;
        audioSource.Play();
        toolBarTrackPlaying.text = trackPlaying.text;
    }
    else if (audioSource.isPlaying && audioSource.clip != this.audioClip) {
        audioSource.clip = this.audioClip;
        audioSource.Play();
        toolBarTrackPlaying.text = trackPlaying.text;
    }
    if (audioSource.isPlaying) {
        onButton.gameObject.SetActive(false);
        offButton.gameObject.SetActive(true);
    }
}
```

Figure 5.40: PlayClip() method

If the audioSource is already playing music and the user chooses a different FM radio button than the one already streaming, PlayClip() changes the audoClip attribute of the audioSource, and starts playing the new .wav track. In case the On button is pressed, StartMusic() method is called, which sets the audioSource on play mode, and activates the Off button, to give users the option to stop the music. When on button is pressed and music starts playing for the first time, the default clip playing is the one attached to FM1 radio button. If it is not the first time, the last chosen FM track starts playing. Either in StartMusic() or PlayClip methods , when the clip to be played is chosen, the name of the FM station playing is saved in toolBarTrackPlaing variable to be used for informing which station is currently playing in the toolBar. Pressing the Plus (+) or Minus (-) buttons increases or decreases the music volume by invoking VolumeUp() and VolumeDown() methods respectively. VolumeUp() increases the volume of the audiosource by 0.1 (since an audiosource volume is a value from 0 to 1), or by 1 in the integer form, and then Math.Truncates the value multiplied by 10, in order to keep its integer value and print it in the music window, as well as in the toolBar through the volumeLevel text variable. VolumeDown() uses the same logic to decrease the volume by 1 unit in integer values. The volume has a high value of 10 which is also the default, and a low value of 0 which mutes the playback sound. There is also a mute button in the left side of the screen, which sets the value of the volume to 0 using the audioSource.mute=true boolean given by unity, or unmutes it if pressed again, setting it back to the previous value. All of the functions toggling music functionalities mentioned above can be accomplished through voice commands. "Play music" voice command set the audiosource on play mode, "stop music" stop the music playback and "volume up" and "volume down" invoke the respective functionality. There are 2 more music related voice commands, "next track" and "previous track". NextTrack() method 5.41 from SpeechRecognition script searching which is the track currently playing, or even chosen without playing, and changes the track to the next radio station. For example, if FM1 is playing, NextTrack() sets FM2 to play, and since there are 5 FM radio stations in total, if FM5 is playing when NextTrack() is recognized, the next track to be played is the .wav of FM1 button. With the same logic, PreviousTrack() method starts playing the track of the previous FM station than the current one.

```
private void NextTrack()
{
    if (audioSource.isPlaying)
    {
        if (audioSource.clip == clip1)
        {
            audioSource.clip = clip2;
            audioSource.Play();
            trackPlaying.text = "FM2";
        }
        else if (audioSource.clip == clip2)
        {
            audioSource.clip = clip3;
            audioSource.Play();
            trackPlaying.text = "FM3";
        }
        else if (audioSource.clip == clip3)
        {
            audioSource.clip = clip4;
            audioSource.Play();
            trackPlaying.text = "FM4";
        }
        else if (audioSource.clip == clip4)
        {
            audioSource.clip = clip5;
            audioSource.Play();
            trackPlaying.text = "FM5";
        }
        else if (audioSource.clip == clip5)
        {
            audioSource.clip = clip1;
            audioSource.Play();
            trackPlaying.text = "FM1";
        }
    }
}
```

Figure 5.41: NextTrack() method

5.6.8 Toolbar Logic

The toolbar of the UI is an informative panel which is always on display, no matter what window the user is currently in. It displays basic information about the UI and car functions that need to be available for the user at any given time. More specifically, it displays information about the active engine preset, a digital clock showing the real time

(UTC+3), and information about the music playback 5.42. Moreover, there are 2 empty panels, one for phone calls, showing the type of the call (incoming or outgoing), the name of the caller, and an end call button, so the user can handle phone calls at all times. The second empty panel is set to show notifications about incoming messages, and warnings from the AD, informing the user for the AV actions. The logic of this toolbar is handled within the UIBarLogic script.

Figure 5.42: Tool Bar Layout

The digital clock is showing the actual UTC+3 time through the UpdateTime() coroutine, which gets the time from Unity's System.DateTime.Now and prints it with the today.ToString(h:mm tt) format, showing the Hour:Minutes of DateTime.now.

In UIBarLogic Update() method, the engine preset currently enabled is showed on the first spot of the toolBar, getting the value through the activeEnginePreset variable which updates its value every time the user changes the preset, as explained in the car subsection. The phone call notification panel in the toolBar gets the name of the name to be displayed through the contactName.text variable in MakePhoneCall() method of UIPrinter, when it comes to outgoing calls. Also, MakePhoneCall() method is only invoked for outgoing calls, printing "outgoing" in the call dial panel of the toolbar.

The incoming notifications are handled by Unity's events. Given specific time stamps handled by coroutines, they delay the event functionality. Specifically, the incoming call and incoming message events are triggered by IncomingCall() and IncomingMessageEvent() coroutines of UIBarLogic script. The delay for those events can be set parametrically as an input during their call in the Start() method of the UIBarLogic script.

IncomingCall() 5.43 coroutine is set to take place 15 seconds after the starts of the simulation, if there is no outgoing call taking place at the moment. Otherwise, it waits until the outgoing call is ended, by checking if the dial panel is deactivated, and then the incoming call takes places. When that happens, the dial panel appears, both in the phone window and the toolbar's phone call information section. The "incoming" label

is enabled, and the name "Anna" is set as the callers name, which can be changed in the IncomingCall() coroutine. Then, the IncomingPhoneCall() method of the UIPrinter script is called with the callerName as an input, in order to set the call prefab in the recent calls panel, as explained in the phone window section.

```
IEnumerator IncomingCall(Boolean callBool) {

    if (callBool && !endCallUpperCanvasButton.activeSelf) {

        yield return new WaitForSeconds(15f);
        callAnswered = false;
        endCallUpperCanvasButton.SetActive(true);
        if (callingLabel.activeSelf) callingLabel.SetActive(false);
        incomingCall.SetActive(true);
        callerName.GetComponent<TextMeshProUGUI>().text = "Anna";
        printer.IncomingPhoneCall(callerName);

    }


}
```

Figure 5.43: IncomingCall() coroutine

IncomingMessageEvent() 5.44 coroutine is set to take place 5 seconds after the simulation starts, if there is no pop-up message coming from the autonomous driver.

```
IEnumerator IncomingMessageEvent(Boolean sendMsg)
{
    if (sendMsg) {

        yield return new WaitForSeconds(5f);
        sendMessage = false;
        messageReceived = true;
        messageNotificationName.GetComponent<TextMeshProUGUI>().text = "markos";
        messageNotificationContext.GetComponent<TextMeshProUGUI>().text = "Hello! How are you?";

    }

}
```

Figure 5.44: IncomingMessageEvent() coroutine

This section of the tool bar is shared between message and autonomous driver notifications, and in the case of an incoming message, it displays the name of the sender and a

part of the message's content that fits in this space. The coroutine then, sets the name of the sender and the context of the message, and then sets the messageReceived boolean to true. In the Update() method of the UIBarLogic script, if the messageReceived boolean is true, the unreadInboxCounter integer is increased by 1, which sets a notification bubble 5.45 on top of the message icon in main window, informing the user for the number of the unread inbox messages.

Afterwards, IncomingMessage() method of UIPrinter script is called, in order to set the inbox message prefab in messages window. This prefab's difference compared to those of the outbox messages, is the sign "Unread" which is written on top of the message object, in red letters. If the user opens this specific message, this sign gets deactivated and the notification bubble decreases its value by one, or even deactivates itself if unreadInboxCounter's value is 0, in the OpenMessageCanvas() method.

Figure 5.45: Unread Message notification

The notifications coming from the autonomous vehicle, always have priority to be displayed in this shared message/AV notifications section of the tool bar. The waypoints are categorized by the behavior of the vehicle when the car's collider hits them. There are

5 categories of possible moves, left, right, slight left, slight right and breaking. Depending on the category of the waypoint that the vehicle hits, PopUpMessages() coroutine 5.46 of UIBarLogic script is called from OnTriggerEnter() method of AICar script, with the corresponding input. PopUpMessage() then activates the correct if statement, depending on the waypoint category, in order to display the corresponding message for the vehicle movement and calls WarningAudio() coroutine so that the user is informed on time.

```
public IEnumerator PopUpMessages(String where) {
    messageNotificationContext.GetComponent<TextMeshProUGUI>().text = " ";
    messageNotificationName.GetComponent<TextMeshProUGUI>().text = " ";
    leftTurn.SetActive(false);
    rightTurn.SetActive(false);
    slightLeftTurn.SetActive(false);
    slightRightTurn.SetActive(false);
    breaking.SetActive(false);
    if (where == "left")
    {
        if(AVWarningSoundBool == true) StartCoroutine(WarningAudio());
        leftTurn.SetActive(true);
    }
    else if (where == "right")
    {
        if (AVWarningSoundBool == true) StartCoroutine(WarningAudio());
        rightTurn.SetActive(true);
    }
    else if (where == "slight left")
    {
        if (AVWarningSoundBool == true) StartCoroutine(WarningAudio());
        slightLeftTurn.SetActive(true);
    }
    else if (where == "slight right")
    {
        if (AVWarningSoundBool == true) StartCoroutine(WarningAudio());
        slightRightTurn.SetActive(true);
    }
    else if (where == "breaking")
    {
        if (AVWarningSoundBool == true) StartCoroutine(WarningAudio());
        breaking.SetActive(true);
    }

    yield return new WaitForSeconds(2f);

    leftTurn.SetActive(false);
    rightTurn.SetActive(false);
    slightLeftTurn.SetActive(false);
    slightRightTurn.SetActive(false);
    breaking.SetActive(false);
}
```

Figure 5.46: PopUpMessages() coroutine

In the last spot of the toolbar, music information is displayed so the user can be aware if it is on or off, which radio station is playing, and the volume of the music. That is really helpful, since these are all components that can be handled by using voice

commands, so having an immediate visual feedback can accurately inform the user if the voice commands input was correct. The Update() method of the UIBarLogic script is always checking the volume of the audioSource so it can update the value displayed in the tool bar. The mute or unmute icon is also changing through this update method by checking if the volume of the audioSource is muted or not. Also, it displays the words "On" or "Off" which shows whether the radio is playing or not, as well as the name of the radio station that is currently playing, if the radio is On.

5.6.9 Audio Events

Audio Events is a script that has 6 different .wav audioClips saved for auditory feedback when specific events take place. Another audioSource named Auido Source - Events was made to be able to play clips of 2 different audiosources at the same time, in case an event takes place while music playback is activated through the radio. Those 6 audioClips include buttonClickAudio, outgoingCallAudio, incomingCallAudio, incomingMessageAudio, messageSent audio and warningFromAV audio. Each one of these .wav audios has been chosen for this specific purpose and is considered to be appropriate for giving the user the correct feedback. Since audioSource-Events is only meant to be used for event triggers, the sounds of the clips is only played one time through audioClip.PlayOneShot(), and not on loop like the general audioSource playback.

The audio for buttonClickAudio is meant to be triggered every time the user clicks a button. Specifically, ClickButton() method of UITriggerGazeButton script uses buttonClickAudio right before invoking the functionality of the button, which means after dealing with all of the checks either for dwell time or eye blink selection technique. outgoingCallAudio is used when the user is make a phone call in recent call or contacts panel of phone window, or using the voice commands for calling an existing contact. In the same way, when an incoming call is taking place through IncomingPhoneCall() method, incomingCallAudio is activated so the user can listen to the incoming phone call ringtone and realise there is a call to be answered. Audio for incomingMessage is activated when there is an incoming message event, and messageSent is the sound playing when the user is pressing the sendMessage button (either in new message or forward message panels). Last, warningFromAV is an audio to let the user know that there is a notification coming from the AV to inform him about the vehicle's actions to be made soon.

Chapter 6

Evaluation & Testing

6.1 Introduction

After finishing the implementation of the application, there needs to be an evaluation to task the effectiveness of the UI. Except for the effectiveness, this testing needs to check the intuitiveness of the interface, and the overall user experience. As mentioned in the evaluation subsection 2.5.4 of Research Overview chapter, testing an interface can be approached with many different methods. In this specific project, the main focus of the evaluation was to test the intuitiveness of the interaction methods, especially the interactions using the eye gaze, as well as the impact of the usability of such an interface in the physical world. More specifically, it includes user experience metrics evaluation to measure task completion speed and error rate, and usability testing evaluation through questionnaires.

6.2 Evaluation Method

The evaluation process starts with a brief training which is accomplished by placing the user in the test Scene which places the car in an empty scene. The purpose of that is to insert the user in the logic of eye gaze and voice command interactions, and get used to the layout of the interface. While browsing the interface and realizing the functionalities of the existing windows, users describe their experience, pointing out ideas about potential functions of the UI and future work, while also mentioning errors that may come up.

The described procedure can also be considered as think aloud evaluation method of the interface. After having enough familiarity with the interface, users are described of the metric evaluation process that will take place when they enter the main scene of the project.

When the main scene of the project starts and the user selects the "Start Experiment" OnGui choice in the first screen of the simulation, experimentOn boolean value is set to true, which activates several gameObjects inside the simulation. More specifically, to obtain the user metrics evaluation results, an experiment that measures task completion speed and task error rate has been implemented. This experiment includes measuring the time needed to accomplish 3 specific tasks within the UI, for each of the eye gaze selection techniques. Each of these tasks contains the selection of a sequence of 5 buttons. The first sequence takes place in the message window, the second in the car window and the third in the music window. Each of these sequences of 5 selections is guided through visual hints that appear in black circles, containing the numbers 1 to 5 to showcase the correct button to select. When the first button of the sequence is pressed, a counter named to represent each of the 3 tasks accordingly, starts calculating the time passed until the 5th button of the sequence is pressed. During that time, the user is told to press the space button of the keyboard for every false interaction that takes place. False interactions include pressing buttons that the user did not meant to, or trying to select a button without being able to do it when intended.

When each of the 3 tasks is completed, the user is guided to the Settings window, where he needs to deactivate the eye blink interaction technique (which is enabled by default), and activate the dwell time interaction technique through their respective toggle switches. Afterwards, the same procedure of completing the same 3 tasks needs to be completed again, this time with a different eye gaze interaction technique. The time needed for each task completion is counted separately, as well as the errors made for each of the 3 tasks in total. Completing these tasks, the user continues cruising the roads of the city while the AD navigates in the predefined route, interacting with the the UI using their preferred eye gaze selection technique.

When the experiment comes to an end, the users are given a set of questionnaires. These questionnaires are separated in 2 different sections. The first section includes questions derived from The System Usability Scale questionnaire which is a standardized version that is used to reliably evaluate interactive products and applications to asses

quality and user experience. This section of the questionnaire got enriched by questions that turned it into a user experience questionnaire with a strong focus on VR applications. The second section of this set, delves on more specific aspects of the UI functionality and simulation specifics. More specifically, it includes questions that get a grasp of the user's view on the interaction techniques used in the project, and their preferences between them. Also, it tries to derive information from each of the user's point of view about the concept of the mobile office, their road awareness during the experiment and the trust they developed towards the AD through the notification messages. These questionnaires aim to gather information regarding the usability and effectiveness of the application, while also comprehending user's view about the usage of such interfaces in autonomously driven vehicles in the physical world.

6.3 Evaluation Results

The results of the experiment have been derived after a number of 12 participants (N=12) went through the process. Those participants had big deviations in experience with VR technology and headsets, varying from those who have never used a headset before to those who use VR headsets every day. This difference between experience with VR can give useful insights about the usability of the application and the intuitiveness of the UI, leading to the comprehension for the need of training. More specifically, the first section of the results shown is about the metric evaluation process, displaying the errors made for each of the eye gaze selection techniques compared with the VR experience of the users.

Analyzing the results shown in chart 6.1, it is clear that the errors for the Eye Blink technique are lower, with most of users making fewer errors. However, results seem to vary in the "Regularly" and "Every day" categories, where some users still make a few errors. Eye Dwell Technique generally shows a higher number of errors compared to Eye Blink, especially in the lower VR experience categories ("Never" and "Once"). As VR experience increases the errors seem to decrease, but there are still notable errors even in the higher experience categories.

Users who have less VR experience show worse results with the Eye Dwell technique, resulting in more errors. This shows that users have a harder time with this technique or a need for increased training. On the other hand, users who have more prior VR

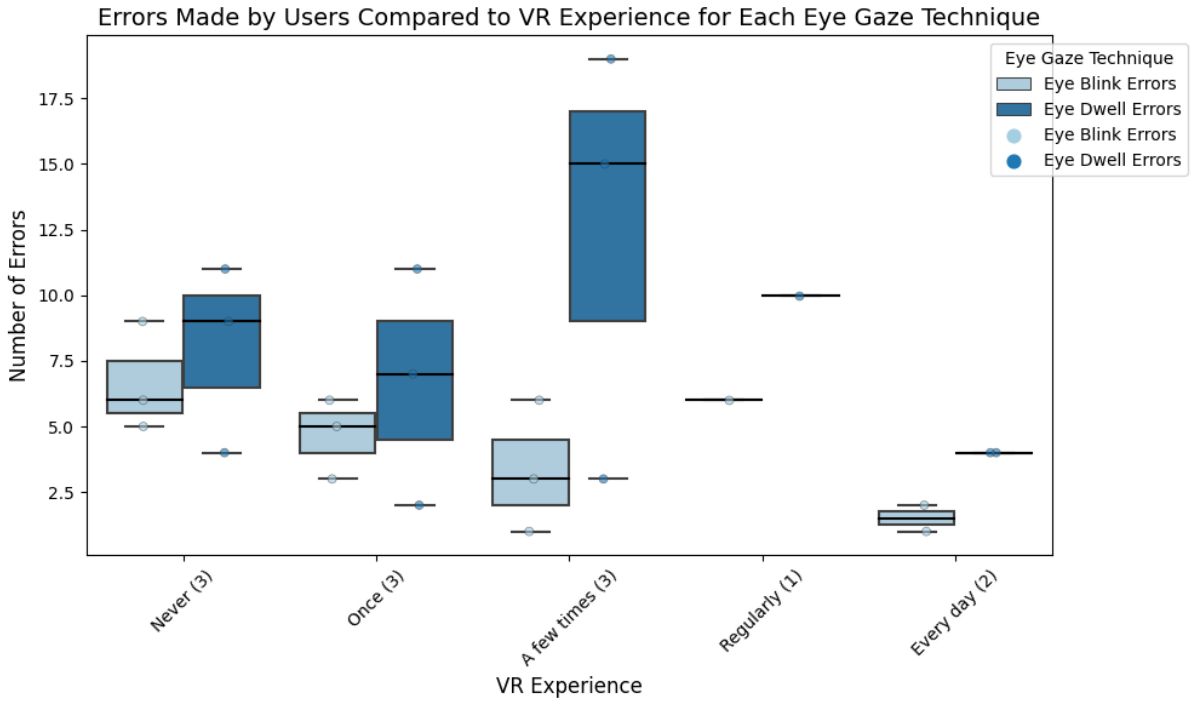


Figure 6.1: Errors made by users compared to their experience with VR, for each Eye Gaze Technique

experience perform better with both techniques, although Eye Dwell still shows a higher error rate than Eye Blink.

The visualization of the boxes show tight concentration for Eye Blink, indicating consistent performance with this technique. For Eye Dwell, the larger spread indicates more variability in performance, which could be due to the technique's known difficulty (popular problems like Midas Touch) or user preference. However, although Eye Blink technique show more consistent results, Eye Dwell technique shows lower absolute value of errors in Never and Once VR experience categories. This could be considered an anomaly since it also shows higher absolute values in the same categories (in fact has the highest absolute value in errors in all categories).

Next we are going to analyze the task completion time needed for each of the 3 tasks comparing both technique with the VR experience of the users. But before that, there needs to be a worthy detail mention for each of the 3 tasks. Those 3 tasks have been specifically designed for the use of this experiment, after many observations through-

out the implementation process of the project. More specifically, while developing this project, there were clear observations about the difference of accuracy and speed of the eye tracking efficiency considering vertical transitions of the eye compared to horizontal ones. Tracking the eye close to the edges of the UI has always been more difficult, which affects vertical transitions much more than horizontal ones because the UI is much smaller vertically than it is horizontally (ratio 1:2). That concludes in much more frequent interactions close to the edge of the UI when it comes to vertical interactions, compared to horizontal ones. Having that in mind, the 3 task have been designed accordingly. First task (message) had smooth transitions in all axes, second task (car) had more intense transitions in the vertical axis and the third task (music) had more intense transitions in the horizontal axis.

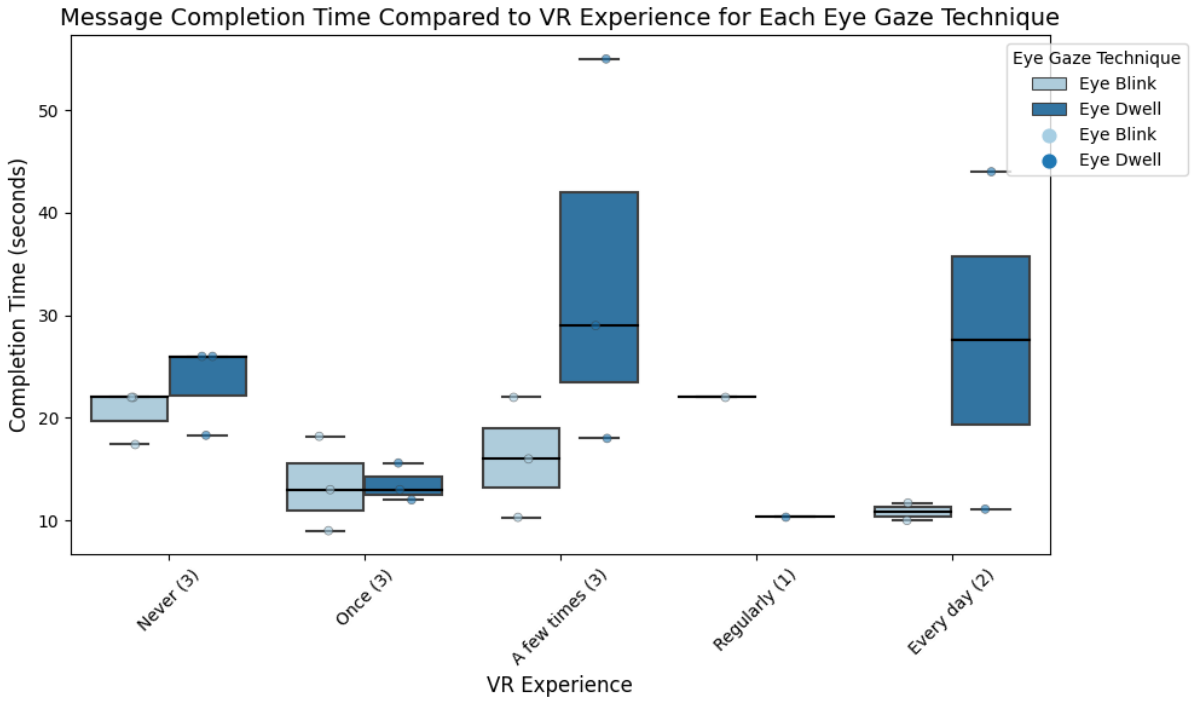


Figure 6.2: Message Task Completion Time compared to VR experience, for each Eye Gaze Technique

The performance in the Message Task 6.2 shows faster completion times and fewer errors in general. This is consistent for all levels of VR experience levels which shows

that the smooth transitions in this task make it easier for users to interact with the UI, regardless of their chosen eye gaze technique. The eye dwell technique seems to be slightly slower, but it also performs relatively well in this task. Less violent transitions seem to reduce the difficulty for users, making the difference between Eye Blink and Eye Dwell techniques less observable in this task.

The smooth transitions in the Message Task seem to minimize the challenges that are affected by vertical or horizontal movements. This explains why both techniques perform similarly, with a slight advantage for Eye Blink due to its general ease of use.

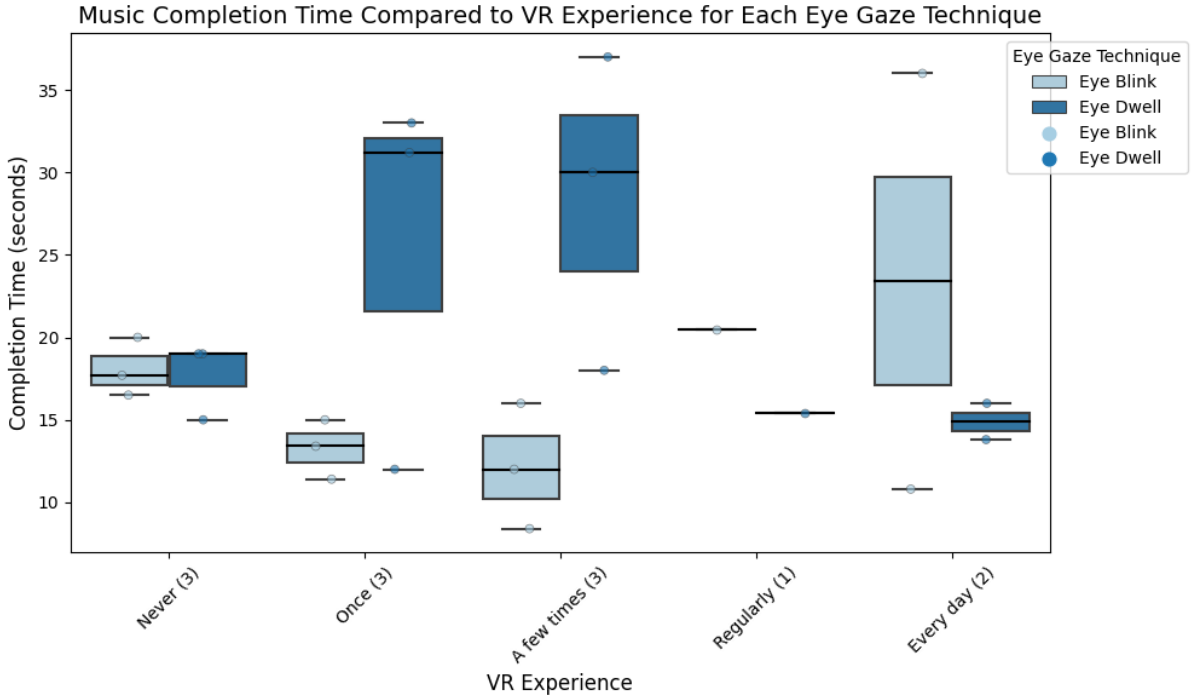


Figure 6.3: Car Task Completion Time compared to VR experience, for each Eye Gaze Technique

Observing the results of Car Task 6.3, the eye blink technique seems to have more variability in performance, especially in terms of completion times. Users with less VR experience might struggle more with the intense vertical transitions, especially because of the UI's proximity to the edges, where eye tracking is less accurate. The Eye Dwell technique shows worse performance in this task. Vertical transitions seem to challenge

users more while using this technique, which results in longer completion times and more errors.

The vertical transitions in the Car Task, especially those near the UI edges, seem to lower performance, especially for the Eye Dwell technique. This is expected according to the observation that eye tracking is less accurate closer to the edges of the UI. Users with less VR experience might find this task challenging, leading to bigger performance differences between the two techniques.

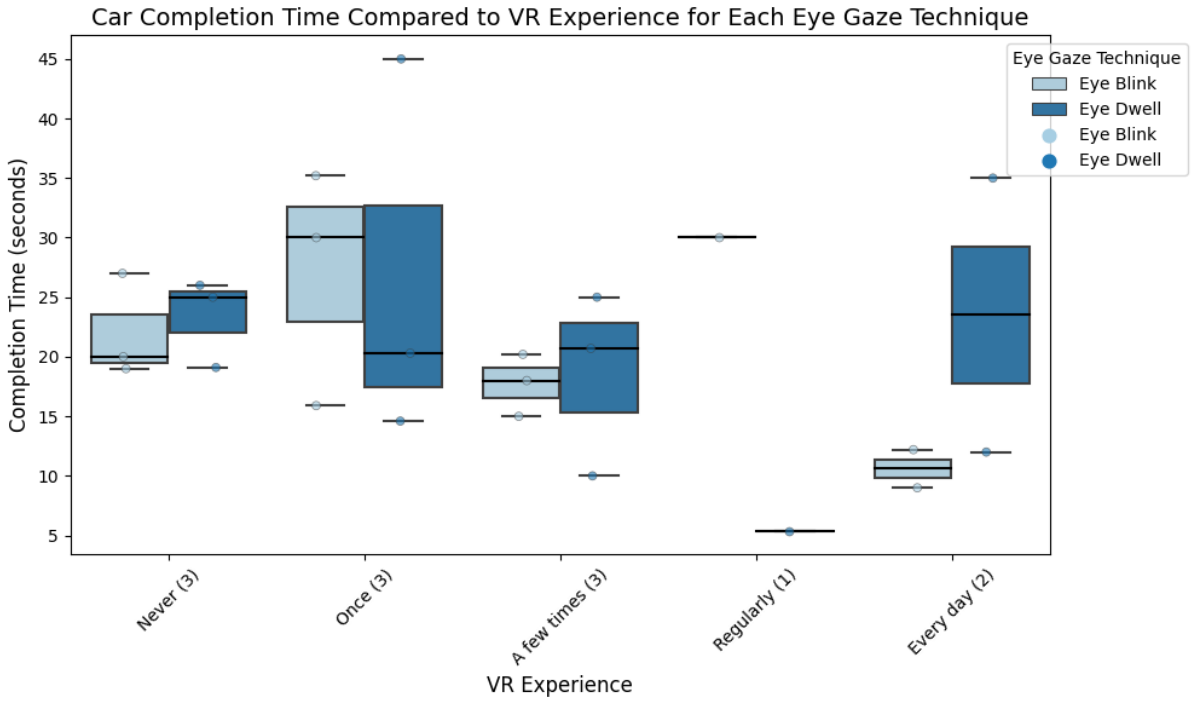


Figure 6.4: Music Task Completion Time compared to VR experience, for each Eye Gaze Technique

The performance of eye blink technique is better in this task 6.4 as well, resulting in fast completion times and low error rates. Horizontal transitions appear to be easier for users, as they involve less interaction near the UI edges. While there are still some challenges, the Eye Dwell technique performs better in this task compared to the Car Task. The horizontal transitions seem to be easier, allowing users to achieve better completion times than in the more vertically intense Car Task.

The design of the Music Task, with its focus on horizontal transitions, seems to fit better with the strengths of both eye gaze techniques. The lower impact of near the edge interactions seems to contribute to the improved performance, especially for the Eye Dwell technique. This shows that horizontal movements are less demanding on users, especially when using less intuitive techniques like Eye Dwell.

The results from the System Usability Scale standardized questionnaire show the user's level satisfaction while using the application. There is a specific approach to how to derive to those results. It includes subtracting 1 from each of the odd numbered questions and subtracting the value of the even numbered questions from 5. Then, these 10 values need to be added and multiplied by 2.5. After this process, the results for our experiment was the number 78.3 out of 100. The average System Usability Scale score is 68 which represents C mark, and the A mark is above 80.3 points. According to that, 78.3 points is a very good score for our project, which means that with a few alternations it could have an A mark, but is already very close.

Afterwards, question considering the User Experience (UX) have been conducted. The results showed will compare the UX evaluation of the users about the application, compared on their level of experience 6.5 and how that might affect their overall experience.

Eye strain is seems to be at low levels across all VR experience categories, mostly around the 1-2 range. There is little to no decrease in eye strain with increased VR experience, suggesting that users did not feel discomfort in this aspect, regardless of their prior experience with VR. The mean values have a slight decrease as the VR experience increases which is normal since experienced VR users probably also have experience in eye gaze interaction, leading to more intuitive moving of the eye which conducts to less eye strain.

Motion sickness is also reported at low levels in most of the VR experience levels. There is a slight increase in motion sickness for those who experienced VR "A few times" but the levels remain low. This result shows that motion sickness was not an important issue for the participants.

Fatigue seems to be more unstable across VR experience levels. Participants who had experienced VR "A few times" showed higher levels of fatigue (around 2-3). This probably suggests that users who are a bit more familiar with VR might still find long

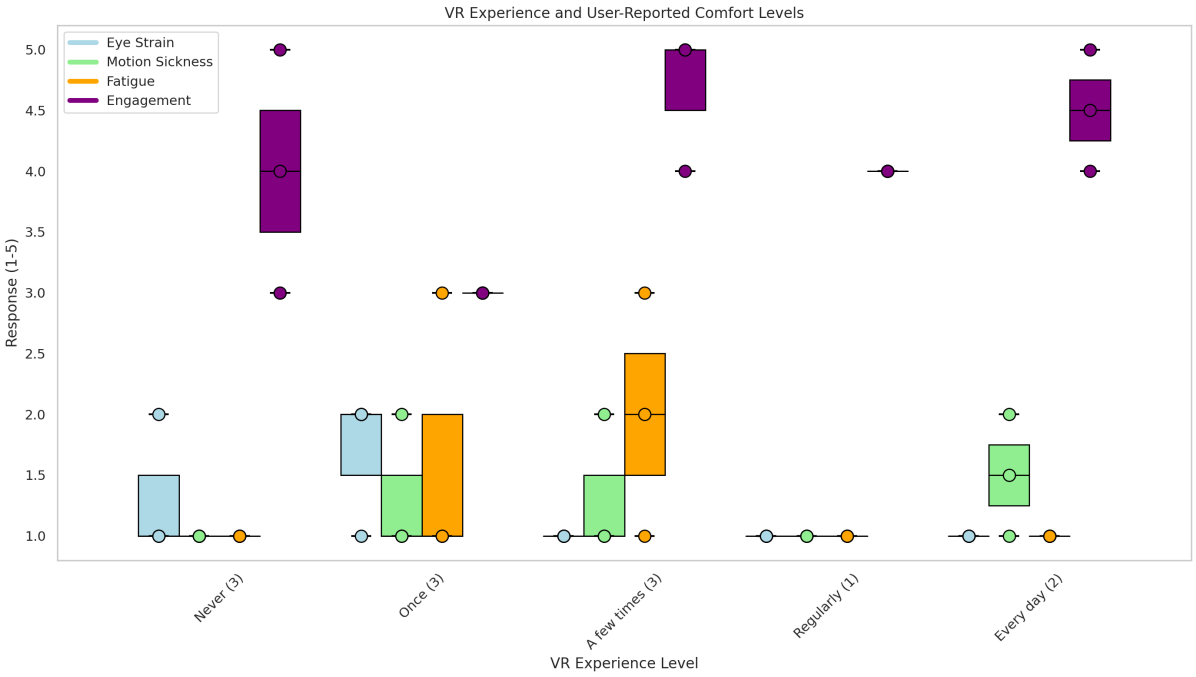
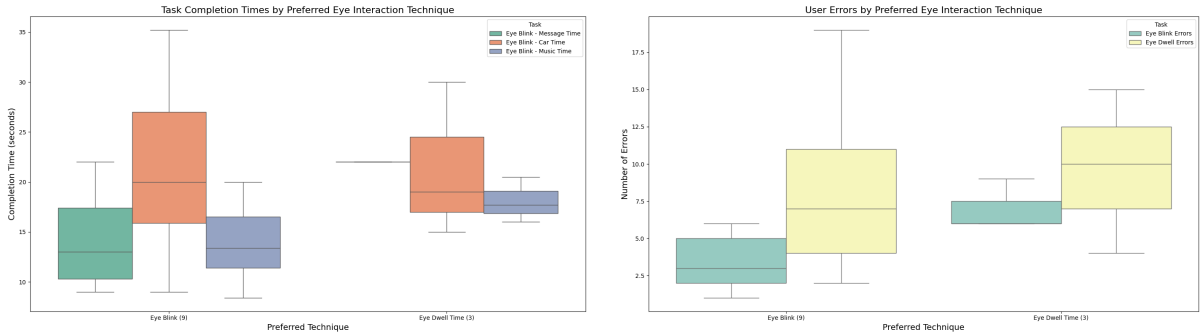


Figure 6.5: User Experience results about the application compared to user's experience with VR

sessions tiring. That result is normal, since they are not as used to using VR for long periods of time as more experienced users.

Engagement scores are showed to be high enough, especially in users with more frequent VR experience. Users who use VR "Every day" report high engagement (around 4-5), showing a high involvement with the VR experience. This suggests that more frequent users are not only more comfortable but also more engaged with the experience. Engagement is lower and more varied among users with less VR experience, indicating that the unfamiliarity of VR might not fully capture their interest or that they might find it less immersive. Also, the struggle to interact with ways that are not used in every day life probably affects users to concentrate more on the interaction part, than this of the general simulation.

In Figure 6.6 we can see the preference of the users between the two eye gaze techniques, based on the metrics derived during the experiment, and analyze if lower task completion times and error rates of one technique necessary lead to being the preferred



(a) User Preference of Eye Gaze Techniques compared to Task Completion Time

(b) User Preference of Eye Gaze Techniques compared to User Errors

Figure 6.6: Preferred Eye Gaze Technique according to User Metrics

one. In the first plot 6.6a, Eye Blink users completed tasks with a bigger variability in completion time, especially for the Car Task, but the mean completion times are considerably lower, and the lowest task times are performed in less than half compared to Eye Dwell lowest times. Eye Dwell users had more consistent but generally slower task times across all tasks. Figure 6.6b shows that users who preferred Eye Blink experienced a wider range of errors, with some of them achieving low error values while some others had a significant number of higher errors. Users who preferred Eye Blink seem to have made consistently more errors.

In general, Eye Blink offers higher variability in both errors and task times, but also offers the lowest completion times and errors by absolute values. This allows for potential better performance when mastered, while Eye Dwell results in more consistent yet slower performance with more errors. Considering those results, it is clear why 9 people chose Eye Blink technique over 3 choosing Eye Dwell. Even users who completed the Car Task much slower using the Eye Blink technique compared to their Eye Dwell task time, chose Eye Blink, showing that the potential of the technique is clearly better once the users get used to it.

After the experiments, while discussing the results and the process with the users, like a think-aloud method added to the usability testing, we got a lot of feedback considering the Eye techniques. Most of the users, no matter their VR experience levels, preferred the Eye Blink technique due to the speed and accuracy it could offer when getting used to it. Nevertheless, some users found Eye Dwell better, either because it suited their personal

preference no matter the effectiveness, or because of having a hard time blinking. More specifically, some people are not used to blink their eyes as easy as others, making this technique seem less intuitive and natural, advancing their eye strain and physical fatigue while using it, especially after a prolonged use of the simulator.

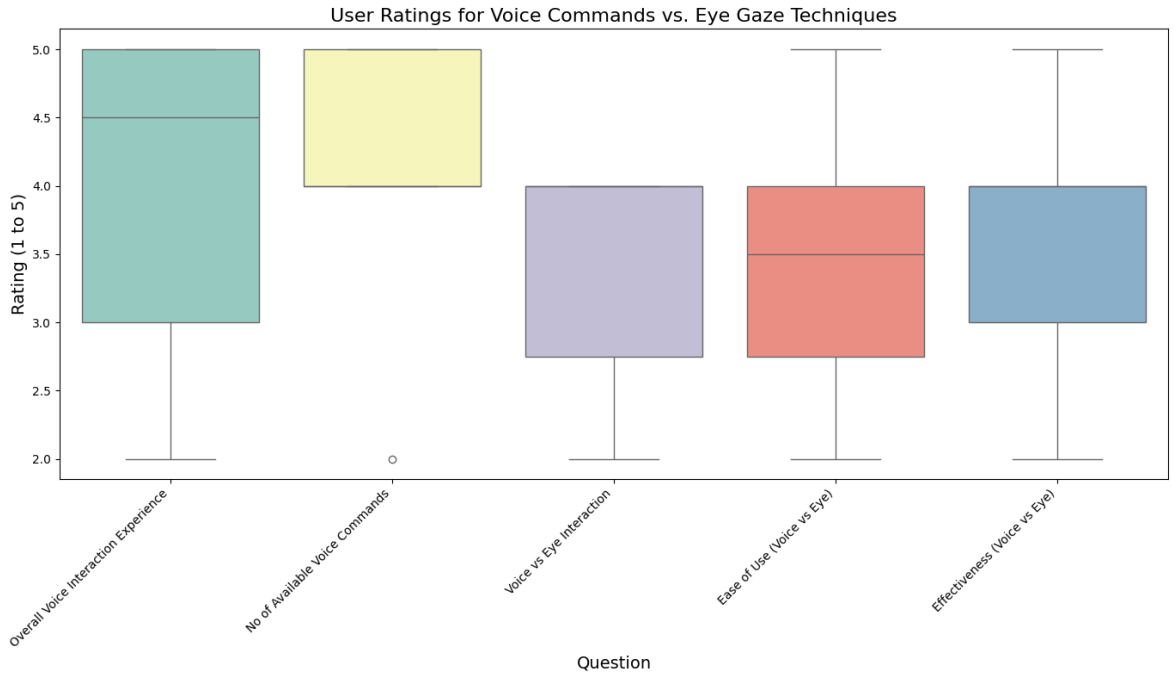


Figure 6.7: Voice Commands interaction results & comparison with Eye Gaze interaction

Figure 6.7 shows the feedback of the users about the interaction with the interface using voice commands. The feedback includes the overall interaction experience as well as comparisons between interactions through voice commands and eye gaze techniques. Starting with the overall experience of the users using voice commands (green box), we can clearly see that most of the users had a good interaction experience using voice commands with mean values around 4 out of 5, while we also see there is a big range in answer, since there are people that answer even the low rate of 2 out of 5. However, the score is more than satisfying, which is to be expected due to the easy and fast task completion voice commands can offer. More specifically, voice commands like previous/next track for example, save valuable time since 2 words can replace the opening of Music window and pressing the preferred radio station, which can be much more times consuming.

However, voice commands are not always accurately working, since the Unity engine voice recognizer is not at a state of the art level like commercial technologies. Accent or talking speed may affect the accuracy of the voice commands. Also, the set of available voice commands cannot implement all of the available interactions within the UI which was expected to affect the overall experience negatively, but apparently users did not find them less than needed, ending up in a mean score of 4.5 out of 5.

Comparing the interaction using voice commands against eye gaze, the users answered about their preference between the two interaction methods as it comes to interacting with the UI in general (purple box), in terms of easy of use (red box) and considering the effectiveness they experienced (blue box). Those questions had a rating from 1 to 5, implying that 3 was the middle, which means that they found voice commands and eye gaze interactions the same, above 3 shows preference in voice commands and below 3 shows preference in eye gaze interactions. Starting with the overall comparison of the 2 techniques, we can see that the interquartile range spans from a bit below 3 to 4, resulting in a mean value of approximately 3.5. This shows that there is a slight preference over voice commands, but values appearing below 3 indicate that some users also liked the eye gaze interaction methods better, with a small percentage of user ratings appearing clearly below 3.

Easy of use and effectiveness show a wide range of variability with users rating from 2 to 5 in both categories. Ease of use shows an interquartile range starting below 3, which implies that there is a decent amount of users preferring eye gaze techniques, however the mean value lies at 3.5 which shows a slight preference over voice commands. Effectiveness also shows quite a variation, with fewer users preferring eye gaze technique this time. Its interquartile range starts a bit above 3, with a mean value of 3.5, showing that users found voice commands a bit more effective than eye gaze techniques.

After receiving feedback from the experiment users, there was a clear liking over voice commands, due to the task completion speed they could offer. However, the accuracy seemed to be a bit less than expected, because of the accent of the users and the input microphone quality, making the overall experience slightly worse. Also, we have been told that many people may feel awkward speaking in English in front of others, making the interaction through voice commands tougher and more stressful.

Figure 6.8 shows the rating of the users about the rate of recovering after interaction errors while using the UI, and their thoughts on how their interaction and experience

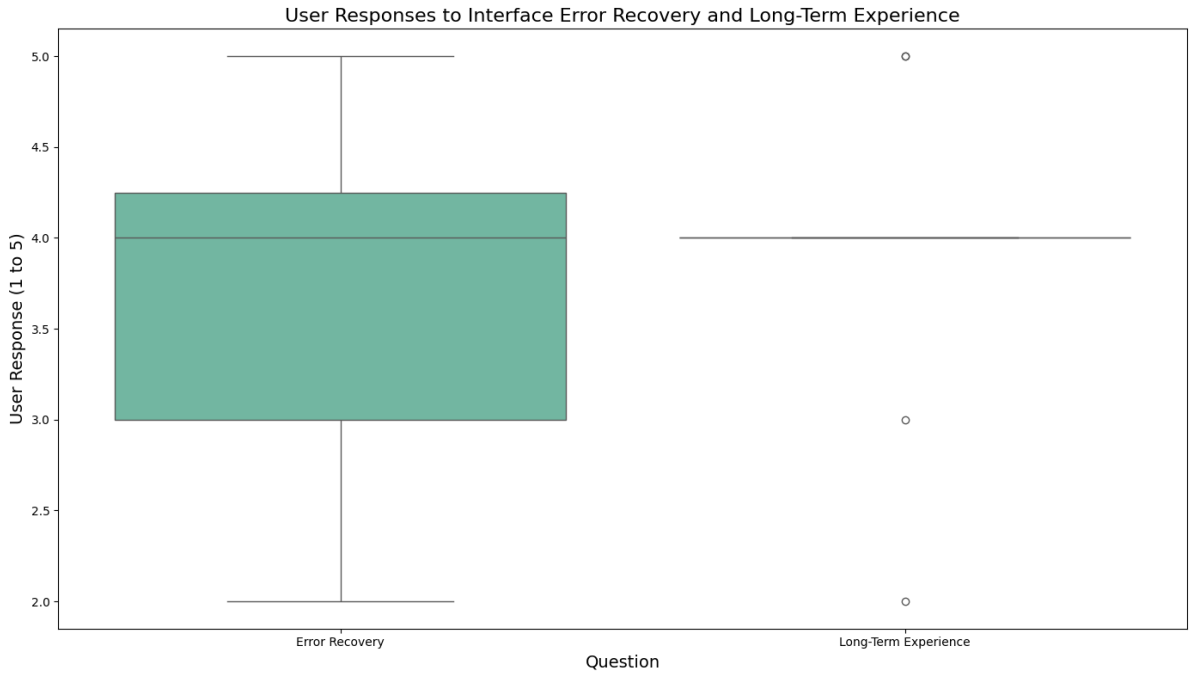


Figure 6.8: Recovery rate after errors & Thoughts on Interaction after long term use

would change after a long period of using it. Error recovery seems to have a big variability in answers, with people rating it from 2 to 5. However the mean value is close to 4 out of 4 which is a very good grade considering that error recovery is very important in UIs. It shows that designing the UI that way has a good an intuitive way of recovering from errors made by users that have never used it before. Rating the change of the UX after a longer period of using the UI seems to have no variability at all, since there is single horizontal line at 4, suggesting that almost all of the users had the same high (4 out of 5) rating. There are only 3 users that rated otherwise, on of the rating higher, at 5, and 2 of them rating lower, with 2 and 3. This shows that there needs to be a bigger training session for users to get used to the UI, but if that happens, they can see the potential of interacting with the UI as it comes to improving effectivness and ease of use.

The next part of our evaluation focused on the trust towards the AV and the mobile office concept. More specifically, users rated their thoughts on the concept of mobile office in general while also rating the trust towards the AV considering their road awareness and how the notifications coming from the vehicle affected them.

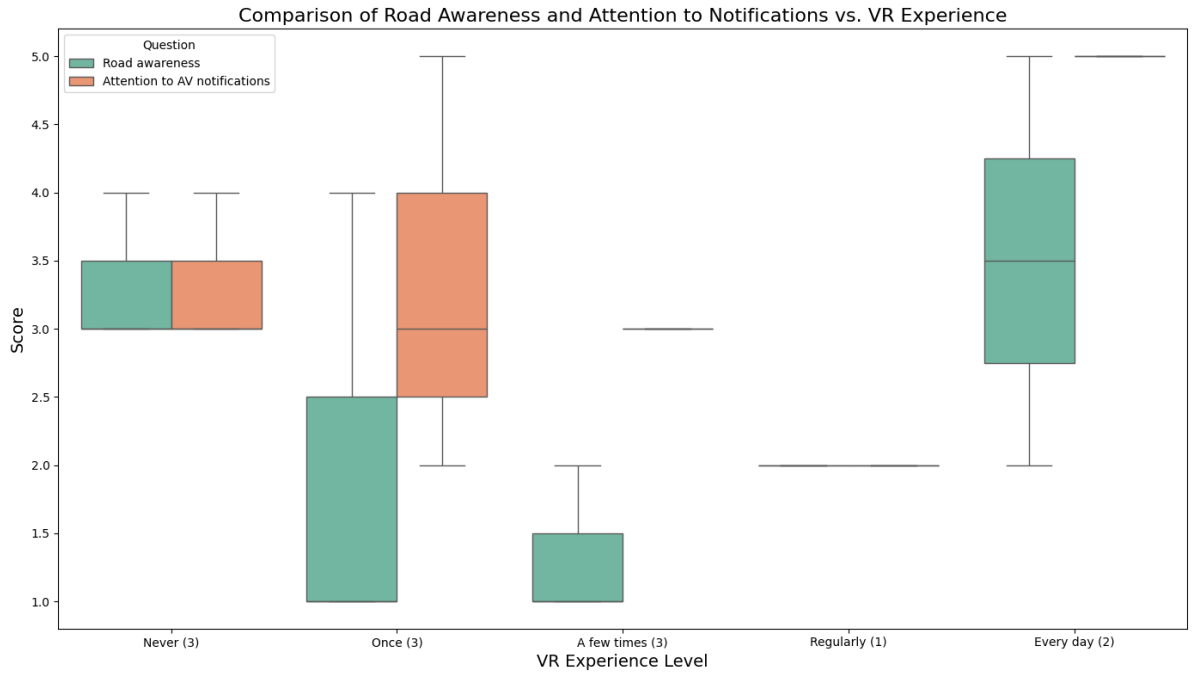


Figure 6.9: Road awareness & Attention to AV notifications of users compared to their Experience with VR

In figure 6.9 we can see the user rating of their road awareness and the attention they paid to the warning notifications coming from the AV, compared to their experience with VR. Road awareness seems to not have been affected by VR experience according to the first 3 levels (Never, Once and A few times). The levels of road awareness are considerably low for all categories expect those who have never experienced VR before, and those who experience it every day, which is not the expected result. However, we can see that there are plenty of users that had more than a decent level of road awareness. As it comes to attention to AV notifications, it seems that the overall level was much higher, while also it was affected by the VR experience of the users. This can be seen, since the results show lower variability in users answers as long as their VR experience advanced, concluding an absolute rating of 5 for participants with the most experience in VR.

Figure 6.10 shows the trust of the users affected by the warnings compared to their attention to the notifications. The chart shows clearly that users who were paying more

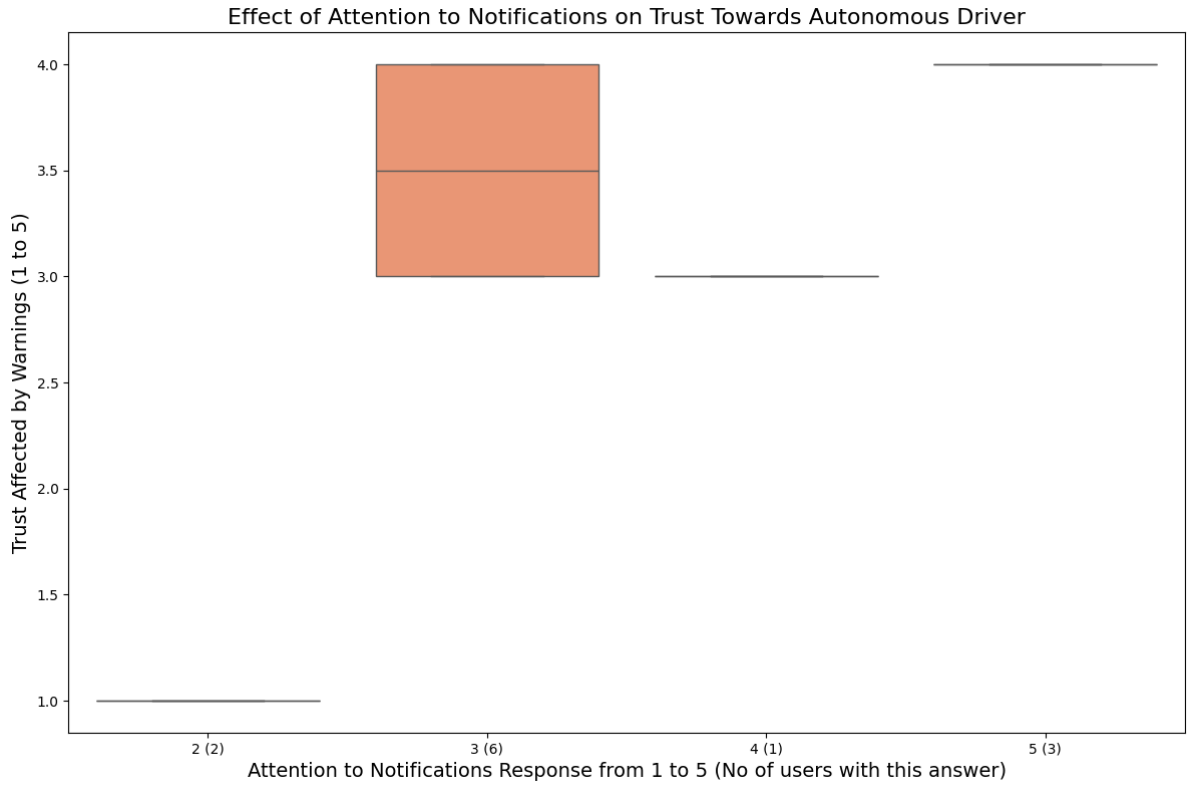


Figure 6.10: How Trust of users towards the AV was affected fromt their attention to AV notifications

attention to the warnings, advanced their trust towards the AV by a lot. This indicates that clear and timed-correctly notifications coming from the AV, informing the user for their the actions, can positively affect the trust of the user towards the decision making of the AV. This is an important fact, since implementing mobile office functionalities, need users to trust the AV, otherwise complex task completion time and overall productivity would be highly decreased.

Figure 6.11 show the user response about mobile office possible functionalities, compared to their thoughts on the mobile office concept. Users responded positively to their thoughts on productivity inside the mobile office concept, showing a little variation for those who have rated the mobile office concept with 3 or 4, nevertheless having an average score above 3.5 and 4.2 respectively. Users seems to trust the vehicle without any supervision with an average score of 4 for the first 3 groups (mobile office thoughts grades

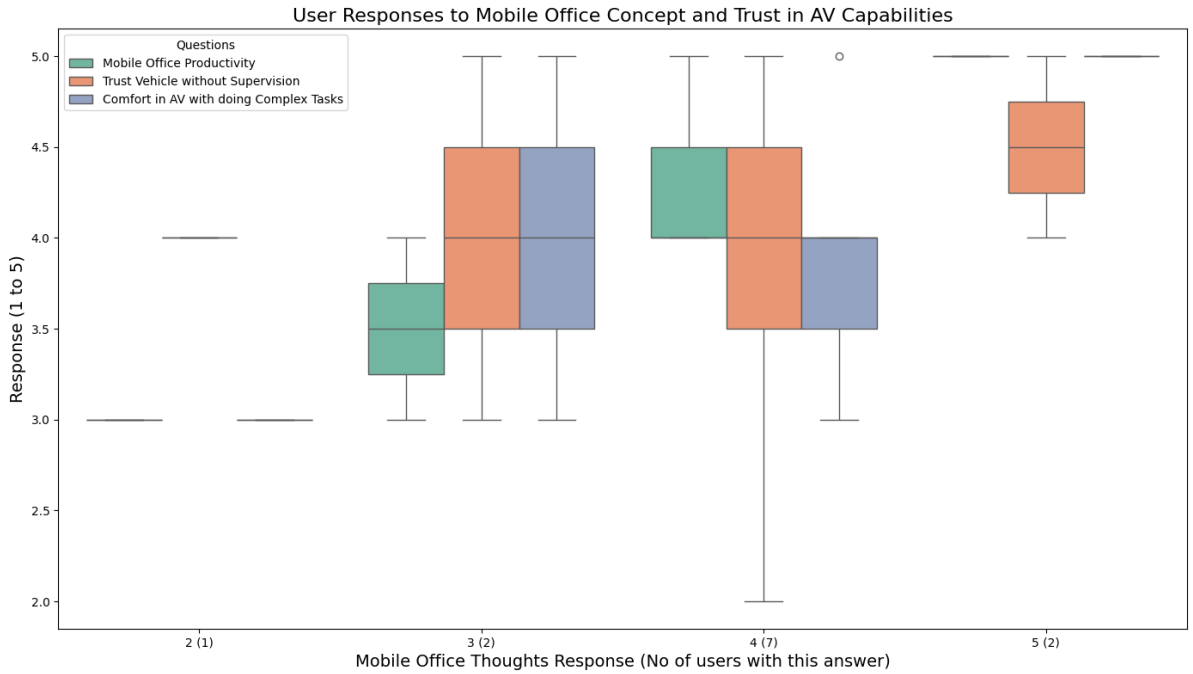


Figure 6.11: Mobile office functions compared to Thoughts of users on mobile office concept

of 2,3 and 4), however, the answers had big variability. The score for this question seems to improve as long as users' answers towards mobile office concept improved, with users grading the concept with 5, having a medium rating of 4.5 as it comes to AVs without supervision. Last, users also responded well to how comfortable they would be in AVs while doing complex tasks. There is a slight increase to this answer, as long as users were more positive to the mobile office concept, with some advanced variability in the middle categories of 3 and 4. However, the overall medium score is above 4, showing that the mobile office concept is highly accepted by users.

Figure 6.12 shows user response towards mobile office functionalities, depending on their trust levels towards the AV. Both answers seems to improving in mean values as long as the trust of the users is improved. Specifically, trusting the vehicle without supervision, seems to improve by a lot which is normal, improving the mean values from 2.5 to 4 and ending up with an impressive 4.5, across the responses of the users for trust towards AV. Also, users responded that they would be more comfortable in doing

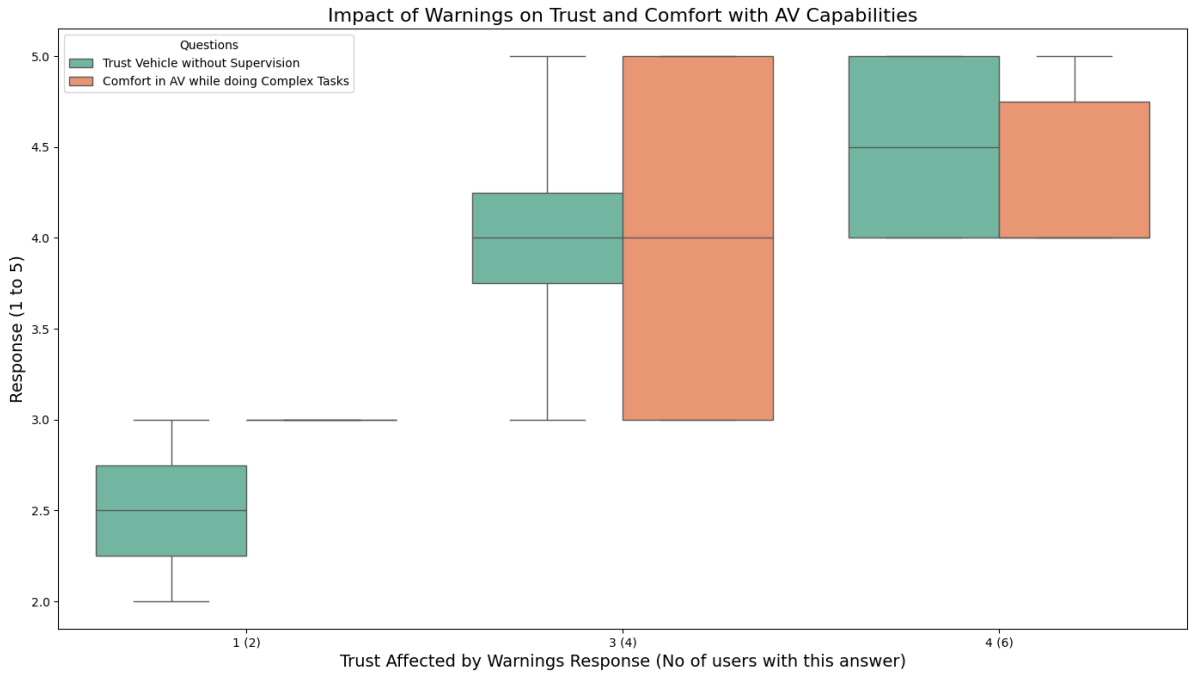


Figure 6.12: Mobile office functions compared to Trust of users towards the AV, affected by the warning notifications Response

complex tasks as long as their respective trust towards the vehicle would improve, with mean values varying from 3 to 4 and ending up with 4.3 across the 3 categories.

The overall analysis of the mobile office concept shows that the users responded with positive feedback. However, according to the evaluation, users need to improve their trust towards the AV which can be achieved through notification coming from the AD. These notifications needs to meet plenty of criteria such as concise phrases signs, both visual and auditory, while also being correctly timed. For that to happen, users need to be more aware of those notifications, either through training, or through finding a more intuitive way of displaying them.

A problem worth mentioning accrued during the evaluation process. People wearing contact lenses were not able to complete the evaluation, since they could not use eye tracking at all, or had aggravated performance. After research, we concluded that this happened because of the HTC Vive Pro eye tracking capabilities, which has a difficult time locating the eye accurately through some specific type of lenses, sometimes leading

to internal errors without being able to locate it at all.

Chapter 7

Conclusion & Future Work

7.1 Conclusion

In this thesis, we created a VR simulator designed to test and evaluate interaction techniques for windshield display User Interfaces in a vehicle that automatically navigates around a city scape. The simulator includes an immersive VR environment depicting city roads with different formats, a fully functional Autonomous Driver that cruises through predefined routes and a WSD UI that can be handled through 2 different eye gaze selection techniques (Eye Blink and Eye Dwell) and voice commands. Through this simulator, we managed to provide a safe and engaging environment for users to explore the concept of mobile office in an autonomously driven vehicle, and handle the interface through hands free interactions. The experiment, designed after the end of the implementation, offers valuable insights into user preferences, performance, and comfort when interacting with the UI comparing the available selection techniques. The evaluation of the experiment also focuses on the impact of the VR experience of users and how that affects their performance, comfort and preference results as well as the trust they showed towards the AD.

Our findings show that, based on the experiment, Eye Blink outperformed the Eye Dwell and resulted in the lowest mean completion times and error percentages at all VR experience levels compared to Eye Dwell. This essentially shows how important intuitive and user-friendly interaction methods are in improving user experience and that known drawbacks for this technique, like Midas Touch, are affecting its performance. However, Eye Dwell still proves to be promising, especially for users that have more VR

experience, thus pointing out that designing the UI based on the interaction techniques that are available is essential to meeting the needs of users with varied technological experience. Also, voice commands were proven to work very well, more especially on the tasks that require higher speed or in tasks that the eye gaze techniques appeared hard to use due to the complexity of the task. This means that a multimodal approach, combining voice commands with eye-tracking interactions, will be most helpful to create an efficient user interface.

We also explored users' trust in the AV and their comfort with the mobile office setup. Results showed that detailed feedback regarding the decisions of the AD significantly improved the trust of users. Further, we saw that trust in the AV significantly influenced user engagement and performance. Users more comfortable with the UI due to a greater amount of trust have produced a better outcome from tasks. This shows how important trust is within the successful operation of autonomous driving technologies. An example could be in the so-called mobile office scenario, where a user can work while the vehicle is autonomously driven. However, in this case, comfort depended on previous VR experience and the familiarity of the users with the autonomous systems. Those users who were more comfortable with the mobile office setup indicated better performance. User adaptation was therefore an important guideline in the design of such spaces, which should have features promoting comfort and reduction of cognitive load.

Overall, our research takes a look on the importance of developing interaction systems for autonomous vehicles that are flexible and effective. By integrating and optimizing different interaction techniques our study provides an understanding of how users might interact with future interfaces in vehicles. According to the results, those interfaces need to consider trust factors and enhance comfort within the mobile office concept. This will pave the way for future innovations in creating autonomous vehicle experiences that concentrate around the user. It shows how important it is to build systems that are adaptable and can of earn user trust.

7.2 Future Work

While the project developed in this thesis seemed to provide a realistic and accurate simulation for developing User Interfaces that can be interacted with eye gaze and voice

commands, there are several areas where further research could be conducted to improve its effectiveness and usability.

One limitation of the simulator is the hardware used and specifically the HTC Vice Pro HMD. Specifically, while this HMD is easy to use and has high quality for an undergraduate study, it is heavy for long time use, and the software used for the eye tracker is specific enough to not be able to transfer the project to other HMDs. Other HMDs like Pimax Vision 8k, offer much bigger image resolution and FoV that affects the immersion of the simulation. Additionally, newer HMDs like Apple Vision Pro have much more accurate eye gaze detection, resolving errors that might have occurred during our experiments. Using edge of technology hardware HMDs could contribute in even better experimental results in both eye gaze selection techniques, or even in new techniques that are easier to implement with more accurate tracking of the eye. Another important addition to interaction techniques that could be added with new hardware is hand interactions. Using hands for touch interactions could be used in a SAE level 5 environment that hands would not be needed even in malicious scenarios for the autonomous driver. Touch interaction is the most common and intuitive interaction for most users, and that is the main reason why it was not implemented in this project. Otherwise, the eye gaze interaction would not be the primary way of selecting interactable objects.

One addition that could really benefit the field is dynamic simulation. Making a simulation different every time it runs, with infinite different scenarios ultimately, would test the trust of the user towards the AV more thoroughly. Specifically, this concept includes having an AD that would almost never collide, except for edge cases, and try out scenarios that includes something appearing in front of the vehicle (pedestrians, animals etc), or inappropriate behavior of other vehicles on the road. In this case, the AD should try to avoid collision while informing the user. Evaluating user's trust in this kind of scenarios can be really constructive towards improving the concept of mobile office.

Another area for future work is the development of more functionalities within the UI. These functionalities can include GPS, applications that can be used in desktop environments, like games, web and social media browsing and even video playback. While these functionalities are considered to be the mostly used by an every day technology user, they require much more attention and focus, thus decreasing road awareness which is still important in today's level ADs and mobile office concept. An important functionality

that could be implemented in the interface is Android and Apple CarPlay. This addition will allow users to have the data of their phone (contacts, messages, mails etc) linked with the interface, contributing to more personalized and pre-customized UIs with less effort and connection problems. Essentially, such an interface could serve as a bigger and more comfortable screen for completing tasks that otherwise would require the use of phone or computer. Inserting the use and familiarization of eye gaze techniques on top of touch and voice interactions that already exist in UIs, could potentially speed up the process of task completion and multitasking. Additionally, most of the operating systems in today's phones and computers already have powerful AI based voice interactions, that could potentially eliminate the problem of predefined voice commands with dynamic voice interactions.

One of the most important additions that can be made in this interface is user collaboration. Specifically, in terms of VR/AR technology, VR collaboration is about coexisting with your colleagues in the same simulation while experiencing virtual contents. In the context of such a UI, collaboration could be a result of two users (driver and front passenger), using the same screen to complete tasks. Those tasks could either be productive, in which case two users could potentially speed up their completion and multiplayer gaming sessions. Another approach to collaborative UIs could be for the passenger to use a small area of the WSD screen, like the bottom right part of it that appears in front of the passenger, or even divide the screen in two equal parts for the driver and passenger to use respectively.

The limitations of such an interface to appear in real world vehicles has many aspects. First of all, the level of vehicle automation only reaches SAE level 4, and even that appears to be in specific areas of the world, which means that such interfaces could not be tried out everywhere. Another important reason is that holographic screens are not in the desirable level for such an interface to exist in the real world. The light conditions affect the visualization of such screen and it cannot appear to be accurate or visually appealing enough. However, the eye tracking capabilities in the physical world seem to be in a good level to be able to implement such interfaces. The solution to these problems, are AR HMDs that can realistically visualize user interfaces in real world conditions accurately. They implement eye, voice and hand tracking with big accuracy and can be programmed to implement holographic UIs in most light conditions. The only drawback is that the

law still does not allow to use such HMDs in driving environments, even in most of the places that SAE level 4 driving automation is already achieved.

Overall, while the multimodal UI in car WSD developed in this thesis represents a significant step forward in the concept of mobile office, there are still several areas where further research could be conducted to improve its overall effectiveness and usability. By addressing the limitations identified in this chapter, the simulator could become an even more powerful tool for advancing the field of mobile office and even transfer it to the physical world if several conditions are met.

Bibliography

[1] H. Ono, L. Lillakas, A. Kapoor, and I. Wong, "Replicating and extending bourdon's (1902) experiment on motion parallax", *Perception*, vol. 42, pp. 45–59, May 2013. DOI: 10.1068/p7269 (cit. on p. 11).

[2] V. I. Ent, "Twentieth century virtual reality education reprise: Stereographs to google cardboard", (cit. on p. 11).

[3] Y. M. Akinola, O. C. Agbonifo, and O. A. Sarumi, "Virtual reality as a tool for learning: The past, present and the prospect", *Journal of Applied Learning and Teaching*, vol. 3, no. 2, pp. 51–58, 2020 (cit. on p. 12).

[4] I. E. Sutherland, "A head-mounted three dimensional display", in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, 1968, pp. 757–764 (cit. on p. 12).

[5] M. Slater and S. Wilbur, "A Framework for Immersive Virtual Environments (FIVE): Speculations on the Role of Presence in Virtual Environments", *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 6, pp. 603–616, Dec. 1997. DOI: 10.1162/pres.1997.6.6.603. eprint: https://direct.mit.edu/pvar/article-pdf/6/6/603/1623151/pres.1997.6.6.603.pdf. [Online]. Available: https://doi.org/10.1162/pres.1997.6.6.603 (cit. on p. 14).

[6] G. B. Petersen, G. Petkakis, and G. Makransky, "A study of how immersion and interactivity drive vr learning", *Computers & Education*, vol. 179, p. 104 429, 2022, ISSN: 0360-1315. DOI: https://doi.org/10.1016/j.compedu.2021.104429. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360131521003067 (cit. on p. 15).

[7] L. Rosenberg, "Virtual fixtures: Perceptual tools for telerobotic manipulation", in *Proceedings of IEEE Virtual Reality Annual International Symposium*, 1993, pp. 76–82. DOI: 10.1109/VRAIS.1993.380795 (cit. on p. 23).

[8] H. Wang, J. Qin, and F. Zhang, "A new interaction method for augmented reality based on artoolkit", in *2015 8th International Congress on Image and Signal Processing (CISP)*, 2015, pp. 578–583. DOI: 10.1109/CISP.2015.7407945 (cit. on p. 24).

[9] R. Hardy and E. Rukzio, "Touch & interact: Touch-based interaction of mobile phones with displays", in *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, 2008, pp. 245–254 (cit. on p. 26).

[10] M. Billinghurst, A. Clark, G. Lee, *et al.*, "A survey of augmented reality", *Foundations and Trends® in Human–Computer Interaction*, vol. 8, no. 2-3, pp. 73–272, 2015 (cit. on p. 26).

[11] M. Serrano, B. M. Ens, and P. P. Irani, "Exploring the use of hand-to-face input for interacting with head-worn displays", in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 3181–3190 (cit. on p. 27).

[12] A. Ferracani, D. Pezzatini, J. Bianchini, G. Biscini, and A. Del Bimbo, "Locomotion by natural gestures for immersive virtual environments", in *Proceedings of the 1st international workshop on multimedia alternate realities*, 2016, pp. 21–24 (cit. on p. 27).

[13] P. Premaratne and P. Premaratne, "Historical development of hand gesture recognition", *human computer interaction using hand gestures*, pp. 5–29, 2014 (cit. on p. 27).

[14] F. Zhou, H. B.-L. Duh, and M. Billinghurst, "Trends in augmented reality tracking, interaction and display: A review of ten years of ismar", in *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, IEEE, 2008, pp. 193–202 (cit. on p. 27).

[15] F. Ma, F. Song, Y. Liu, and J. Niu, "Quantitative analysis on the interaction fatigue of natural gestures", *IEEE Access*, vol. 8, pp. 190 797–190 811, 2020. DOI: 10.1109/ACCESS.2020.3031967 (cit. on p. 27).

[16] T. Poitschke, F. Laquai, S. Stamboliev, and G. Rigoll, "Gaze-based interaction on multiple displays in an automotive environment", in *2011 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, 2011, pp. 543–548 (cit. on p. 28).

[17] M. Parisay, C. Poullis, and M. Kersten-Oertel, "Felix: Fixation-based eye fatigue load index a multi-factor measure for gaze-based interactions", in *2020 13th International Conference on Human System Interaction (HSI)*, 2020, pp. 74–81. DOI: 10.1109/HSI49210.2020.9142677 (cit. on p. 28).

[18] H. Vrzakova and R. Bednarik, "That's not norma (n/l) a detailed analysis of midas touch in gaze-based problem-solving", in *CHI'13 Extended Abstracts on human Factors in Computing Systems*, 2013, pp. 85–90 (cit. on p. 28).

[19] P. Majaranta, U.-K. Ahola, and O. Špakov, "Fast gaze typing with an adjustable dwell time", in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09, Boston, MA, USA: Association for Computing Machinery, 2009, pp. 357–360, ISBN: 9781605582467. DOI: 10.1145/1518701.1518758. [Online]. Available: https://doi.org/10.1145/1518701.1518758 (cit. on p. 28).

[20] B. Velichkovsky, A. Sprenger, and P. Unema, "Towards gaze-mediated interaction: Collecting solutions of the "midas touch problem"", in *Human-Computer Interaction INTERACT'97: IFIP TC13 International Conference on Human-Computer Interaction, 14th–18th July 1997, Sydney, Australia*, Springer, 1997, pp. 509–516 (cit. on p. 28).

[21] L. C. Trutoiu, E. J. Carter, I. Matthews, and J. K. Hodgins, "Modeling and animating eye blinks", *ACM Trans. Appl. Percept.*, vol. 8, no. 3, Aug. 2011, ISSN: 1544-3558. DOI: 10.1145/2010325.2010327. [Online]. Available: https://doi.org/10.1145/2010325.2010327 (cit. on p. 28).

[22] S. Wibirama, S. Murnani, and N. A. Setiawan, "Spontaneous gaze gesture interaction in the presence of noises and various types of eye movements", in *ACM Symposium on Eye Tracking Research and Applications*, ser. ETRA '20 Short Papers, Stuttgart, Germany: Association for Computing Machinery, 2020, ISBN: 9781450371346. DOI: 10.1145/3379156.3391363. [Online]. Available: https://doi.org/10.1145/3379156.3391363 (cit. on p. 28).

[23] M. Benzeghiba *et al.*, "Impact of variabilities on speech recognition", in *Proc. SPECOM*, 2006, pp. 3–16 (cit. on p. 29).

[24] A.-L. Georgescu, A. Pappalardo, H. Cucu, and M. Blott, "Performance vs. hardware requirements in state-of-the-art automatic speech recognition", *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2021, no. 1, p. 28, 2021 (cit. on p. 29).

[25] L. Sidenmark and H. Gellersen, "Eye&head: Synergetic eye and head movement for gaze pointing and selection", in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '19, New Orleans, LA, USA: Association for Computing Machinery, 2019, pp. 1161–1174, ISBN: 9781450368162. DOI: 10.1145/3332165.3347921. [Online]. Available: https://doi.org/10.1145/3332165.3347921 (cit. on p. 29).

[26] A. T. Duchowski *et al.*, "Eye tracking methodology: Theory and practice [electronic resource]", (cit. on p. 29).

[27] V. Rajanna and T. Hammond, "A gaze-assisted multimodal approach to rich and accessible human-computer interaction", *arXiv preprint arXiv:1803.04713*, 2018 (cit. on p. 30).

[28] A. M. Feit *et al.*, "Toward everyday gaze input: Accuracy and precision of eye tracking and implications for design", in *Proceedings of the 2017 Chi conference on human factors in computing systems*, 2017, pp. 1118–1130 (cit. on p. 30).

[29] D. W. Hansen and Q. Ji, "In the eye of the beholder: A survey of models for eyes and gaze", *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 3, pp. 478–500, 2009 (cit. on p. 30).

[30] M. Turk, "Multimodal interaction: A review", *Pattern recognition letters*, vol. 36, pp. 189–195, 2014 (cit. on p. 30).

[31] S. Oviatt, "Ten myths of multimodal interaction", *Commun. ACM*, vol. 42, no. 11, pp. 74–81, Nov. 1999, ISSN: 0001-0782. DOI: 10.1145/319382.319398. [Online]. Available: https://doi.org/10.1145/319382.319398 (cit. on p. 31).

[32] A. Naumann, I. Wechsung, and J. Hurtienne, "Multimodality, inclusive design, and intuitive use", *Is prior experience the same as intuition in the context of inclusive design*, 2009 (cit. on pp. 31, 48).

[33] S. International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles", 2021. DOI: https://doi.org/10.4271/J3016_202104. [Online]. Available: https://www.sae.org/standards/content/j3016_202104/ (cit. on p. 32).

[34] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review", *Sensors*, vol. 21, no. 6, p. 2140, 2021 (cit. on p. 34).

[35] D. Gruyer, V. Magnier, K. Hamdi, L. Claussmann, O. Orfila, and A. Rakotonirainy, "Perception, information processing and modeling: Critical stages for autonomous driving applications", *Annual Reviews in Control*, vol. 44, pp. 323–341, 2017 (cit. on p. 34).

[36] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving", *Journal of field robotics*, vol. 37, no. 3, pp. 362–386, 2020 (cit. on pp. 34, 36).

[37] C. Cadena *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age", *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016 (cit. on p. 35).

[38] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving", *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017 (cit. on p. 35).

[39] S. D. PENDLETON, "Generalized predictive planning for autonomous driving in dynamic environments", 2017 (cit. on p. 35).

[40] L. Sun, C. Peng, W. Zhan, and M. Tomizuka, "A fast integrated planning and control framework for autonomous driving via imitation learning", in *Dynamic Systems and Control Conference*, American Society of Mechanical Engineers, vol. 51913, 2018, V003T37A012 (cit. on p. 35).

[41] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles", *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 740–759, 2022. DOI: 10.1109/TITS.2020.3024655 (cit. on p. 36).

[42] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving", *arXiv preprint arXiv:1610.03295*, 2016 (cit. on p. 36).

[43] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles", *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016. DOI: 10.1109/TIV.2016.2578706 (cit. on p. 36).

[44] M.-j. Lee and Y.-g. Ha, "Autonomous driving control using end-to-end deep learning", in *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2020, pp. 470–473. DOI: 10.1109/BigComp48618.2020.00-23 (cit. on p. 36).

[45] H. Yin and C. Berger, "When to use what data set for your self-driving car algorithm: An overview of publicly available driving datasets", in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2017, pp. 1–8 (cit. on p. 37).

[46] Tesla, "Tesla autopilot and full self-driving capability", 2024. [Online]. Available: https://www.tesla.com/autopilot (cit. on p. 37).

[47] Baidu, "Apollo: Leading the autonomous driving revolution", 2024. [Online]. Available: https://en.apollo.auto/ (cit. on p. 38).

[48] Cruise, "Av compute: Deploying to an edge supercomputer", 2024. [Online]. Available: https://www.getcruise.com/news/blog/2023/av-compute-deploying-to-an-edge-supercomputer/ (cit. on p. 38).

[49] Waymo, "Waymo driver", 2024. [Online]. Available: https://waymo.com/waymo-driver/ (cit. on p. 39).

[50] T. von Sawitzky, P. Wintersberger, A. Riener, and J. L. Gabbard, "Increasing trust in fully automated driving: Route indication on an augmented reality head-up display", in *Proceedings of the 8th ACM International Symposium on Pervasive Displays*, ser. PerDis '19, Palermo, Italy: Association for Computing Machinery, 2019, ISBN: 9781450367516. DOI: 10.1145/3321335.3324947. [Online]. Available: https://doi.org/10.1145/3321335.3324947 (cit. on pp. 39, 47).

[51] C. Gold, D. Damböck, L. Lorenz, and K. Bengler, ""take over!" how long does it take to get the driver back into the loop?", in *Proceedings of the human factors and ergonomics society annual meeting*, Sage Publications Sage CA: Los Angeles, CA, vol. 57, 2013, pp. 1938–1942 (cit. on p. 40).

[52] M. R. Endsley, "From here to autonomy: Lessons learned from human–automation research", *Human factors*, vol. 59, no. 1, pp. 5–27, 2017 (cit. on pp. 40, 47).

[53] W.-T. Fu, J. Gasper, and S.-W. Kim, "Effects of an in-car augmented reality system on improving safety of younger and older drivers", in *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2013, pp. 59–66. DOI: 10.1109/ISMAR.2013.6671764 (cit. on p. 40).

[54] J. Koo, J. Kwac, W. Ju, M. Steinert, L. Leifer, and C. Nass, "Why did my car just do that? explaining semi-autonomous driving actions to improve driver understanding, trust, and performance", *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 9, pp. 269–275, 2015 (cit. on p. 40).

[55] S. Langlois and B. Soualmi, "Augmented reality versus classical hud to take over from automated driving: An aid to smooth reactions and to anticipate maneuvers", in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, 2016, pp. 1571–1578. DOI: 10.1109/ITSC.2016.7795767 (cit. on p. 40).

[56] A. M. Nascimento *et al.*, "The role of virtual reality in autonomous vehicles' safety", in *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, 2019, pp. 50–507. DOI: 10.1109/AIVR46125.2019.00017 (cit. on p. 41).

[57] S. Yao, J. Zhang, Z. Hu, Y. Wang, and X. Zhou, "Autonomous-driving vehicle test technology based on virtual reality", *The Journal of Engineering*, vol. 2018, no. 16, pp. 1768–1771, 2018 (cit. on p. 41).

[58] B. Kim and E. Kang, "Toward large-scale test for certifying autonomous driving software in collaborative virtual environment", *IEEE Access*, vol. 11, pp. 72 641–72 654, 2023. DOI: 10.1109/ACCESS.2023.3295500 (cit. on p. 41).

[59] E. Ejichukwu, L. Tong, G. Hazime, and B. Jia, "Enhancing autonomous vehicle design and testing: A comprehensive review of ar and vr integration", *arXiv preprint arXiv:2404.19021*, 2024 (cit. on p. 42).

[60] B. Ikeda and D. Szafir, "Advancing the design of visual debugging tools for roboticists", in *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2022, pp. 195–204. DOI: 10.1109/HRI53351.2022.9889392 (cit. on p. 42).

[61] C. P. Janssen, A. L. Kun, S. Brewster, L. N. Boyle, D. P. Brumby, and L. L. Chuang, "Exploring the concept of the (future) mobile office", in *Proceedings of the 11th international conference on automotive user interfaces and interactive vehicular applications: Adjunct proceedings*, 2019, pp. 465–467 (cit. on p. 43).

[62] R. Yin, B. Zhang, M. Kang, T. Li, K. Chen, and Y. Kang, "Basic principles of information system ui design", in *Proceedings of the 15th International Conference on Man–Machine–Environment System Engineering*, Springer, 2015, pp. 419–423 (cit. on p. 44).

[63] N. U. Bhaskar, P. P. Naidu, S. Babu, and P. Govindarajulu, "General principles of user interface design and websites", *International Journal of Software Engineering (IJSE)*, vol. 2, no. 3, pp. 45–60, 2011 (cit. on p. 44).

[64] N. Hamidli, "Introduction to ui/ux design: Key concepts and principles", *Academia. URL: https://www. academia. edu/98036432/Introduction_to_UI_UX_Design_Key_Concepts_and_P [accessed 2024-04-27]*, 2023 (cit. on pp. 44, 45, 51).

[65] B. Albert and T. Tullis, *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes, 2013 (cit. on pp. 45, 52).

[66] D. Norman, *The design of everyday things: Revised and expanded edition*. Basic books, 2013 (cit. on p. 45).

[67] D. Vyas, C. M. Chisalita, and G. C. van der Veer, "Affordance in interaction", in *Proceedings of the 13th Eurpoean Conference on Cognitive Ergonomics: Trust and Control in Complex Socio-Technical Systems*, ser. ECCE '06, Zurich, Switzerland: Association for Computing Machinery, 2006, pp. 92–99, ISBN: 9783906509235. DOI: 10.1145/1274892.1274907. [Online]. Available: https://doi.org/10.1145/1274892.1274907 (cit. on p. 45).

[68] J. Jerald, *The VR book: Human-centered design for virtual reality*. Morgan & Claypool, 2015 (cit. on p. 45).

[69] J. J. LaViola Jr, E. Kruijff, R. P. McMahan, D. Bowman, and I. P. Poupyrev, *3D user interfaces: theory and practice*. Addison-Wesley Professional, 2017 (cit. on pp. 46, 53).

[70] W. Karwowski, M. M. Soares, and N. A. Stanton, *Human factors and ergonomics in consumer product design: Uses and Applications*. CRC Press, 2011 (cit. on p. 46).

[71] J.-S. Kim and S.-W. Lee, "Study on how to improve visibility of transparent display for augmented reality under various environment conditions", *Optics Express*, vol. 28, no. 2, pp. 2060–2069, 2020 (cit. on p. 46).

[72] R. Haeuslschmid, B. Pfleging, and F. Alt, "A design space to support the development of windshield applications for the car", in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ser. CHI '16, San Jose, California, USA: Association for Computing Machinery, 2016, pp. 5076–5091, ISBN: 9781450333627. DOI: 10.1145/2858036.2858336. [Online]. Available: https://doi.org/10.1145/2858036.2858336 (cit. on p. 47).

[73] N. Merat, A. H. Jamson, F. C. Lai, M. Daly, and O. M. Carsten, "Transition to manual: Driver behaviour when resuming control from a highly automated vehicle", *Transportation research part F: traffic psychology and behaviour*, vol. 27, pp. 274–282, 2014 (cit. on p. 47).

[74] J. L. Gabbard and J. E. Swan II, "Usability engineering for augmented reality: Employing user-based studies to inform design", *IEEE Transactions on visualization and computer graphics*, vol. 14, no. 3, pp. 513–525, 2008 (cit. on p. 48).

[75] J. Nielsen and R. Molich, "Heuristic evaluation of user interfaces", in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1990, pp. 249–256 (cit. on p. 49).

[76] A. Inan Nur, H. B. Santoso, and P. O. Hadi Putra, "The method and metric of user experience evaluation: A systematic literature review", in *Proceedings of the 2021 10th International Conference on Software and Computer Applications*, 2021, pp. 307–317 (cit. on pp. 49, 53).

[77] P. G. Polson, C. Lewis, J. Rieman, and C. Wharton, "Cognitive walkthroughs: A method for theory-based evaluation of user interfaces", *International Journal of man-machine studies*, vol. 36, no. 5, pp. 741–773, 1992 (cit. on p. 49).

[78] M. Hassenzahl, "User experience and experience design", *The encyclopedia of human-computer interaction*, vol. 2, pp. 1–14, 2013 (cit. on pp. 50, 51).

[79] M. Jaspers, "The think aloud method and user interface design", in *Encyclopedia of Human Computer Interaction*, IGI Global, 2006, pp. 597–602 (cit. on p. 52).

[80] J. Vanderdonckt, M. Zen, and R.-D. Vatavu, "Ab4web: An on-line a/b tester for comparing user interface design alternatives", *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. EICS, pp. 1–28, 2019 (cit. on p. 52).

[81] A. Poole and L. J. Ball, "Eye tracking in hci and usability research", in *Encyclopedia of human computer interaction*, IGI global, 2006, pp. 211–219 (cit. on p. 52).

[82] J. J. LaViola Jr, "A discussion of cybersickness in virtual environments", *ACM Sigchi Bulletin*, vol. 32, no. 1, pp. 47–56, 2000 (cit. on p. 53).