Development of a Driver Model to Optimize Takeover Requests Using Imitation Learning

Philip Toolan

MASTERARBEIT

eingereicht am

Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2023

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere. This printed copy is identical to the submitted electronic version.

Hagenberg, September 5, 2023

Philip Toolan

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to my supervisor Philipp Wintersberger DI Dr. BSc for his unwavering guidance, invaluable insights, and continuous support throughout the journey of crafting this thesis.

[Spanish] A mi novia Alicia, tu paciencia, comprensión e inquebrantable fe en mí han sido una fuente constante de motivación. Tu presencia a mi lado ha hecho que los retos sean más manejables y los éxitos más alegres.

[Irish] Táim fíor-bhuíoch freisin do mo theaghlach as a ngrá agus a spreagadh gan teorainn. Bhí bhur n-íobairtí agus bhur spreagadh mar bhunchloch mo chuid gníomhaíochtaí acadúla, agus táim buíoch go deo as bhur dtacaíocht gan staonadh.

Abstract

This thesis explores the creation of a potentially human-like AI driving model for situations after a takeover request has been issued. The models can be used to simulate a variety of takeover request scenarios and ultimately be used to optimize the handover process between humans and AI. The models use various machine learning techniques, their performance is evaluated, and the models are compared to human drivers.

The findings indicated that the AI models developed as part of this project did not exhibit statistically significant differences compared to human drivers, except for one human driver who was identified as a statistical outlier.

The thesis presents a novel approach to testing AI models, building upon previous work, and emphasizes the importance of statistically relevant and reliable testing and evaluation methodologies. The AI models developed during this thesis exhibited human-like performance to a takeover request (TOR) across various road types, suggesting their potential for testing additional TOR situations and determining optimal TOR scenarios.

The findings also highlight the need for multiple models to account for significant variance in driving abilities among different human drivers. To accurately replicate this variance, additional AI models and human demonstrations would be necessary to cover a wider range of drivers.

Ultimately, the project successfully achieved its objectives by developing various AI driving models and subjecting them to comprehensive evaluation against their human counterparts, revealing a lack of statistically significant divergence.

In conclusion, the work presented in this thesis substantiates the potential of the AI models developed within it to act human-like and enhance TOR optimization.

Kurzfassung

In dieser Masterarbeit wird die Entwicklung eines potenziell menschenähnlichen KI-Fahrmodells für Situationen nach einer Übergabeaufforderung untersucht. Die Modelle können verwendet werden, um eine Vielzahl von Szenarien für Übernahmeanfragen zu simulieren und letztlich den Übergabeprozess zwischen Mensch und KI zu optimieren. Die Modelle verwenden verschiedene Techniken des maschinellen Lernens, ihre Leistung wird bewertet, und die Modelle werden mit menschlichen Fahrern verglichen.

Die Ergebnisse haben gezeigt, dass die in diesem Projekt entwickelten KI-Modelle im Vergleich zu menschlichen Fahrern keine signifikanten Unterschiede aufwiesen. Die einzige Ausnahme ist ein menschlicher Fahrer, der als ststistischer Ausreißer identifiziert wurde.

In dieser Masterarbeit wird ein neuartiger Ansatz zum Testen von KI-Modellen vorgestellt, der auf früheren Arbeiten beruht und die Bedeutung statistisch relevanter und zuverlässiger Test- und Bewertungsmethoden hervorhebt. Die in dieser Arbeit entwickelten KI-Modelle zeigten bei einer Übernahmeaufforderung (TOR) auf verschiedenen Straßentypen ein menschenähnliches Verhalten, was auf ihr Potenzial für das Testen weiterer TOR-Situationen und die Ermittlung optimaler TOR-Szenarien hindeutet.

Die Ergebnisse machen auch deutlich, dass mehrere Modelle erforderlich sind, um die signifikante Varianz der Fahrfähigkeiten verschiedener menschlicher Fahrer zu berücksichtigen. Um diese Varianz genau nachzubilden, wären zusätzliche KI-Modelle und menschliche Demonstrationen erforderlich, um eine größere Bandbreite von Fahrern abzudecken.

Letztendlich wurden die Ziele des Projekts erreicht, indem verschiedene KI-Fahrmodelle entwickelt und einer umfassenden Bewertung im Vergleich zu menschlichen Fahrern unterzogen wurden, wobei keine statistisch signifikanten Abweichungen festgestellt werden konnten.

Zusammenfassend kann festgestellt werden, dass die in dieser Arbeit vorgestellten KI-Modelle das Potenzial haben, menschenähnlich zu handeln und die TOR-Optimierung zu verbessern.

Chapter 1

Introduction

In this chapter all the background information for the project will be discussed along with the project goals, scope, and road-map of the paper.

1.1 Project Background

This is an original study into autonomous vehicles and specifically how AI models can be used as a method of testing performance in a situation were human intervention is required. Autonomous vehicles have the potential to significantly improve transportation safety, efficiency, and accessibility. However, to fully realize these benefits, it is crucial to develop robust and realistic driver models that can accurately mimic human driving behavior [32]. One of the key challenges in autonomous driving is the handover process between the autonomous vehicle and the human driver, known as driver takeover requests (TOR) [32]. A takeover request refers to the moment when the autonomous vehicle's control system determines that it is unable to safely navigate its surroundings and requests the human driver to take control of the vehicle. This is a critical safety feature that is essential in ensuring the safety of both the passengers in the vehicle and other road users.

The need for a takeover request arises from the fact that while autonomous vehicles are designed to operate without direct human input, they are not yet capable of fully replacing human drivers in all driving scenarios [32]. For example, they may encounter unexpected obstacles or weather conditions that their sensors and algorithms are unable to interpret correctly. In such cases, the autonomous vehicle's control system alerts and hands over control to the human driver, who can then take appropriate action to safely navigate the situation.

The decision to initiate a takeover request is made by the autonomous vehicle's control system based on a range of inputs from sensors, cameras, and other data sources [32]. When the system determines that it is unable to safely continue driving, it sends a signal to the human driver requesting them to take control of the vehicle. The human driver is then responsible for safely navigating the vehicle until the autonomous control system is able to resume control.

It represents a key area of ongoing research and development in the field, as engineers work to refine the algorithms and sensors used to detect potentially hazardous and

uncertain situations, and develop better methods for alerting human drivers to the need for a takeover request.

Ultimately, the successful implementation of takeover requests will be important in achieving widespread adoption of autonomous vehicles. The presence of an effective TOR system should help overcome the fear of surrendering human control to skeptics of autonomous driving.

A way to optimize this TOR process could be to develop an AI model that replicates human behavior. With this model, it would be possible to test TOR in a multitude of situations without the need for human testing [1]. To create a model with this goal, imitation learning, a technique rooted in machine learning, has emerged as a promising avenue. By leveraging real-world driving data, imitation learning seeks to train AI models to mimic the nuanced behaviors and decision-making patterns exhibited by human drivers. As engineers continue to refine the algorithms and methodologies involved, the use of imitation learning in crafting AI drivers that understand and replicate human behavior post-TOR can become an integral aspect of achieving a smoother and safer driving experience in the area of autonomous vehicles.

The aim of this work is to create a model using these techniques and to determine an accurate way to compare this model with human drivers. The model created in this work can be used in future to help optimize the TOR process and indeed, with adjustments, other human interactions with automated machines.

1.2 Project Description

This paper investigates if machine learning can be used to help in optimizing the implementation of takeover requests. This is done by creating a post-takeover request (TOR) driver model to accurately simulate situations after a TOR has been issued. It asks how can an imitation learning-based driver model be effectively evaluated, and what is the performance of this model compared to human drivers' reactions when given a TOR? The project involves the development of a system that can be trained to recognize post-TOR request driver patterns and imitate the behavior of an experienced driver. The project uses multiple machine learning algorithms to recognize patterns in the data collected from experienced drivers after a TOR is issued. Specifically, the algorithms being used are Behavioral Cloning (BC), Generative Adversarial Imitation Learning (GAIL) and a reinforcement learning algorithm known as Proximal Policy Optimization (PPO). This combination allows for the agent to replicate human behavior accurately without needing days of demonstrations to be recorded. The project also includes an evaluation of the performance of the created driver model, comparing it to real world drivers. This comparison is achieved by checking if there is a statistically significant difference between the actions performed by the model to the actions performed by human drivers. These metrics are the outcome of the scenario, the average speed of the vehicle and the reaction time to the TOR.

1.3 Aims and Objectives

This research aims to investigate the reliability of using a combination of imitation and reinforcement learning to build a potentially realistic driver model to simulate situations after a TOR has been issued.

The first objective is to create an AI driving model that could replicate human behavior patterns when given a TOR. The second objective is to evaluate the performance of the proposed driver model in terms of its ability to accurately mimic human driving behavior. The third objective is to attain any ancillary information achieved during the process of developing and evaluating the AI models.

1.4 Project Scope

The project includes an environment in *Unity* that contains a procedural road network which is used to test and teach the driver model. The environment includes a car controller that is used by the agent and human demonstrators to drive a car around the road. This environment is further used to display the abilities of the agent in its current form. Extensive testing of the agent is additionally included and an approach to comparing AI driver models with humans is discussed and used to evaluate the performance of the AI agents.

1.5 Thesis Road-map

1.5.1 Related Work

Chapter 2 explores background research related to the area of autonomous vehicles, takeover requests, imitation learning and machine learning in *Unity*. Following this, a cross-examination of similar driver models created with and without imitation learning will be completed. Finally, this chapter will discuss an array of other relevant research completed for this project.

1.5.2 Project Design

Chapter 3 delves into the software and project methodology chosen and how these choices came to be. Following this, a detailed technical architecture of the environment and machine learning aspects with be explored. Finally, an overview of the approach taken to ensure accurate testing and results will be discussed.

1.5.3 Project Development

Chapter 4 breaks down the entire development process of the system regarding the technical architecture outlined in the design chapter. All the challenges encountered in the development process and the development to this point will be explored.

1.5.4 Testing and Evaluation

Chapter 5 describes how all the testing and evaluation of the project was executed. Each phase of testing will be described in detail. How the comparison between the agent and real drivers is tested and evaluated will be discussed and the results of the tests will be examined.

1.5.5 Conclusions and Future Work

Chapter 6 reflects on the results of the project and will discuss the conclusions drawn and the future work recommended.

Chapter 2

Related Work

This chapter reviews related work done in the area of autonomous vehicles, takeover requests, machine learning and other relevant research for the project.

2.1 Autonomous Driving

2.1.1 Levels of Automation

One of the fundamental concepts in autonomous driving is the classification of different levels of automation. The Society of Automotive Engineers (SAE) has defined six levels of driving automation, ranging from Level 0 to Level 5 [40]. Each level represents a different degree of autonomy and human involvement in the driving process. Muhammed et al. [20] provides an overview of these levels and highlights the challenges and future directions in autonomous driving.

Here is a breakdown of the different levels:

Level 0: No Automation The driver is fully responsible for all aspects of driving. There is no automation present in the vehicle.

Level 1: Driver Assistance The vehicle has basic driver assistance features, such as adaptive cruise control or lane-keeping assist. The driver is still responsible for most driving tasks, but the vehicle can assist in specific areas.

Level 2: Partial Automation The vehicle has advanced driver assistance systems (ADAS) that can control both steering and acceleration/deceleration. The driver must remain engaged and ready to take control at any time. Examples include *Tesla's Autopilot*.

Level 3: Conditional Automation The vehicle can handle most aspects of driving under certain conditions. The driver is not required to monitor the road constantly but must be ready to take control when prompted by the system. Level 3 vehicles have "environmental detection" capabilities and can make informed decisions for themselves, such as accelerating past a slow-moving vehicle [40]

Level 4: High Automation The vehicle can perform all driving tasks under specific conditions or within certain geographic areas. The driver may have the option to take control but is not required to do so. Level 4 vehicles are capable of completing an entire trip from origin to destination without driver input or intervention [38].

Level 5: Full Automation The vehicle is fully autonomous and can perform all driving tasks under all conditions. There is no need for human intervention or control. Level 5 vehicles are not yet commercially available.

It is important to note that while higher levels of automation offer increased autonomy, they also present challenges in terms of safety, reliability, and human-machine interaction. The transition between automation levels, particularly from Level 2 to Level 3 and beyond, requires careful consideration of driver behavior, takeover time, and system design [29] [19]. As the automotive industry continues to advance in autonomous driving technology, researchers and engineers are working to address these challenges and improve the safety and efficiency of autonomous vehicles at each level of automation [15] [7].

2.1.2 Driving Behavior during Takeover Requests

Understanding driver behavior during takeover requests is crucial for the safe and efficient operation of autonomous vehicles. Muto et al. [21] investigated the effect of driver postures on driving behavior during takeover requests. Their study sheds light on how different driver postures can influence the response time and decision-making process during a takeover request. Eriksson and Stanton [9] examine the takeover time in highly automated vehicles during noncritical transitions to and from manual control. Their research provides insights into the factors that affect the time it takes for drivers to regain control of the vehicle and resume manual driving.

2.1.3 Factors Influencing Takeover Request Timing

The timing of takeover requests plays a significant role in the interaction between the autonomous vehicle and the human driver. Radlmayr et al. [25] explore the effects of traffic density and automation reliability on takeover request timing. Their findings highlight how these factors can influence the decision-making process of drivers in determining when to request control from the autonomous system.

Overall, these studies contribute to our understanding of autonomous driving and the dynamics of takeover requests. They provide valuable insights into the levels of automation, driving behavior during takeover requests, and the factors that influence takeover request timing. By considering these factors, researchers and developers can design more effective autonomous systems that prioritize safety and enhance the overall driving experience. The work developed in this thesis builds upon the research by providing a new way to explore and investigate TOR situations.

2.1.4 Takeover Request Optimisation

Other work has been conducted to determine the most effective situations and circumstances under which a TOR is made. Kuen et al. [16] proposed a method to optimize TOR through the use of reinforcement learning. The car would learn the best times and situations to request a takeover by rewarding the agent when it would stay on the road after a takeover. This method used a simple post-TOR driver model, that essentially drove in a straight line after the TOR. It was determined that the best times to issue a takeover were on straights or after a corner while issuing a takeover request

in the middle of a corner was the worst time. The agent learned this behaviour and would issue a TOR at the optimized times. The work did mention that for future work, a more complex and human-like post-TOR driver model could be useful. A complex driver model would still allow for the advantages of machine learning as endless data could be recorded and could potentially match the performance of using human testers. This provided the inspiration for this thesis and the basis for the driver model created.

2.2 Machine Learning for Autonomous Driving

There has been a growing level of research and industry interest in using imitation learning for autonomous driving applications. For example, Codevilla et al. [5] proposed a framework for imitation learning that uses demonstrations from a human driver to teach a robotic car to drive in a complex urban environment. Similarly, Bansal et al. [3] used imitation learning to train a self-driving car to drive in a simulated environment. Overall, imitation learning has been applied to create models for complex autonomous driving tasks. Despite this, it has not been used for the specific scenario of driver behavior in a response to a TOR.

Another branch of machine learning that has been used in autonomous driving and the simulation of drivers is with reinforcement learning [30]. A recent prominent approach was created with *Gran Turismo's Sophy* AI. Deep reinforcement learning was utilized to create the AI. A complex system of rewards was carefully managed to guide the AI into making human-like decisions when racing [33]. This approach resulted in a convincing driver AI that was capable of being faster than the best *Gran Turismo* drivers. There were some downsides to this method, the biggest being the need for extensive and powerful hardware (over 1,000 *PlayStation 4* consoles were used in parallel for training [33]) to train the agents for an inordinate amount of time. The time to prepare the agents and adjust reward functions was also considerable. It is also worth noting that this approach is particularly well suited to a single track and does not scale well to multiple environments.

Reinforcement learning can be an effective technique to create human-like driving AI, however as stated above it can quickly become impractical if used alone. A combination of reinforcement learning and imitation learning was chosen to develop the AI models in this work.

2.3 Comparing Humans and AI

Research has also been made in the area of comparing real drivers' behavior with the driving models of autonomous vehicles. Remonda et al. [26] proposes a solution on how to compare the behavior of human and autonomous drivers under an aggressive driving scenario. The research was based on an autonomous driver that was created using reinforcement learning and some of the same principles apply for the driver model in this work. The analysis was mainly focused on lap times and sector times of the participants, the participants additionally took part in a NASA-TLX survey to measure workload. Other analysis included recording driver inputs for steering, accelerator and brake. The work was focused on performance and thus emphasized the lap times over

driver input. Instead of using lap and sector times, it was chosen to use average speed as one of the factors to compare between the AI model and human drivers. Driver inputs are also compared, specifically reaction times and reaction methods similar to the previous work. In order to determine if there is a statistically significant difference between the data of the AI model and human drivers a paired samples T-test can be used. The paired samples t-test is a statistical analysis technique designed to assess whether there is a significant difference between the means of two related groups [27]. The relation between the groups can be a particular statistic that is shared. In the context of this thesis, these related groups would be specific scenarios, such as different types of road segments and the place where a TOR has been issued.

2.4 Evaluating Machine Learning Models

There have been numerous research papers in the area of evaluating the performance of machine learning models. One such common technique of evaluating models is through using the mean squared error (MSE) [24], which is calculated as:

$$\text{MSE } = (1/n)^{\star}\Sigma\left(y_i - f\left(x_i\right)\right)^2$$

Where n is the total number of observations, y_i is the response to the i^{th} observation and $f(x_i)$ is the predicted response to the i^{th} observation. A small MSE value indicates a closer alignment between model predictions and actual observations, a big MSE value suggests non alignment between the model predictions and actual observations.

In practice, the following process is used to calculate the MSE of a given model. First, the dataset is split into two sections, the training data and the testing data. The training data is the data that is used to train the model and the testing data is left out of training and is solely used to test the model after it has been trained. Typically the proportion of training data to testing data is somewhere around 80:20 [17]. The test MSE value gives an idea of how well a model will perform on data it has not previously seen. A problem with this technique is that the MSE value can vary greatly depending on which portion of the dataset is used for training and testing. To combat this issue Leave-One-Out Cross-Validation (LOOCV) [22] can be used.

LOOCV works similarly, it splits the dataset like before into a training set and a testing set. The difference is that the training set consists of all the data except for one entry and the testing set is the one data point not included in the training set. This process is then completed multiple times over the course of the entire dataset. The final MSE value is then calculated as the average MSE value of all the MSEs for each case.

$$\text{Final MSE } = (1/n)^{\star}\Sigma MSE_i$$

Although MSE is conventionally associated with regression tasks, (the methodologies frequently employed alongside MSE) the MSE value can be effectively adapted for this thesis' objectives. To this extent, LOOCV is used in combination with a paired samples T-test and a standard error of SPDC (sample proportions difference calculation) to evaluate the overall performance of the models.

Chapter 3

Project Design

This chapter delves into the design process of every aspect of the thesis and project. The decisions made, issues encountered and alternative methods worked with, are all outlined.

3.1 Thesis Methodology

A clear methodology was defined early on in the design process to answer the research questions asked by this work. The first element was to develop a project that included a procedurally generated road, a car and a way to train and test the AI agents. The next step in the process was to gather data, this data included all the necessary data to train the AI models and the all the data necessary to test the models in comparison with the human drivers. The final parts of this methodology were to ultimately train the AI agents and then test them against the human drivers.

3.2 Project Methodology

In this master thesis, a feature-driven development approach was adopted to create and implement the project. Feature-driven development (FDD) is a software development methodology that prioritizes the identification and implementation of specific features or functionalities.

A comprehensive analysis of the project requirements was conducted at the beginning. Based on this analysis, a prioritized list of features was defined that the project needed to include. The goal was to deliver a set of valuable and usable features early in the development process.

The development process was broken down into short, time-boxed iterations, each focused on delivering one or more features from the prioritized list. The first step in each iteration was the detailed design of the selected features. This involved creating clear specifications and requirements, defining the user interface if necessary, and outlining the interactions with other components of the system.

Once the implementation was complete, testing and quality assurance measures were applied to validate the functionality of the features. Any defects or issues identified during testing were promptly addressed and resolved.

As the project progressed through each iteration, new features were continuously added and existing ones were enhanced. The iterative nature of FDD facilitated effective responses to changing requirements such as additional functionality or data recording. Throughout the development process, a clear and up-to-date feature list was maintained, ensuring that each feature was tracked and documented.

In conclusion, adopting a feature-driven development approach in this master thesis enabled for each aspect of the project to get the attention it needed and allowed for multiple iterations to create improved functionality.

3.3 Road Generation

A substantial body of research has been dedicated to the domain of procedural road generation such as Teng and Bidarra [31], Cura et al. [6] and Galin et al. [11]. Predominantly, the focus of these investigations has centered on the creation of road networks as opposed to individual road segments. One notable methodology put forth involves the implementation of a chunk-based system [31]. This type of system consists of discrete chunks, each representing an area that encompasses a road network or a specific segment of a road. These chunks are the spawned next to each other following a particular set of rules. These rules can be random, other times these rules can follow Geographic Information System data [6].

Other methods introduced were to use a Lindenmayer System (L-system). The computer game *NeoCab* suggested an approach to procedural road generation involving L-systems [35] along with the use of chunks. The L-system would define the layout of the road network and the chunks would define the style of each individual road segment. This approach does run into a problem since each type of road segment needs to be predefined and created before run-time. L-systems have additionally been used exclusively to create the road network [23] and to create geometry itself [18]. These approaches can be effective in creating a convincing grid style road network however they lack realism when being used for a single road segment [11].

Addressing the specific context of individual road segments, a spline-based methodology has gained attention. This technique revolves around the manipulation of various Bézier curves to create the smooth and twisting style of a road [10].

Despite an exhaustive trawl of current research, none was found on the generation of a road segment, which is what is used in this thesis.

The road generation aspect of the thesis went through many iterations over the course of the project and was based on the previous research conducted in the field as mentioned above. The first methods to generate a procedural road used a chunk technique. Each chunk contained a different road segment such as a straight or a turn. These chunks would then be placed after each other to create a randomly generated road. Firstly, these chunks were placed randomly but this resulted in a number of issues. The generated roads would either be too erratic and unrealistic or would tend to turn in on themselves, seen in figure 3.1. The other problem with this kind of technique was a lack of control over the generated road so it would be difficult to repeat the exact same conditions of the environment when testing the AI model.

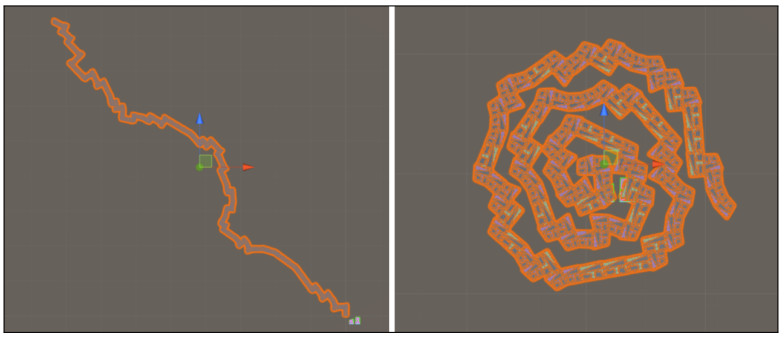


Figure 3.1: The failed chunk system, erratic example left and turning in on itself example right.

3.3.1 Lindenmayer System

To combat these problems a structure to the chunks was needed. This was achieved with the use of a Lindenmayer system (L-system). An L-system, is a framework that represents the development of complex structures. It offers a powerful means of describing the iterative production of intricate patterns and shapes through a set of rewriting rules and an initial axiom. Fundamentally, an L-system consists of three primary components: an alphabet of symbols, a set of production rules, and an axiom. The alphabet defines the basic units or symbols that make-up the system. These symbols represented the different types of road chunks. The production rules establish the transformations that occur at each iteration of the system. Essentially, they define how each symbol in the axiom or previous iteration is replaced by a sequence of symbols. The axiom serves as the initial starting point or seed for the system, providing the foundation from which subsequent iterations derive. At each iteration, the L-system applies the production rules to the symbols present in the previous iteration. By repeatedly executing this process, the system generates a sequence of strings. These strings represent the overall structure of the road. The use of the L-system allowed for a wide variety of generated roads while allowing for enough control. Below is an example of a simple L-system and an early version of the road generator using the L-system can be seen in figure 3.2:

L-System Example:
Alphabet of symbols:

- F: Move forward and draw a line
- +: Turn right by a fixed angle
- -: Turn left by a fixed angle

Production rule:

- F $\rightarrow$ F+F-F

Axiom:

- F

Iterations:
Iteration 0 (Axiom):

- F

Iteration 1:

- F → F+F-F

Iteration 2:

- F → F+F-F+F+F-F-F+F+F-F+F+F-F+F-F-F+F-F+F+F

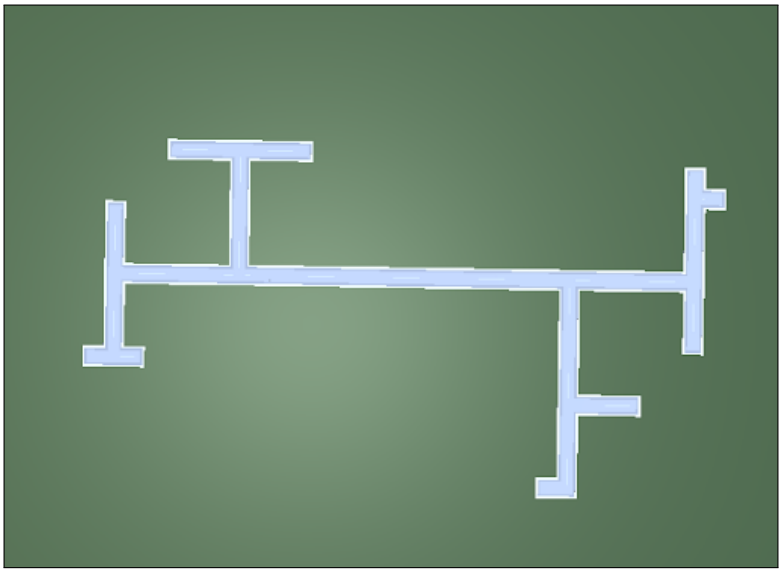


Figure 3.2: A more complex L-system road.

And so on, as you continue applying the production rule to the symbols in each iteration, the pattern will grow and become more complex.

Despite this, it had one fundamental problem of relying on the chunks. This meant that for each possible rotation of a turn a unique chunk needed to be made. The problem gets magnified when different possible lengths of a turn are considered. To solve this the final iteration of the road generator was considered.

3.3.2 Spline Road Generator

The final road generator uses a spline system to create the road structure. This allowed for similar control over the road structure like the L-system but for greater variation as each type of road segment did not need to be specified.

The system works in a number of steps, starting with creating a Bézier path. The Bézier path is a path made by stitching together any number of cubic Bézier curves. A single cubic Bézier curve is defined by four points: two anchor points (start and end points) and two control points. The shape of the curve is affected by the positions of the two control points. By randomizing these positions multiple times over the course of an entire road, it can be altered in a multitude of ways while still retaining a standard road shape.

The next step is to define a set of vertices (points) along the Bézier path. A constructor takes in the Bézier path and creates a vertex path. This is done to achieve a number of aspects to aid in simulation and testing of the environment. Before the vertex path

is created, a car cannot move at a constant speed over a Bézier curve directly because Bézier curves do not represent linear distance uniformly along their length. When you move along a Bézier curve, a parameter "t" typically represents a percentage of the total path length covered, but the distribution of points along the curve is not uniform. This means that equal increments of "t" do not result in equal distances covered along the curve. As a result, moving at a constant speed along the Bézier curve would require more complex calculations and adjustments in each frame of movement. To enable constant speed movement along the Bézier curve, the curve needs to be sampled and converted into a vertex path, where the points are distributed more uniformly along the path. The vertex path essentially acts as a discretized representation of the Bézier curve, which facilitates constant speed movement and other calculations related to the path.

The constant speed is needed for when the simple AI is used to control the car pre-TOR. If the car travelled at different speeds, the testing and learning of the post-TOR AI model would be negatively affected. The vertex path also allows for more improved path following for the simple AI so it can traverse the road accurately.

Additionally, the vertex path has finer details that can be controlled such as a minimum angle. These finer details are essential as they prevent the road from following the Bézier curve too closely and being too unrealistic.

The most important aspect of the vertex path is to create a base that the actual road geometry can be built on. The script to create the geometry uses the provided vertex path to create a road mesh along the path. It calculates the additional vertices, UVs (texture coordinates), normals (perpendicular vectors to the surface), and triangles (actual mesh) required to form the road geometry.

The top surface of the road is represented by two rows of vertices, creating a flat road surface. The bottom surface of the road is formed by offsetting the top vertices along the path's normal direction by a specified thickness value. The sides of the road are formed by duplicating the top vertices and offsetting them to create a box-like shape.

Once the road geometry has been generated, collisions for the road are created. The collision system is generated by *Unity* using the road geometry. The collisions are used to tell when the car is on the road. An example of the final road generator can be seen in figure 3.3

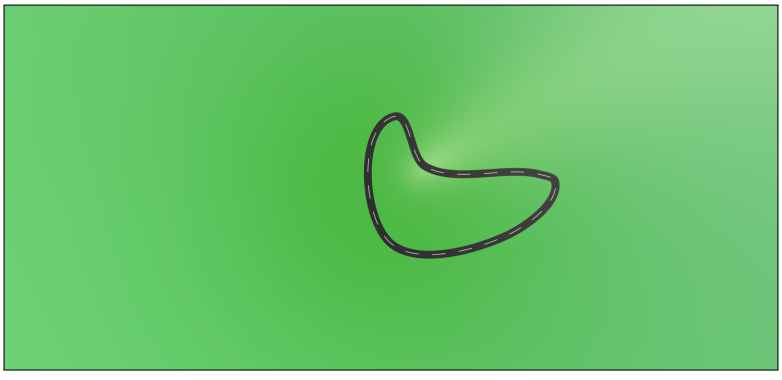


Figure 3.3: The Spline Road system.

3.4 Vehicle Controller in Unity

Utilizing a potentially realistic car controller was essential for the results from this thesis to be accurate. Previous research concerning car controllers encompasses two primary branches, each delving into distinct aspects of vehicle control and both of which being integral to the success of this work. The first of which being research into the actual input devices used to control the car. Some studies compare input devices like keyboards and joysticks [2], while others introduce novel methods that leverage gestures [13] and steering wheels [4] for vehicular control.

The comparative analysis between keyboard and joystick utilization yielded minimal difference, signifying limited influence on vehicular control performance. Conversely, using the real apparatus to control a vehicle was found to have significant impact of driver performance. Moreover, the implementation of a steering wheel accentuated the performance difference between people with real driving experience and those lacking such exposure. For these reasons, it was clear that the integration of a realistic steering wheel was necessary for the work.

The other branch of research into car controllers addresses the simulation of vehicular physics and the interaction with an environment. Generally it was stated that while the *Unity* game engine, can have realistic physics, these physics are not inherently suited to car dynamics [8]. Consequently, to achieve realism, proprietary car controllers that manipulate the physics engine in *Unity* are required [34]. One such proprietary car controller that is used is the realistic car kit from Mehdi Rabiee [36]. This specialized toolkit contains an intricate template for realistic physics simulations across diverse car types, while providing the flexibility to customize the controller parameters to suit the requirements of individual projects. This balance between simulated physics and tailored control mechanisms remains pivotal for achieving an authentic and immersive vehicular experience within the *Unity* game engine. The vehicle control setup used in this thesis was designed and implemented with all of this research and information considered.

3.5 Machine Learning

The approach to the machine learning aspect of the thesis involved many iterations. The final approach adopted combines multiple types of imitation and reinforcement learning to create an AI model that is not statistically significantly different from human behavior.

3.5.1 Behavioral Cloning

Behavioral cloning (BC) is a prominent approach within the field of imitation learning, where an agent, is trained to imitate the behavior of an expert by learning from the expert's demonstrated actions. The expert may be human or another AI and in the case of this thesis the expert term is used in reference only, none of the participants were professional drivers and would not be considered experts in the field of driving. The decision to use average drivers was chosen as the agent should act like the average driver rather than as a professional driver.

The algorithm aims to replicate the expert's decision-making process and actions,

enabling the agent to perform a specific task or behavior in a similar manner such as driving a car after a TOR has been issued.

When applying BC to this project, the first step involved collecting a dataset of driving demonstrations performed by a human expert. This dataset comprises pairs of input states: sensor readings from the car's environment, along with corresponding expert actions such as steering commands, acceleration, and braking decisions.

To train the model using BC, the collected dataset is utilized as the training data. The model is then trained to predict the appropriate driving actions based on the input state, mimicking the expert's behavior.

During training, the model is optimized to minimize the discrepancy between its predicted actions and the expert actions in the training dataset. This is achieved by minimizing a loss function which quantifies the dissimilarity between the predicted and expert actions. By iteratively adjusting its internal parameters, the model gradually learns to replicate the expert's behavior and make similar driving decisions when presented with similar input states.

There are certain considerations and challenges associated with BC. One significant challenge is the distributional mismatch between the training data and the conditions that the car may encounter in the simulated environment. This discrepancy can arise due to the procedural nature of the road segments that change on every single episode and may not repeat a particular section of road. As a result, a model trained using only behavioral cloning may struggle to generalize effectively to unseen situations. Additionally, behavioral cloning does not inherently possess the capability to handle unanticipated scenarios or make decisions beyond the expert's demonstrated actions. Since the model solely imitates the expert, it may fail to handle novel situations that were not present in the training data. Consequently, the agent's performance may be limited in scenarios that deviate from the demonstrated behavior. Both of these problems with BC are combated by using other techniques. In this project, it was decided to use BC as a pre-training process before other techniques are used.

3.5.2 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) is a novel approach in machine learning that combines the principles of imitation learning and generative adversarial networks (GANs). The objective of GAIL is to enable an agent to learn a policy by imitating expert behavior through a two-player adversarial game.

In the context of this project, GAIL is applied to generate an autonomous driving policy that emulates the driving behavior of a human driver. The imitation learning component of GAIL involves training a policy to imitate the driver's actions. This is achieved by using a dataset of demonstrations provided by the driver (the same dataset as is used with BC), where each demonstration consists of the state of the car and the corresponding action taken by the driver in that state.

The core idea behind GAIL is to use a discriminator network, which is trained simultaneously with the policy network. The discriminator's role is to distinguish between the expert demonstrations and the actions generated by the policy network. The policy network, on the other hand, aims to generate actions that are indistinguishable from those of the driver, effectively fooling the discriminator. This adversarial process encour-

ages the policy network to improve its performance and generate actions that closely resemble the expert's behavior.

During the training process, the policy network and the discriminator network play a minimax game. The policy network aims to maximize the discriminator's error rate by generating actions that are difficult to distinguish from the driver's actions. On the other hand, the discriminator network strives to minimize its error rate by correctly classifying the source of each action. The training iteratively updates both networks until an equilibrium is reached, where the policy network successfully imitates the driver's behavior, and the discriminator can no longer differentiate between the two sources of actions.

Once the GAIL training process is complete, the policy network can be used to drive the car autonomously. By learning from the driver demonstrations, the policy network has acquired the ability to make appropriate decisions and control the car in a manner that emulates the behavior of the driver. The GAIL framework allows for effective knowledge transfer from the human to the autonomous agent, enabling the agent to perform complex tasks, such as driving, with a high level of proficiency.

One possible issue that can occur with any imitation learning task is that of survivorship bias. Survivorship bias refers to the logical error that occurs when only successful or surviving entities are considered in an analysis, while those that failed or did not survive are overlooked or excluded. This bias can lead to distorted conclusions or inaccurate perceptions of reality by focusing solely on the observed outcomes and neglecting important information from the failed or non-surviving cases. In the case of this project, survivorship bias occurred when the policy network would be too good at deceiving the discriminator based on the agents perception of the environment. It would see an increase in reward and simply stay in place when the perception of the environment would provide the most award. To combat this, two factors were included in the learning process.

The first of these solutions was to include a greater emphasis on the reward gained by the policy network when it beats the discriminator based on its actions rather than solely on its perception of the environment. The other solution was to include a small element of reinforcement learning with extrinsic rewards that would encourage more human like behaviour.

3.5.3 Extrinsic Rewards & Reinforcement Learning

Extrinsic rewards play a significant role in the field of reinforcement learning, a branch of machine learning concerned with developing algorithms that allow an agent to learn optimal decision-making policies through interactions with an environment. In this context, extrinsic rewards are external signals or feedback provided to the agent to guide its learning process.

The agent takes actions in the environment, and based on these actions, it receives feedback in the form of rewards or penalties. The primary objective of the agent is to maximize its cumulative reward over time by learning a policy that dictates the optimal actions to take in different states of the environment.

In the context of teaching an agent to drive more human-like, the reinforcement learning process involves the agent learning an optimal driving policy by receiving ex-

trinsic rewards as feedback. The learning process involves updating the car's policy based on the observed rewards and the perceived value of different state-action pairs. Initially, the car explores the environment by taking actions based on the outputs of the imitation learning algorithms used. Gradually the agent is encouraged to act more generally through the use of extrinsic rewards such as average speed and ability to stay on the road. These extrinsic rewards guide the agent towards learning a policy that aligns with human driving norms and prevent overfitting.

It is important to note that the design of extrinsic rewards required careful consideration. Poorly designed rewards could have lead to unintended behaviors or exploits by the agent, as it may solely focus on maximizing the rewards without adhering to the desired human-like driving behavior. If the reward was too great also, the agent would end up ignoring the reward signal from all the imitation learning algorithms and not produce human-like behavior. Balancing the reward system to encourage human-like driving was crucial to ensure the agent's behavior is natural.

Chapter 4

Project Development

In this chapter an overview of the entire system is giving along with a detailed description of the main scripts used in the programming of the project.

4.1 System Overview

The overall system consists of a number of aspects. There is the road generator, machine learning models, pre-TOR AI, TOR system, car controller, wheel integration and simulation environment. There is a few communication layers however most of the system is handled within *Unity* this includes the road generator, pre-TOR AI, TOR system, car controller and simulation environment. The machine learning models use a python interface to communicate between the algorithms and the simulation environment in *Unity*. The wheel uses a logitech SDK to translate the wheel inputs so they can be used in *Unity*.

The software system comprises several integral components, each playing a crucial role in achieving the desired functionality and performance. These components can be broadly categorized into the road generator, machine learning models, pre-TOR AI, TOR system, car controller, wheel integration, and simulation environment. While certain communication layers exist, the majority of the system's operations are executed within the *Unity* engine, which handles the road generator, pre-TOR AI, TOR system, car controller, and the simulation environment.

Another element is the machine learning models, a crucial component that empowers the AI agent to learn from experiences, make decisions, and refine its behavior. There is a Python interface that facilitates seamless communication, enabling the exchange of crucial information between the AI agent algorithms and the *Unity* environment, thus facilitating the learning process.

Furthermore, the wheel integration component integrates external hardware. A Logitech SDK is used to seamlessly incorporate inputs from the physical steering wheel into the environment. This enables a more immersive experience for users and facilitates intuitive control over the car.

In summary, the software system is a comprehensive amalgamation of various components, each contributing significantly to the development and training of an AI agent capable of driving autonomously within the Unity environment. The integration of ma-

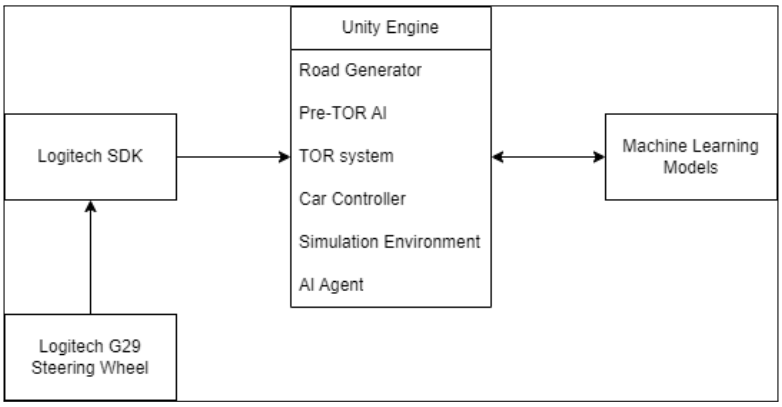


Figure 4.1: System Architecture Diagram.

chine learning models, the TOR system, pre-TOR AI, car controller, wheel integration, and the flexible Unity simulation environment work in harmony to create a dynamic and evolving AI-driven driving experience.

4.2 Unity Machine Learning Agents

The *Unity ML Agents toolkit*[14] is a comprehensive and versatile framework designed for developing and implementing machine learning algorithms within the *Unity* game engine [37]. Developed by *Unity*, this toolkit combines the power of machine learning with the immersive and interactive capabilities of *Unity*, facilitating the creation of intelligent agents capable of learning and interacting within virtual environments. The *Unity ML Agents toolkit* provides a rich set of tools and resources for training and evaluating intelligent agents. It offers seamless integration with *Unity*, allowing the agents to perceive the virtual environment through sensors and interact with it through actuators, thereby enabling complex and interactive decision-making processes. By leveraging the extensive features and functionalities of *Unity*, such as realistic physics simulations, rendering capabilities, and scene management, the toolkit allows for the development of sophisticated and lifelike environments for training agents. This is a particularly useful for the thesis as it allows for endless possibilities for training scenarios. At the core of the *Unity ML Agents toolkit* is the implementation of machine learning algorithms in a Python layer that can be utilised within *Unity*. The toolkit supports various reinforcement learning paradigms as well as imitation learning algorithms such as Behavioural Cloning (BC) and Generative Adversarial Imitation Learning (GAIL). Additionally, it provides tools for data collection, pre-processing, and visualization, enabling efficient analysis and monitoring of the learning process.

Previous research has used the *Unity ML Agents toolkit* to create human-like artificial intelligence using imitation learning. Rubak [28] did a feasibility study on using Unity to create agents through imitation learning. The thesis tried to create AI players for a first person shooter game using only imitation learning. Rubak determined that it is possible to use *Unity* to create these agents, however, training times can be long when more complex behaviors are desired. The driver model proposed in this paper would not

be affected by this issue as the desired behaviors are less complex than this particular example.

It was for the reasons outlined here that it was chosen to use the *Unity ML Agents toolkit* to complete the work necessary in this project.

4.3 Road Generation

The road generation consists of multiple scripts working together to create the road. In the design section an overview of the system was discussed, here more detail into the use of each script is explained.

4.3.1 BezierPath

The class 'BezierPath' represents the Bézier path and provides methods for working with the path. The path is defined by a list of 3D points. Each point in the list represents either an anchor point or a control point. The anchor points are the start and end points of each Bézier curve, and the control points determine the shape of the curve between the anchor points.

There is different control modes which determines how the control points behave. The implemented modes are Aligned, Mirrored, Free, and Automatic. In Aligned mode, the controls stay in a straight line with their anchor points. In Mirrored mode, they stay equidistant from their anchor points. In Free mode, there are no constraints. In Automatic mode, the controls are automatically placed to try to make the path smooth. The mode used in testing was 'aligned' as it provided the least amount of complications when procedurally generating the road.

New anchor points to the path can be added to the path, inserted, deleted or moved to allow for the procedural nature of the road.

The path is closed, meaning the last anchor point connects back to the first one, forming a loop.

The path can exist in 3D space (xyz) or 2D space (xy). In 2D space, the points are clamped to the respective plane, so z-coordinates are ignored. The path was kept in 2D space to reduce an additional layer of randomization to the testing.

The system can also calculate the bounding box of the path and provides the option to flip the normal vectors by 180 degrees.

4.3.2 VertexPathUtility

This class contains a method which takes a BezierPath and additional parameters as input. The purpose of this method is to split the input Bézier path into a series of vertices based on the angle error between consecutive points on the path. The main parameters used are:

maxAngleError: This determines the maximum allowable angle error between consecutive points. If the angle between points exceeds this threshold, a new vertex is added to the path.

minVertexDst: This parameter sets the minimum distance between consecutive vertices. If the distance between vertices exceeds this threshold, a new vertex is added

to the path.

accuracy: This parameter controls the granularity of the path splitting process, affecting the number of divisions in each Bézier curve segment.

The method proceeds by iteratively evaluating the Bézier curve and measuring the angles at different points. If the angle error or the distance between vertices exceeds the specified thresholds, a new vertex is added to the path. The process continues until all segments of the Bézier path have been traversed, resulting in a list of vertices that effectively represent the original path while adhering to the given angle and distance criteria. The resulting data is encapsulated in the PathSplitData class, containing the list of vertices, tangents, cumulative lengths, and anchor vertex mapping.

4.3.3 VertexPath

This class represents the vertices along the Bézier path and provides a number of functions that are used by other parts of the system. It takes the vertex path from the VertexPathUtility class. The various functions are used by the pre-TOR AI to navigate the road and by the RoadMeshCreator script to create the geometry of the road.

4.3.4 RoadMeshCreator

The essential part of this script is the create road mesh function.

The function is responsible for generating the road mesh based on the provided path. It uses various arrays to store vertices, UV coordinates, and normals, and several integer arrays to store the triangle indices for different parts of the road mesh. The function starts by iterating through each point in the given path. It calculates the position of vertices on the top and bottom surfaces of the road at each point. Additionally, it duplicates some vertices to achieve flat shading for the sides of the road. The function also assigns UV coordinates and normals to the vertices, which are essential for texturing and lighting of the road mesh. Triangles are then defined to connect the vertices and form the surface of the road. Different triangle arrays are used to handle the top, bottom, and sides of the road separately. The resulting mesh is stored in the mesh variable and used to render the road and the collider is updated.

```
1     Vector3[] verts = new Vector3[path.NumPoints * 8];
2     /.../
3     int numTris = 2 * (path.NumPoints - 1) + ((path.isClosedLoop) ? 2 : 0);
4     int[] roadTriangles = new int[numTris * 3];
5     int[] underRoadTriangles = new int[numTris * 3];
6     int[] sideOfRoadTriangles = new int[numTris * 2 * 3];
7     int vertIndex = 0;
8     int triIndex = 0;
9     int[] triangleMap = { 0, 8, 1, 1, 8, 9 };
10    int[] sidesTriangleMap = { 4, 6, 14, 12, 4, 14, 5, 15, 7, 13, 15, 5 };
11
12    for (int i = 0; i < path.NumPoints; i++) {
13        Vector3 localUp = (usePathNormals) ? Vector3.Cross (path.GetTangent (i),
      path.GetNormal (i)) : path.up;
14        Vector3 localRight = (usePathNormals) ? path.GetNormal (i) : Vector3.Cross (
      localUp, path.GetTangent (i));
15
```

```
16          // Find position to left and right of current path vertex
17          Vector3 vertSideA = path.GetPoint (i) - localRight * Mathf.Abs (roadWidth);
18          Vector3 vertSideB = path.GetPoint (i) + localRight * Mathf.Abs (roadWidth);
19
20          // Add top of road vertices
21          verts[vertIndex + 0] = vertSideA;
22          verts[vertIndex + 1] = vertSideB;
23          // Add bottom of road vertices
24          verts[vertIndex + 2] = vertSideA - localUp * thickness;
25          verts[vertIndex + 3] = vertSideB - localUp * thickness;
26
27          // Duplicate vertices to get flat shading for sides of road
28          verts[vertIndex + 4] = verts[vertIndex + 0];
29          verts[vertIndex + 5] = verts[vertIndex + 1];
30          verts[vertIndex + 6] = verts[vertIndex + 2];
31          verts[vertIndex + 7] = verts[vertIndex + 3];
32          /.../
33          // Set triangle indices
34          if (i < path.NumPoints - 1 || path.isClosedLoop) {
35              for (int j = 0; j < triangleMap.Length; j++) {
36                  roadTriangles[triIndex + j] = (vertIndex + triangleMap[j]) % verts.
    Length;
37                  // reverse triangle map for under road so that triangles wind the other way
    and are visible from underneath
38                  underRoadTriangles[triIndex + j] = (vertIndex + triangleMap[
    triangleMap.Length - 1 - j] + 2) % verts.Length;
39              }
40              for (int j = 0; j < sidesTriangleMap.Length; j++) {
41                  sideOfRoadTriangles[triIndex * 2 + j] = (vertIndex +
    sidesTriangleMap[j]) % verts.Length;
42              }
43          }
44          vertIndex += 8;
45          triIndex += 6;
46      }
47      /.../
```

4.4 Vehicle Control

For this project, the car controller aspect was facilitated using the *Realistic Car Kit* [36], a robust *Unity* plugin that offers a realistic physics template for car simulation and handles all the necessary logic for controlling the car. Leveraging this plugin proved to be effective, as it provided ample functionality that aligned with the project requirements, sparing the need to develop a car controller from scratch.

In order to interface the car controller system with the physical *Logitech G29* steering wheel and pedals, the *Logitech Unity SDK* was employed, along with an up-to-date driver script tailored for the specific steering wheel. To enhance the driving experience and mimic real-world conditions, a spring force was applied to the steering wheel, adding a sense of weight and authenticity to the vehicle handling.

A notable point encountered during integration was the need to adapt the acceleration and brake system of the *Realistic Car Kit* to line-up with the Logitech pedals. The original configuration tied the reverse gear to the accelerator axis, which required

careful reworking to ensure smooth and accurate vehicle control. The dead-zone for the pedals also required significant attention to fine tune. Additionally, the brake force was originally designed as binary, necessitating modifications to enable finer control over the vehicle's braking behavior. The steering wheel setup can be seen in figure 4.2

Figure 4.2: The Steering Wheel Setup.

By integrating the *Logitech G29* steering wheel and pedals with the *Realistic Car Kit* plugin, the project achieved a compelling and immersive driving experience within the Unity environment. The combination of realistic physics, responsive controls, and intuitive interface allowed for a seamless connection between the physical input devices and the virtual car, providing an effective platform for AI training and testing in a lifelike driving simulation.

4.5 Rudimentary Autonomous Vehicle AI

To establish a TOR, a driver model simulating an autonomous vehicle was developed. The model leverages the car controller to imitate the desired behavior of the autonomous vehicle before a TOR is issued. The model has access to the steering and accelerator inputs of the car, adjusting these parameters to maintain the vehicle's trajectory on the road.

The Rudimentary Autonomous Vehicle AI (RAVAI) operates by having an object follow the vertex points path that defines the road geometry. The RAVAI then compares the orientation of its forward vector to that of the object, adjusting the angle of its forward vector via steering to align with the road. This approach reduces side-to-side and jerky movement of the car, enabling it to follow the road more smoothly, as it does not go to every point on the road perfectly but rather imitates an object that does.

```
1 Vector3 normalizedDirection = targetGameObject.transform.position - transform.
      position;
```

```
2 float whichWay = Vector3.Cross(transform.forward, normalizedDirection).y;
```

The accelerator is controlled by limiting the car's speed within a minimum and maximum range. If the car's speed drops below the minimum, the accelerator is engaged, and if it exceeds the maximum, the accelerator is disengaged. This technique ensures that the car's speed remains stable and prevents excessive forward or backward movement. Keeping the range minimal improved testing as a constant speed for when a TOR was issued could be achieved.

4.6 Machine Learning

As stated in the design chapter, various different algorithms were used in conjunction with each other to create the AI driver model. The algorithms used were Behavioral Cloning (BC) and Generative Adversarial Imitation Learning (GAIL) from imitation learning and Proximal Policy Optimization (PPO) from reinforcement learning. Within *Unity ML Agents* a number of hyperparameters and variables are used to gain access to the algorithms.

4.6.1 Human Demonstrations

The core aspect to all of the AI models comes from human demonstration data. This data comprises of a multitude of pairs of actions and observations. The actions consist of the ways in which the human interacts with the environment while the observations are what the human sees in the environment at the moment the actions took place. The actions recorded are the use of the accelerator, use of the brake and steering angle. The observations recorded are a set of 20 ray-casts that describe where the vehicle is in relation to the road and how the road layout looks in front of the vehicle. The pairs of actions and observations were recorded every 5 frames which was approximately 20-30 times a second depending on the performance of the computer. The machine learning algorithms use this data to aid in the learning process as described earlier.

4.6.2 Behavioral Cloning

The implementation of BC in *Unity* is as described in the design chapter. The access to the BC algorithm is limited but allows for enough control to customise the effectiveness of the system.

```
1 behavioral_cloning:
2     demo_path: Demos/Tor_L.demo
3     steps: 50000
4     strength: 0.5
5     samples_per_update: 0
```

The demo path specifies the path to the demonstration data used for training. Demonstrations consist of the data that is collected from experts that guide the imitation learning process.

The steps indicates the total number of training steps or iterations the BC algorithm will perform. Each step corresponds to a single update of the policy network using a batch of data from the demonstrations.

The strength controls the influence of the behavioral cloning loss during training. The relatively high strength value, 0.5, indicates that the policy network's updates will be influenced to a greater extent by the imitation loss. This means the policy will try to closely match the expert actions in the demonstration data.

The samples per update determines the number of samples used to compute the behavioral cloning loss in each policy update. A value of 0 indicates that all available samples from the demonstration data are used in each update. This approach ensures that the policy network learns from a wide range of demonstrations.

4.6.3 Generative Adversarial Imitation Learning

The GAIL algorithm implementation in *Unity* follows the structure as described in the design chapter. The GAIL training involves an adversarial training loop where the generator tries to generate actions that fool the discriminator, and the discriminator aims to distinguish between expert and generated actions. This can be seen in the pseudo code segment below:

```
    for (int epoch = 0; epoch < num_epochs; epoch++) {
    var generatedActions = generator.GenerateActions(states);
    var discriminatorLoss = discriminator.Train(states, expertActions,
    generatedActions);
    var generatorLoss = -discriminatorLoss; // The generator seeks to minimize the
    discriminator's performance
    generator.Update(generatorLoss);
}
```

A loss function is used to guide this training process. The loss function quantifies the difference between the expected outcomes and the actual outcomes. For the generator, the loss function might encourage generating actions that lead to lower discriminator confidence. For the discriminator, the loss function measures its ability to accurately classify actions as either expert or generated.

```
    public float Train(List<State> states, List<Action> expertActions, List<Action>
    generatedActions) {
    var expertLabels = Enumerable.Repeat(1, states.Count); // Label for expert actions
    var generatedLabels = Enumerable.Repeat(0, states.Count); // Label for generated
    actions

    var expertLoss = ComputeBinaryCrossEntropy(states, expertActions, expertLabels);
    var generatedLoss = ComputeBinaryCrossEntropy(states, generatedActions,
    generatedLabels);

    var totalLoss = expertLoss + generatedLoss;
    totalLoss.Backward();

    optimizer.Step();

    return totalLoss.Value;
}
```

The hyperparamters and variables used with GAIL define the other aspects of the algorithm.

```
gamma: 0.99
strength: 0.01
```

```
network_settings:
    normalize: true
    hidden_units: 128
    num_layers: 2
    vis_encode_type: simple
learning_rate: 0.0003
use_actions: true
use_vail: false
demo_path: Demos/Tor_L.demo
```

Gamma is the discount factor used to balance immediate rewards with future rewards in reinforcement learning. A value close to 1 gives more weight to future rewards, promoting long-term planning.

The strength parameter indicates the strength of the adversarial imitation learning process. A small value like 0.01 means that the generator's policy updates are guided by a relatively small influence from the discriminator's feedback. This prevents the model from overfitting and allows the AI driver to adapt outside the state space.

The network settings specify the neural network that is used by the algorithm for the training process. Normalize being set to true results in the observations (what the agent sees in the environment) being normalized before entering the neural network.

The hidden units relate to the number of neurons in each of the hidden layers in the neural network, 128 indicates a large and potentially more expressive network architecture.

The next parameter dictates the amount of hidden layers in the network. More layers requires higher computational power but also can capture more complex relationships.

The last parameter of the network settings pertains to how visual data is encoded before being sent to the network. This is set to simple as no visual data is sent to the network.

Learning rate controls the step size taken during gradient descent optimization. A smaller learning rate of 0.0003 ensures cautious updates to the neural network's weights to prevent overfitting.

The last 3 parameters tell the network to use the actions of the agent to aid in learning, to not use a variation of the GAIL algorithm and to specify the path to the demonstrations recorded for training.

4.6.4 Proximal Policy Optimization

The implementation of PPO in *Unity ML Agents* provides a significant amount of customisation and control. The choices made reflect a balance between exploration, stability, and training efficiency.

```
trainer_type: ppo
    hyperparameters:
        batch_size: 2024
        buffer_size: 20240
        learning_rate: 0.0003
        beta: 0.005
        epsilon: 0.2
        lambd: 0.95
        num_epoch: 3
        learning_rate_schedule: linear
```

```
    network_settings:
        normalize: true
        hidden_units: 512
        num_layers: 3
        vis_encode_type: simple
    reward_signals:
    extrinsic:
        strength: 1.0
        gamma: 0.99
```

The trainer type specifies that the trainer being used is based on the PPO algorithm.

The batch size determines the number of samples used in each training batch. A larger batch size can provide more stable gradient estimates and potentially accelerate learning.

The buffer size refers to the size of the experience replay buffer. Experience replay is a technique that stores past experiences for training stability. The large buffer can help reduce correlations between consecutive samples.

The learning rate controls the step size during optimization. The small learning rate ensures cautious updates to policy parameters.

Beta is the coefficient for entropy regularization. Entropy encourages exploration by adding randomness to the policy. The small beta value emphasizes exploration.

Epsilon is the PPO clipping parameter. It bounds the ratio of new policy probabilities to old policy probabilities. A smaller epsilon constrains policy updates to be within a certain range.

Lambda is the Generalized Advantage Estimation (GAE) parameter. GAE balances the trade-off between bias and variance in estimating advantages for policy updates.

The number of epochs determines the number of optimization epochs within each batch. More epochs allow the network to update policy parameters more thoroughly.

The learning rate schedule specifies how the learning rate changes over time. Linear decay is used to gradually reduce the learning rate.

The network settings work the same as for GAIL, the inputs are normalized, there is a large number of neurons and hidden layers, and the visual encoding type is set to simple for the same reason.

Only an extrinsic reward signal is used, the strength controls the weight of the extrinsic reward signal relative to other reward signals. It is set to 1 as no other reward signal is used with PPO in this case.

The gamma works the same as with GAIL, as a discount factor for future rewards. The high gamma value prioritizes long-term rewards.

It is important to note that the use of PPO was carefully monitored to ensure that it would not negatively impact the performance of GAIL and BC but only enhance it and prevent some of the issues with those algorithms.

4.7 The AI Agent

For all of the machine learning algorithms to work, an AI agent had to be developed within *Unity*. This script handles all the behaviors in *Unity* and sends observations and information to be handled by the algorithms in the python layer. It is included as part of the car game object as it needs access to the car controller, to determine what the

car sees and where it is.

The observations are handled by ray-casts as they take significantly less processing power than images. There is a total of twenty ray-casts used to observe the environment. They are spread around the vehicle at various angles and distances, all they check is whether or not they hit the road or the ground.

All the actions undertaken by the agent are handled here. There are a total of three different actions, accelerate, brake and steer. All three of the actions are continuous between the values of zero and one, zero and one, and minus one and one respectfully.

The agent also defines when and how much rewards the agent receives in the PPO algorithm. The agent is rewarded for lap time as without using PPO, the agent would drive too slow and justify its slow speed by getting high rewards for observations only. With this addition, a closer emulation of human driving behavior while mitigating the risk of deviating excessively from the experts' actions was achieved.

```
1 //forward the values to vehicle
2 control.agentaccel2 = acc;
3 control.agentsteer2 = steer;
4 control.agentbrake2 = brake;
5 /.../
6 // When the Agent reaches the end of a road segment
7 if (other.gameObject == FinishTrigger)
8 {
9     //episode completed
10    AddReward(distance / laptime);
11    pathGen.GenNew();
12    tor.EndEpP();
13    EndEpisode();
14 }
```

4.8 Takeover Request System

The TOR manager stores a reference to the agent and through this can use the functions to control when a TOR is issued. It knows when an episode (each time the car starts at the beginning of a road segment until it reaches the end or goes off the road) begins and ends. At the beginning of the episode it decides some random point to issue the TOR, it is limited so that it will not occur too early or too late along the road segment. When the TOR is issued, a sound is played and the user interface is notified to change. In addition to this functionality it also captures all the data used for testing the model. It stores the position of the TOR, the reaction time of the driver, the reaction method, the design of the road segment, the average speed and the outcome of the episode (whether the car reached the end or not). It writes all of this data to a Comma-Separated Values (CSV) file, an example of this with the outcome value can be seen below.

```
1 private void AddOutcomeToCSV(string outcome)
2 {
3     string filePath = "Assets/Reactions/Outcome.csv";
4     string delimiter = ",";
5     // Check if the file exists
6     bool fileExists = File.Exists(filePath);
7     // Open or create the file
8     StreamWriter writer = new StreamWriter(filePath, true);
```

```
9       // If the file doesn't exist, write the header
10      if (!fileExists)
11      {
12          writer.WriteLine("Outcome");
13      }
14      writer.WriteLine(outcome); // Write the outcome value
15      // Close the file
16      writer.Close();
17  }
```

4.9 Foliage Spawner

The foliage spawner is used instantiate trees in the environment to allow for improved immersion and to better judge speed. It starts by getting random positions in the environment and checking if the position hits the road or not.

```
1  private Vector3 GetRandomPosition()
2  {
3      Vector3 randomPosition;
4      bool isColliding;
5      do
6      {
7          float randomX = Random.Range(minBounds.x, maxBounds.x);
8          float randomY = 0.0f;
9          float randomZ = Random.Range(minBounds.z, maxBounds.z);
10         randomPosition = new Vector3(randomX, randomY, randomZ);
11         RaycastHit hit;
12         isColliding = meshCollider.Raycast(new Ray(randomPosition + Vector3.up * 10f
       , Vector3.down), out hit, 20f);
13     }
14     while (isColliding);
15     return randomPosition;
16 }
```

Using the generated random positions that do not collide with the road, it then proceeds to spawn the trees in the environment.

```
1  private void SpawnObjects()
2  {
3      for (int i = 0; i < numberOfObjects; i++)
4      {
5          Vector3 spawnPosition = GetRandomPosition();
6          Instantiate(objectPrefab, spawnPosition, Quaternion.identity);
7      }
8  }
```

This is updated every episode so that when a new road is generated, the trees that intersect the road are deleted from memory and are replaced randomly somewhere else in the environment.

4.10 User Interface

The user interface consists of two parts, the speedometer and the TOR state. The speedometer is updated in real time based on the speed of the car and is displayed in

kilometers per hour. The TOR state updates based on whether or not the driver must take control of the car. The camera was chosen to be placed just in front of windscreen on the driver side. Since the experts were using a steering wheel in the real world the camera was placed in such a way to make it seem like an extension of their surroundings.

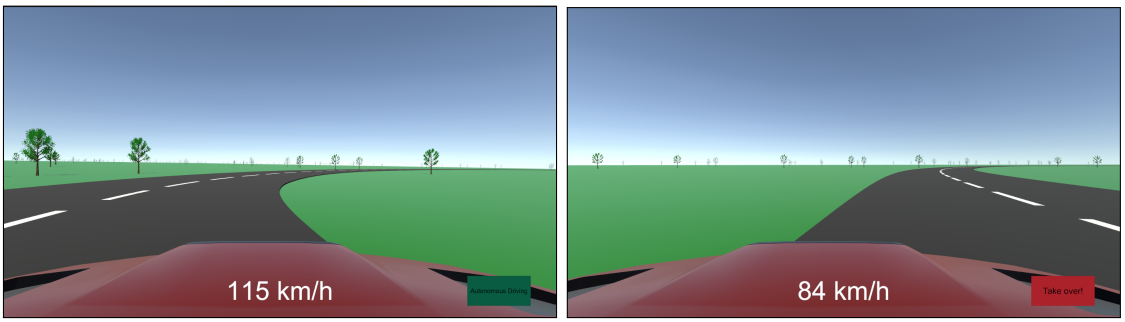


Figure 4.3: The UI states.

Chapter 5

Testing & Evaluation

This chapter details the testing process, how human demonstrations were recorded, the methods used to evaluate the AI models and the results from the experiment.

5.1 Testing Process

Before any data is recorded the subjects were given a demographics survey (which can be seen in Appendix A) to complete in order to attain details about them. Next, they were giving whatever time they needed (up to a total of ten minutes) to become comfortable with the driving system. During this process, it was explained to the contributors what they would have to do during the data collection phase.

Participants were given a task to complete, the task consisted of reading a particular Wikipedia article with the aim to be able to answer questions about said article after the testing was complete. This task is designed to distract the drivers before the TOR is issued. Explicit instructions were given to maintain visual focus on the screen without diverting attention to the road ahead until prompted by the TOR.

At a randomly chosen moment during the reading task, the TOR was activated, requiring participants to immediately assume control of the vehicle. Participants then drove the vehicle either to the end of the designated road segment or, in case of loss of control, off the road. Ray-casts were used to detect where the road is at all times in relation to the vehicle, this data was recorded and stored to be used for the learning process. Other data recorded for the learning process includes the steering angle, accelerator usage and brake usage. In addition to this, the reaction time of the participant is recorded. This reaction time is the time from when the TOR is issued until either the accelerator, brake or steering wheel has been used by the participant. The style of the road segment and the position where the TOR was issued are recorded along with the outcome of the scenario.

The test took approximately 40 minutes in total to complete and generally the individuals had to partake in between 45 to 75 different TOR situations, the majority of participants were given 75 situations.

5.2 Evaluation

5.2.1 Paired Samples T-Test

This test is employed when working with paired data points, often collected under different conditions or at different points in time. In this analysis they were collected under different conditions, the human driver and the AI model. The method evaluates whether the observed differences between the pairs are likely to have occurred by chance or if they represent a true underlying difference in the populations from which the data were derived. Essentially, demonstrating if there is a significant difference or not.

The paired samples t-test is particularly useful in scenarios where each data point in one group is directly linked or matched to a corresponding data point in the other group. This pairing helps control for individual variability and reduces the influence of confounding factors. The data points that are linked include the road segment and position where the TOR was issued.

The test involves calculating the mean and standard deviation of the paired differences between the two groups. It then compares this observed difference to what would be expected under the null hypothesis of no difference. If the calculated t-statistic (a measure of the difference standardized by the standard error) is significantly different from zero, and assuming other assumptions of the test are met, it suggests that a statistically significant difference exists between the two groups.

In the context of evaluating AI models, this test can help determine whether there is a significant improvement or difference in outcomes between different experimental conditions.

5.2.2 Binomial Proportions

Some of the data recorded for testing was discrete (outcomes). To compare this type of data a t-test could not be used as it functions on continuous data only. To find the standard error between two sample proportions the following formula can be used:

$$\frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}}$$

Where, for example, $\hat{p}_1$ is the percentage of an AI model's sample outcomes that are positive and $\hat{p}_2$ is the percentage of a human's sample outcomes that are positive. n_1 and n_2 are the sizes of the AI samples and human samples, in every case this is the same for both AI and human as they are tested on the same amount of road segments. In order for the result from this formula to be used it needs to be proven that the denominator is equivalent to the population. The proof for this can be seen in appendix B. With this proof, it is possible to determine the correct critical values for each of the AI models as some of them have different sample sizes. The results from this test are discussed later in this chapter.

5.2.3 Leave One Out Cross Validation

To ensure reliable data points, the Leave-One-Out Cross-Validation (LOOCV) technique was employed as outlined in the Related Work chapter. An AI model was developed

using data from eleven of the twelve participants and then the resultant model was compared with the twelfth subject. This procedure was repeated for each participant, resulting in a total of twelve distinct AI models.

This method helped address potential limitations stemming from the relatively small sample size of twelve participants. By systematically testing each participant's data, a more comprehensive understanding of the AI models' performance was gained, enhancing the validity of the findings within the constraints of the participant pool.

5.3 Participant Demographics

A total of twelve participants took part in the study with an age range from 21 to 30 and a mean age of 26.

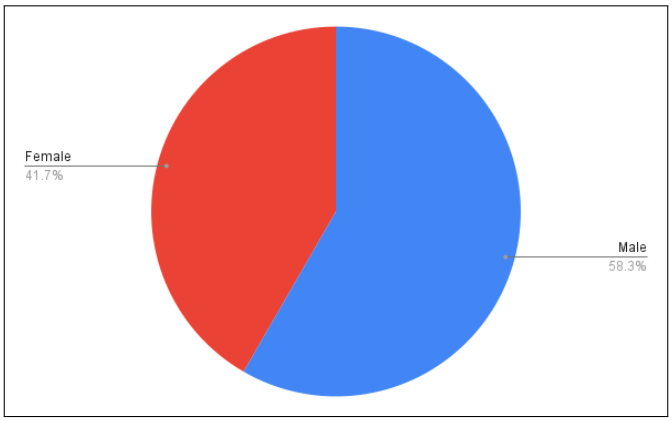


Figure 5.1: The split of Gender.

Ten out of the twelve individuals who partook in the study possessed valid drivers' licenses and all participants had some level of previous experience with driving.

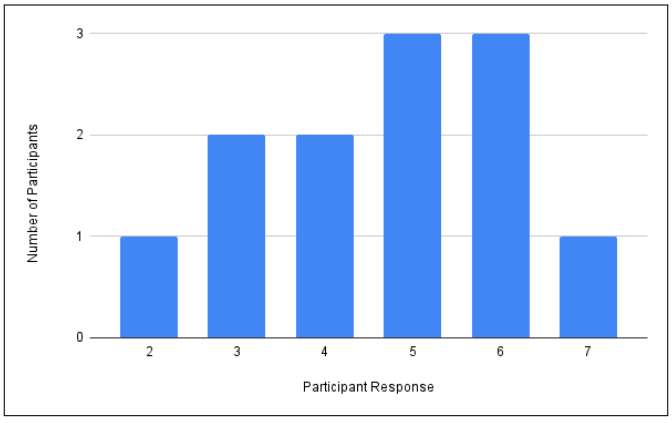


Figure 5.2: Responses to driving experience.

When asked to rate their experience on a seven-point Likert scale (eg. strongly

disagree to strongly agree), the mean score was 4.6, suggesting a slightly above-average level of overall experience among the contributors. However, the question results can be seen to be approximating a normal distribution with a positively skewed bias in figure 5.2. This could reflect the youth of the participants and their overconfidence in their ability to drive. Plainly put, young people might tend to think they drive better than older generations.

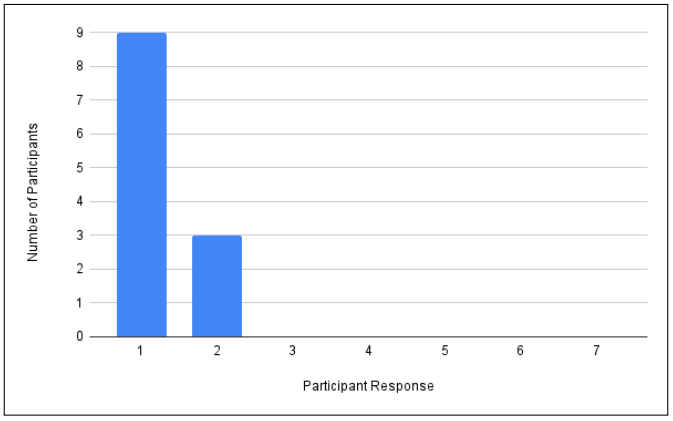


Figure 5.3: Responses to experience with advanced driving assists.

A quarter of the participants reported possessing very limited familiarity with driving aids like lane switching and adaptive cruise control.

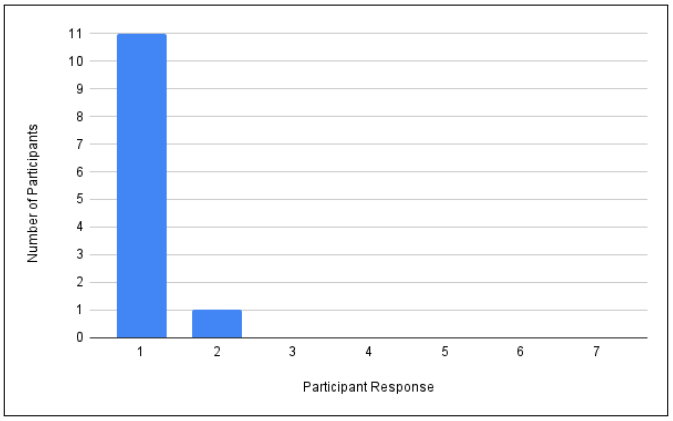


Figure 5.4: Responses to experience with automated driving.

An even smaller subset (8%) acknowledged minimal exposure to automated driving systems, such as the *Tesla Autopilot*. The majority of participants indicated a complete lack of experience with advanced driving aids or automated driving technologies. These results reflect the fact that automated vehicles are not widely available, particularly to the participants selected for the study.

5.4 Results from the Learning Process

From the learning process two main areas were looked at, the rewards over time (measured in steps) and the policy loss over time. In figure 5.5 we can see that the extrinsic reward increases over time while in figure 5.6 it is shown that the reward for GAIL also increases at the same rate over time. This is a positive result for the AI models as it shows that as the AI improves over time the reward keeps increasing, it is additionally positive to see that the results are similar across all of the models.

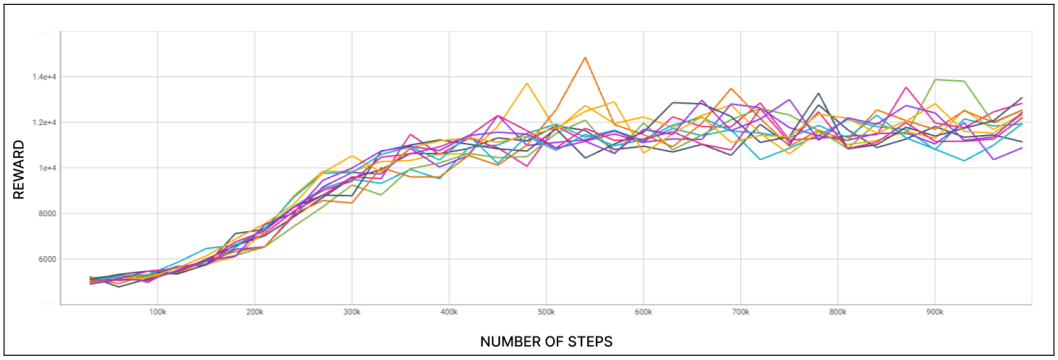


Figure 5.5: Extrinsic reward, each colour represents one of the AI models.

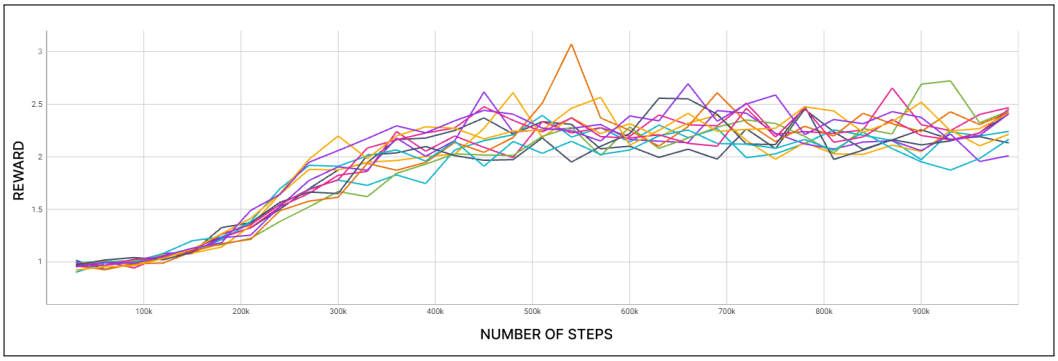


Figure 5.6: GAIL reward, each colour represents one of the AI models.

The policy loss over time can be seen to decrease in figure 5.7. This is another positive result as by minimizing the policy loss, the AI model is learning a policy that imitates the humans' actions and performs well in complex environments.

5.5 Results from AI comparison with Humans

From the participants a total of 900 TOR were issued. Due to the randomized nature of the road generation, some road segments were repeated multiple times and others appeared only once during testing. Each comparison between the AI model and human

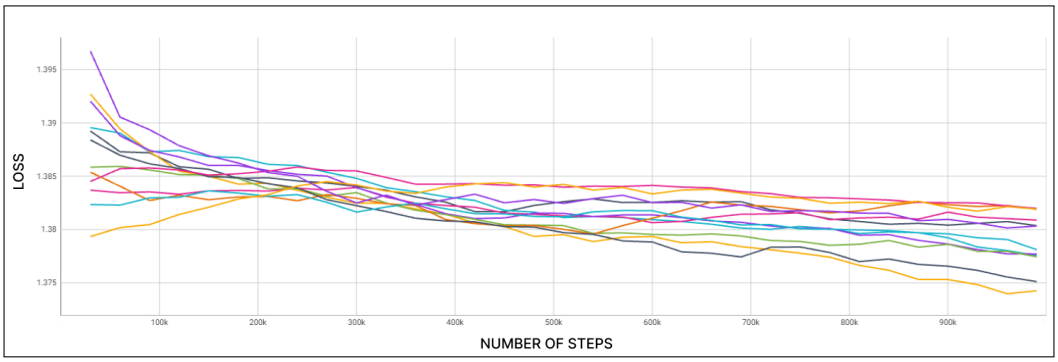


Figure 5.7: Policy loss, each colour represents one of the AI models.

driver was conducted under the same circumstances (identical road segment and same TOR position).

5.5.1 Outcomes

The outcomes of each TOR (table 5.1) was one element of data that was used to compare the AI models to human drivers. The outcome had two possible options, positive or negative. A positive outcome is considered when the car has reached the end of the road segment after the TOR without straying off the road onto the grass. A negative outcome occurs when the car has strayed off the road after a TOR has been issued. All of these outcomes where tested using the LOOCV method on the exact same road segments with the TOR occurring at the same position on the road.

A null hypothesis of the AI models not being different from the human drivers is considered. On average the standard error of the sample proportion difference was calculated as 0.85354167. The standard error is distributed similarly to a standard normal distribution with a critical value of 1.96. Since the standard error is less than the critical value overall the null hypothesis is not rejected and the result shows that there is no statistically significant difference between the outcomes of the AI models and the outcomes of the human drivers.

The calculated standard error and critical value for each comparison between the AI model and humans can be seen in table 5.2. The standard error was calculated as stated before and the critical values were chosen from the T-tables at [39] based on sample size. It is worth noting that the critical values were chosen based on a significance level of 0.05 (95%) and it was determined to be two tailed since the difference between the AI and human could be negative or positive.

Out of the twelve participants, one of the drivers provided a rejection of the null hypothesis (driver 2). The standard error in this case was calculated as 3.4, thus suggesting a statistically significant difference. The probable cause of this was due to the driver being an outlier when compared with the other participants' results.

Participant	AI Positive	AI Negative	Human Positive	Human Negative
1	34	17	34	17
2	29	16	14	31
3	29	19	28	20
4	37	18	30	25
5	41	34	40	35
6	47	28	41	34
7	45	30	40	35
8	44	31	40	35
9	49	26	48	33
10	58	17	43	32
11	48	27	45	30
12	44	31	40	35

Table 5.1: AI and human outcomes after a TOR was issued, in participant order.

Participant	Standard Error	Critical Value
1	0.0	1.9845
2	3.4	1.9878
3	0.1993	1.986
4	1.209	1.9827
5	0.1647	1.9766
6	1.209	1.9766
7	0.8298	1.9766
8	0.5867	1.9766
9	0.6533	1.9766
10	0.773	1.9766
11	0.64	1.9766
12	0.5867	1.9766

Table 5.2: Calculated standard error and critical value for outcomes, in participant order.

The driver had a positive outcome 30% of the time in comparison to the average positive outcome rate of 55.64% from the other human drivers. The participant is over 2 standard deviations away from the mean (36.9167) and thus is considered as a potential outlier. This is seen in figure 5.8, where driver 2 can be seen in red. The line in red is the linear trendline for all the outcomes while the line in blue is the linear trendline for all the outcomes except for driver 2.

5.5.2 Average Speed

The average speed used over each road segment was additionally used to compare the AI with the human drivers. Similarly to the outcomes it used LOOCV and was compared in the same way, this time using the T-test. A confidence interval of 0.05 (95%) was used to determine if the data was statistically significantly different. The null hypothesis

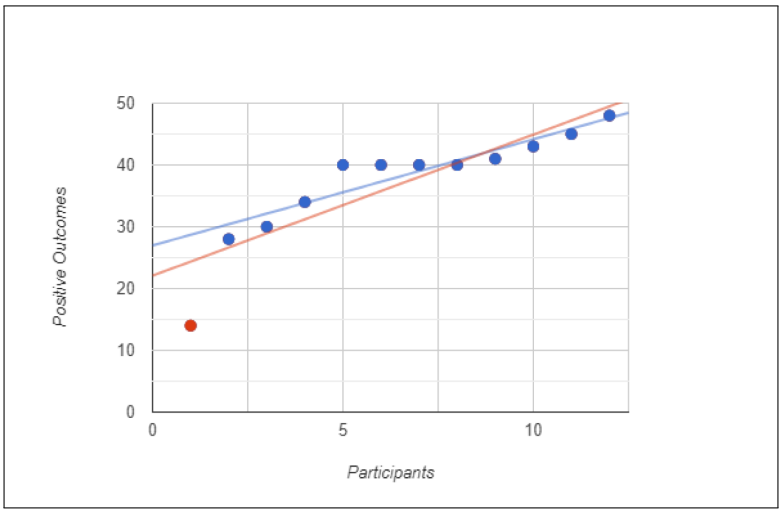


Figure 5.8: Human drivers positive outcomes, from lowest to highest.

Participant	P value
1	0.929
2	<0.001
3	0.945
4	0.855
5	0.227
6	0.743
7	0.521
8	0.729
9	0.196
10	0.320
11	0.295
12	0.497

Table 5.3: T-test results for average speed comparison, in participant order.

used was that the average speed of the human drivers would be equal to the average speed of the AI models.

All of the data was unable to reject the null hypothesis except for the same outlier as before in driver 2. Despite driver 2 having the highest number of negative outcomes, they also drove the slowest out of all the human drivers. They had a mean average speed of 89.4km/h compared to 113.7km/h from the rest of the group. Driver 2 again was over 2 standard deviations away from the mean of the whole group (111km/h). This difference is illustrated in figure 5.9.

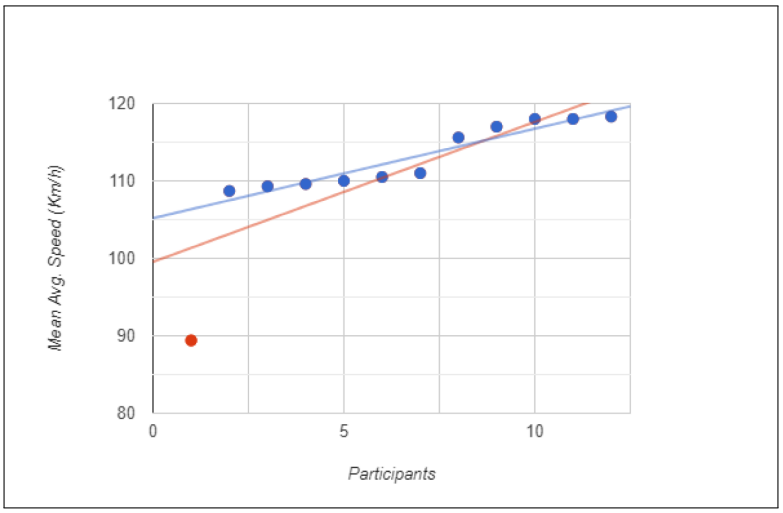


Figure 5.9: Human drivers mean average speed, from lowest to highest.

5.5.3 Reactions

The reaction time was compared in the same way as the average speed. The reaction method was always the accelerator for the AI models. From personal observations the AI models would generally tweak and change the value of the accelerator more often but to a smaller degree than the human drivers. This behavior would then always trigger the accelerator for the reaction method. The null hypothesis used for the paired samples t-test was that the human drivers' reactions would be equal to the AI models' reactions.

In every case except two the p value was higher than the confidence value of 0.05 and thus was deemed not a statistically significant enough difference to reject the null hypothesis. As was the case with the outcome and average speed results, the p value when comparing the AI model's reactions to driver 2 was small and therefore rejected the null hypothesis. Overall, the AI model matched better with drivers who had quicker reaction times on average than those who reacted slower. Most likely this is due to the AI's reaction being standardized to a certain extent. Due to how ML agents work in *Unity* a decision from the AI is requested every few frames (in this case every 5 frames), the decision to do nothing would be overridden if intervention was necessary. This fact caused the AI to react quickly after a TOR. It is also worth noting that for the same reason, the reaction times would be frame dependant so the AI's reactions could differ depending on the hardware used to run it.

Comparing the reaction times of the AIs and human drivers to related work proved to be a challenge as much of the previous work done has used a lead-in time. This is a buffer time between the notification of the TOR and when the driver actually has to take over control of the car. Despite this fact it is still possible to compare to related work by Eriksson and Stanton [9] and Huang and Pitts [12] who both recorded reaction times after participants were reading. Both of these works included a lead-in time, however, they recorded the reactions from the notification time so the results are still relevant to the work in this thesis. The mean reaction times for both of these works were 3.2

Participant	P value
1	0.069
2	0.006
3	0.791
4	0.866
5	0.468
6	0.305
7	0.223
8	0.002
9	0.077
10	0.028
11	0.208
12	0.905

Table 5.4: T-test results for reaction times comparison, in participant order.

and 3.8 seconds respectively. The reaction times from Huang and Pitts also take into account the reaction method so it is further relevant to the work in this thesis. It can be seen that these reactions differ greatly from the mean reaction time of human drivers in this thesis of 1.6025 and the AI mean reaction time of 1.21 seconds. One possible cause of this could be from the fact there was no lead-in time so the participants panicked and reacted quicker than if they knew they had a lead-in time.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

The project was set up to see if a potentially human-like AI driving model could be created using imitation learning techniques, how this model can be evaluated and how it compares to human drivers. Additionally, a goal was to determine if the model or models built using the same framework could be used in future to optimize the TOR process. From the results gained, it can be seen that it is possible to create a human-like driving model using the machine learning techniques discussed in this thesis. The findings show that the AI models created, as part of this project, do not act in a statistically significant way that is different to the human drivers. One of the human drivers did prove to be significantly different from the AI model tested against them. This was not deemed to be fatal to the project's conclusions as they are a statistical outlier when compared with the other human drivers.

A novel approach to testing AI models, that builds upon previous work, was presented in the thesis. It was important to ensure that the testing and evaluation of the project was statistically relevant and could be relied upon consistently. In particular the extensive work committed to ensuring the statistical difference between the AI outcomes and human outcomes was accurate and defensible, proved to be invaluable when evaluating the performance of the AI models. Similarly, comparing the average speed of each human driver with the AI models backed up the results found in the outcome section.

One point to note is that the results gained from the reactions are open to variability due to the way that *Unity ML Agents* work and computer hardware used. This shortcoming can be overcome by utilizing the same hardware for recording data and testing the AI models.

Comparisons of the results achieved in this thesis were made with related work. This was limited to the reaction times due to the variability the road network has on the results from outcomes and average speed. It was seen that the reactions time in this thesis were faster on average than those in related work and it is possible that perhaps a greater distraction was needed before the TOR was issued.

It is clear from the work achieved in this project that the AI driver model or a model built using the same framework could be used to optimize TOR situations. The AI

models developed during this thesis proved to achieve human-like results on a variety of road types. These models could be used to test additional TOR situations and determine the best scenarios to issue a TOR.

From the findings it is also clear that multiple models are needed since there can be significant variance in driving ability of different human drivers.

Ultimately, the project's objectives were effectively met.

Various AI driving models were developed and underwent comprehensive evaluation against their human counterparts, with the results revealing a lack of statistically significant divergence. The established evaluation methodologies, designed to ensure accuracy and consistency, further reinforce the robustness of the findings.

Finally, the work presented in this thesis substantiates the potential of the AI models developed to enhance TOR optimization.

6.2 Future Work

While significant progress has been made, there are several avenues for future research and development that can further enhance the efficacy and applicability of AI-driven driving behavior.

It is recommended that additional human demonstrators are used in any continuing work. The study was effectively limited by the number of participants due to the practical constraints of time and resources available. Furthermore, due to the nature of imitation learning, the AI models will benefit from extra data points to train on and test against.

Further work can be done to use the AI models or AI models made with the techniques explored in this thesis to optimize TOR. The models can be tested across millions of road segments for millions of hours to enhance the TOR process and gain new insight into the challenges faced by TOR systems.

It would be important to validate the findings from this work and others with real world trials that replicate the scenarios. Conducting controlled experiments and field trials can provide insights into the practical feasibility of using AI models to optimize TOR. Perhaps the techniques used in this thesis could be replicated in the real world in a safe open space to negate any issues with the simulation aspect of the environment.

In conclusion, this thesis has provided extensive insight into the area of replicating human behavior after a TOR is issued and in how to evaluate AI models made to replicate human behavior. The areas for future exploration highlighted above present exciting prospects for refining and expanding upon the research in this paper and in improving autonomous driving.

Appendix A

Demographics Survey

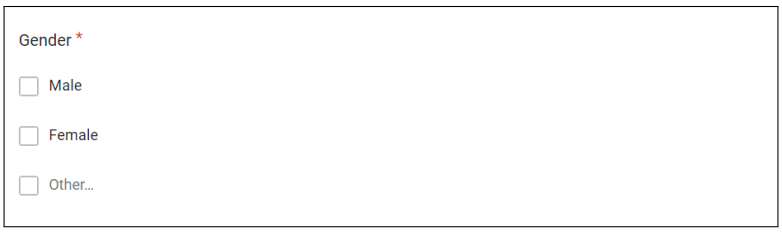


Figure A.1: Gender.

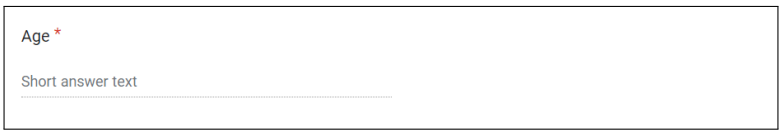


Figure A.2: Age.

Figure A.3: Driver's License.

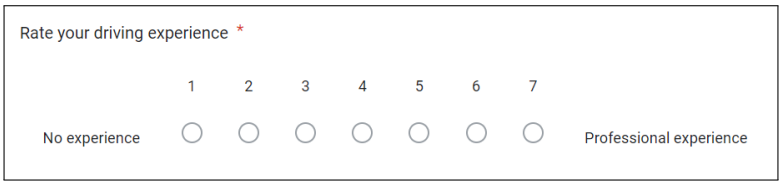


Figure A.4: Driving Experience.

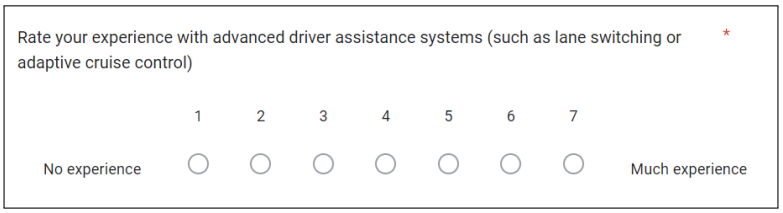


Figure A.5: Experience with advanced driving assists.

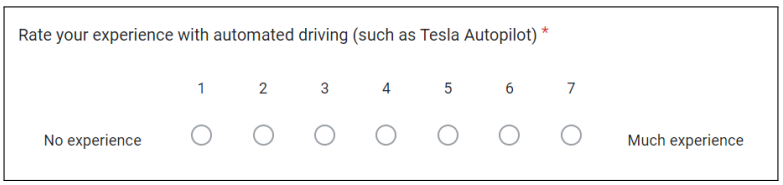


Figure A.6: Experience with automated driving.

Appendix B

Standard Error Proof

To prove that the equation for the population sample:

$$\frac{(n_1p_1 + n_2p_2)(1 - n_1p_1 - n_2p_2)}{n_1 + n_2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)$$

is equivalent to the denominator of the standard error between two sample proportions:

$$\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}$$

Step 1: Distribute the first term inside the first set of parentheses:

$$\frac{n_1p_1 + n_2p_2 - n_1p_1^2 - n_2p_2^2}{n_1 + n_2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)$$

Step 2: Distribute the second set of parentheses:

$$\frac{n_1p_1 + n_2p_2 - n_1p_1^2 - n_2p_2^2}{n_1n_2 + n_1^2 + n_2^2 + n_1n_2}$$

Step 3: Combine terms in the numerator:

$$\frac{n_1p_1 + n_2p_2 - n_1p_1^2 - n_2p_2^2}{(n_1 + n_2)(n_1 + n_2)}$$

Step 4: Factor out n_1 and n_2 from the numerator:

$$\frac{n_1(p_1 - p_1^2) + n_2(p_2 - p_2^2)}{(n_1 + n_2)^2}$$

Step 5: Factor out $p_1(1-p_1)$ and $p_2(1-p_2)$ from the terms:

$$\frac{p_1(1-p_1)n_1 + p_2(1-p_2)n_2}{(n_1 + n_2)^2}$$

Step 6: Separate the fractions and simplify:

$$\frac{p_1(1-p_1)}{n_1 + n_2} + \frac{p_2(1-p_2)}{n_1 + n_2}$$

Step 7: Factor out $1/(n_1 + n_2)$:

$$\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}$$

This result is equivalent to the denominator of the standard error between two sample proportions equation:

$$\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}$$

Appendix C

Glossary

AI Artificial Intelligence

TOR Takeover Request

BC Behavioral Cloning

GAIL Generative Adversarial Imitation Learning

PPO Proximal Policy Optimization

Unity Game engine that utilizes the C# programming language

Unity ML Agents A machine learning toolkit that allows for the incorporation of various machine learning algorithms within the Unity game engine

MSE Mean Squared Error

LOOCV Leave One Out Cross Validation, a testing methodology

T-test Student's test, A statistical hypothesis test used to determine if there is a significant difference between the means of two groups

L-system Lindenmayer system, a formal grammar used to model the growth and development of complex patterns.

FDD Feature Driven Development

SDK Software Development Kit

UV Additional set of coordinates similar to xyz, traditionally used for textures

Bézier curve A Bézier curve is a mathematical representation used in computer graphics and modeling to describe smooth and curved shapes through control points' manipulation

Logitech G29 A steering wheel and pedal system developed by *Logitech* that works with simulation software and games.

RAVAI Rudimentary Autonomous Vehicle Artificial Intelligence, a relatively simple AI driving model developed for this project

Hyperparamters Parameters in machine learning algorithms that are set before the learning process and influence the model's behavior and performance

GAE Generalized Advantage Estimation, a reinforcement learning technique that computes a value function advantage by combining immediate rewards with predicted future rewards, helping to improve policy optimization

CSV Comma-Separated Values, a simple file format used to store tabular data, where each line represents a row, and values within each row are separated by commas

Ray-Casts A technique used in computer graphics and physics simulations to determine intersections and collisions by tracing a line (ray) from a starting point in a specific direction

SPDC Sample Proportions Difference Calculation

References

Literature

[1] Laith Alzubaidi et al. "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions". *Journal of Big Data* 8.1 (Mar. 31, 2021), p. 53. DOI: 10.1186/s40537-021-00444-8. (Visited on 09/02/2023) (cit. on p. 2).

[2] Birk Knut Astrup et al. "Car steering in racing games: similar performance with joysticks and keyboards". In: *Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments*. PETRA '19: The 12th PErvasive Technologies Related to Assistive Environments Conference. Rhodes Greece: ACM, June 5, 2019, pp. 309–310. DOI: 10.1145/3316782.3321547. (Visited on 08/09/2023) (cit. on p. 14).

[3] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst". In: *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*. Ed. by Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson. 2019. DOI: 10.15607/RSS.2019.XV.031 (cit. on p. 7).

[4] S.R. Burns, R.T. O'Brien, and J.A. Piepmeier. "Steering controller design using scale-model vehicles". In: *Proceedings of the Thirty-Fourth Southeastern Symposium on System Theory (Cat. No.02EX540)*. Thirty-Fourth Southeastern Symposium on System Theory. Huntsville, AL, USA: IEEE, 2002, pp. 476–478. DOI: 10.1109/SSST.2002.1027092. (Visited on 08/09/2023) (cit. on p. 14).

[5] Felipe Codevilla et al. "End-to-end driving via conditional imitation learning". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4693–4700 (cit. on p. 7).

[6] Rémi Cura, Julien Perret, and Nicolas Paparoditis. *StreetGen : In base city scale procedural generation of streets: road network, road surface and street objects*. Jan. 17, 2018. arXiv: 1801.05741 [cs]. URL: http://arxiv.org/abs/1801.05741 (visited on 08/07/2023) (cit. on p. 10).

[7] Mike Daily et al. "Self-Driving Cars". *Computer* 50.12 (Dec. 2017), pp. 18–23. DOI: 10.1109/MC.2017.4451204. (Visited on 08/22/2023) (cit. on p. 6).

[8] Esteban Egea-Lopez et al. "Vehicular Networks Simulation With Realistic Physics". *IEEE Access* 7 (2019), pp. 44021–44036. DOI: 10.1109/ACCESS.2019.2908651. (Visited on 08/09/2023) (cit. on p. 14).

[9] Alexander Eriksson and Neville A. Stanton. "Takeover Time in Highly Automated Vehicles: Noncritical Transitions to and From Manual Control". *Human Factors: The Journal of the Human Factors and Ergonomics Society* 59.4 (June 2017), pp. 689–705. DOI: 10.1177/0018720816685832. (Visited on 08/22/2023) (cit. on pp. 6, 39).

[10] Jonas Freiknecht and Wolfgang Effelsberg. "A Survey on the Procedural Generation of Virtual Worlds". *Multimodal Technologies and Interaction* 1.4 (Oct. 30, 2017), p. 27. DOI: 10.3390/mti1040027. (Visited on 08/09/2023) (cit. on p. 10).

[11] E. Galin et al. "Procedural Generation of Roads". *Computer Graphics Forum* 29.2 (May 2010), pp. 429–438. DOI: 10.1111/j.1467-8659.2009.01612.x. (Visited on 08/07/2023) (cit. on p. 10).

[12] Gaojian Huang and Brandon J. Pitts. "Takeover requests for automated driving: The effects of signal direction, lead time, and modality on takeover performance". *Accident Analysis & Prevention* 165 (Feb. 2022), p. 106534. DOI: 10.1016/j.aap.2021.106534. (Visited on 09/02/2023) (cit. on p. 39).

[13] Institute of Electrical and Electronics Engineers, ed. *2007 Symposium on Computational Intelligence and Games: Honolulu, HI, 1-5 April 2007*. Piscataway, N.J: IEEE, 2007. 388 pp. (cit. on p. 14).

[14] Arthur Juliani et al. "Unity: A general platform for intelligent agents". *arXiv preprint arXiv:1809.02627* (2020) (cit. on p. 19).

[15] Won Kim et al. "Take-Over Requests after Waking in Autonomous Vehicles". *Applied Sciences* 12.3 (Jan. 28, 2022), p. 1438. DOI: 10.3390/app12031438. (Visited on 08/22/2023) (cit. on p. 6).

[16] Jakob Kuen, Clemens Schartmüller, and Philipp Wintersberger. "The TOR Agent:Optimizing Driver Take-Over with Reinforcement Learning". In: *13th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. AutomotiveUI '21 Adjunct. Leeds, United Kingdom: Association for Computing Machinery, 2021, pp. 47–52. DOI: 10.1145/3473682.3480262 (cit. on p. 6).

[17] Jonghak Lee et al. "Model Evaluation for Forecasting Traffic Accident Severity in Rainy Seasons Using Machine Learning Algorithms: Seoul City Study". *Applied Sciences* 10.1 (Dec. 23, 2019), p. 129. DOI: 10.3390/app10010129. (Visited on 08/07/2023) (cit. on p. 8).

[18] Radomír Měch and Przemyslaw Prusinkiewicz. "Generating subdivision curves with L-systems on a GPU". In: *ACM SIGGRAPH 2003 Sketches & Applications*. SIGGRAPH03: Special Interest Group on Computer Graphics and Interactive Techniques. San Diego California: ACM, July 27, 2003, pp. 1–1. DOI: 10.1145/965400.965520. (Visited on 08/07/2023) (cit. on p. 10).

[19] Vivien Melcher et al. "Take-Over Requests for Automated Driving". *Procedia Manufacturing* 3 (2015), pp. 2867–2873. DOI: 10.1016/j.promfg.2015.07.788. (Visited on 08/22/2023) (cit. on p. 6).

[20] Khan Muhammad et al. “Deep Learning for Safe Autonomous Driving: Current Challenges and Future Directions”. *IEEE Transactions on Intelligent Transportation Systems* 22.7 (July 2021), pp. 4316–4336. DOI: 10.1109/TITS.2020.3032227. (Visited on 08/22/2023) (cit. on p. 5).

[21] Koki Muto et al. “Driving Behavior during Takeover Request of Autonomous Vehicle: Effect of Driver Postures”. *Behavioral Sciences* 12.11 (Oct. 28, 2022), p. 417. DOI: 10.3390/bs12110417. (Visited on 08/22/2023) (cit. on p. 6).

[22] Seyed Soroush Pakzad, Naeim Roshan, and Mansour Ghalehnovi. “Comparison of various machine learning algorithms used for compressive strength prediction of steel fiber-reinforced concrete”. *Scientific Reports* 13.1 (Mar. 4, 2023), p. 3646. DOI: 10.1038/s41598-023-30606-y. (Visited on 08/07/2023) (cit. on p. 8).

[23] Yoav I. H. Parish and Pascal Müller. “Procedural modeling of cities”. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. SIGGRAPH01: The 28th International Conference on Computer Graphics and Interactive Techniques. ACM, Aug. 2001, pp. 301–308. DOI: 10.1145/383259.383292. (Visited on 08/07/2023) (cit. on p. 10).

[24] A G Priya Varshini et al. “Comparative analysis of Machine learning and Deep learning algorithms for Software Effort Estimation”. *Journal of Physics: Conference Series* 1767.1 (Feb. 1, 2021), p. 012019. DOI: 10.1088/1742-6596/1767/1/012019. (Visited on 08/07/2023) (cit. on p. 8).

[25] Jonas Radlmayr et al. “How Traffic Situations and Non-Driving Related Tasks Affect the Take-Over Quality in Highly Automated Driving”. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 58.1 (Sept. 2014), pp. 2063–2067. DOI: 10.1177/1541931214581434. (Visited on 08/22/2023) (cit. on p. 6).

[26] Adrian Remonda, Eduardo Veas, and Granit Luzhnica. “Comparing driving behavior of humans and autonomous driving in a professional racing simulator”. *PLOS ONE* 16.2 (Feb. 3, 2021). Ed. by Chen Lv, e0245320. DOI: 10.1371/journal.pone.0245320. (Visited on 01/25/2023) (cit. on p. 7).

[27] Amanda Ross and Victor L. Willson. “Paired Samples T-Test”. In: *Basic and Advanced Statistical Tests*. Rotterdam: SensePublishers, 2017, pp. 17–19. DOI: 10.1007/978-94-6351-086-8_4. (Visited on 08/07/2023) (cit. on p. 8).

[28] Mario Rubak. “Imitation Learning with the Unity Machine Learning Agents Toolkit”. Masters. University of Applied Sciences FH Campus Wien, May 6, 2021. 76 pp. URL: https://pub.fh-campuswien.ac.at/obvfcwhsacc/content/titleinfo/6294692/full.pdf (cit. on p. 19).

[29] Kevin Joel Salubre and Dan Nathan-Roberts. “Takeover Request Design in Automated Driving: A Systematic Review”. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 65.1 (Sept. 2021), pp. 868–872. DOI: 10.1177/1071181321651296. (Visited on 08/22/2023) (cit. on p. 6).

[30] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on p. 7).

[31] Edward Teng and Rafael Bidarra. "A semantic approach to patch-based procedural generation of urban road networks". In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*. FDG'17: International Conference on the Foundations of Digital Games 2017. Hyannis Massachusetts: ACM, Aug. 14, 2017, pp. 1–10. DOI: 10.1145/3102071.3110569. (Visited on 08/07/2023) (cit. on p. 10).

[32] Jorge Vargas et al. "An Overview of Autonomous Vehicles Sensors and Their Vulnerability to Weather Conditions". *Sensors* 21.16 (Aug. 10, 2021), p. 5397. DOI: 10.3390/s21165397. (Visited on 09/02/2023) (cit. on p. 1).

[33] Peter R. Wurman et al. "Outracing champion Gran Turismo drivers with deep reinforcement learning". *Nature* 602.7896 (Feb. 2022), pp. 223–228. DOI: 10.1038/s41586-021-04357-7. (Visited on 08/09/2023) (cit. on p. 7).

[34] Chi-Wen Yang et al. "Unity 3D production and environmental perception vehicle simulation platform". In: *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*. 2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE). Tainan, Taiwan: IEEE, Nov. 2016, pp. 452–455. DOI: 10.1109/ICAMSE.2016.7840349. (Visited on 08/09/2023) (cit. on p. 14).

Software

[35] *Neo Cab*. Version 1.0. May 10, 2019. URL: https://neocabgame.com/ (cit. on p. 10).

[36] Mehdi Rabiee. *Realistic Car Kit*. Version 2.3. Mar. 2, 2016. URL: https://assetstore.unity.com/packages/tools/physics/realistic-car-kit-18421#description (cit. on pp. 14, 22).

[37] *Unity*. Version 2020.3.4f1. Apr. 13, 2021 (cit. on p. 19).

Online sources

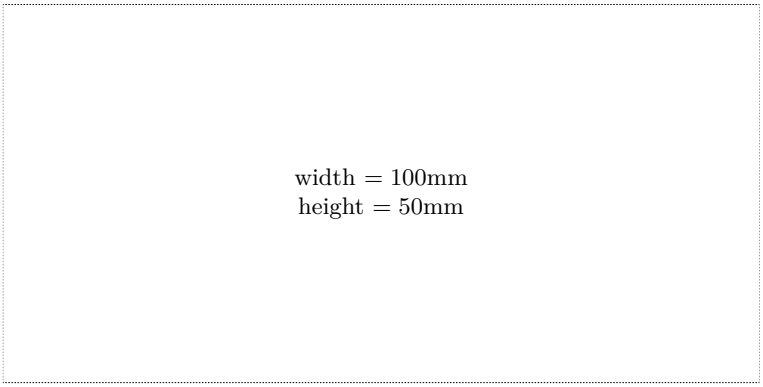
[38] Nancy Grugle. *Human Factors in Autonomous Vehicles*. URL: https://www.americanbar.org/groups/tort_trial_insurance_practice/publications/tortsource/2019/fall/human-factors-autonomous-vehicles/ (visited on 08/22/2023) (cit. on p. 5).

[39] *T Critical Value Calculator (t Table Calculator)*. URL: https://www.allmath.com/t-critical-value.php (visited on 08/21/2023) (cit. on p. 36).

[40] *The 6 Levels of Vehicle Autonomy Explained | Synopsys Automotive*. URL: https://www.synopsys.com/automotive/autonomous-driving-levels.html (visited on 08/22/2023) (cit. on p. 5).

Check Final Print Size

— Check final print size! —



— Remove this page after printing! —