AU-CARL: AUTONOMOUS VEHICULAR NAVIGATION WITH AUGMENTED REALITY AND REINFORCED LEARNING

A PROJECT REPORT

Submitted by

ADHIRAJ BHAGAWATI [Reg No: RA1811003010023]
ROHIT RAMESH [Reg No: RA1811003010055]

Under the guidance of
Dr. R.I. Minu
(Associate Professor, Department of Computing Technologies)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that 18CSP109L project report titled "**AU-CARL: AUTONOMOUS VEHICULAR NAVIGATION WITH AUGMENTED REALITY AND REINFORCED LEARNING** " is the bonafide work of " **ADHIRAJ BHAGAWATI [Reg No: RA1811003010023]** and **ROHIT RAMESH [Reg No: RA1811003010055]**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

GUIDE
Dr. R.I. Minu
Associate Professor
Dept. of Computing Technologies

PANEL HEAD
Dr. R.I. Minu
Associate Professor
Dept. of Computing Technologies

CO-GUIDE(if any)
Name of the Co-guide
designation
Dept. of ——

HEAD OF THE DEPARTMENT
Dr. M. Pushpalatha
Professor
Dept. of Computing Technologies

INTERNAL EXAMINER

EXTERNAL EXAMINER

SRM Institute of Science & Technology
School of Computing
Department of Computing Technologies

Own Work Declaration Form

This sheet must be filled in and signed with date along with the student registration number, work will not be marked unless this is done.

To be completed by the student:

Degree/ Course : Bachelors in Technology/ Computer Science and Engg.

Student Name : Adhiraj Bhagawati and Rohit Ramesh

Registration Number : RA1811003010023 and RA1811003010055

Title of Work : Au-CARL: Autonomous vehicular navigation with Augmented Reality and Reinforced Learning

I / We hereby certify that this work compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this project is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

Rohit Ramesh

Adhiraj Bhagawati

Signature of the Student(s)

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr.C. Muthamizhchelvan**, Vice Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support. We would like to extend our sincerest gratitude to **Dr.T.V. Gopan**, Dean, College of Engineering and Technology. We would also like to extend our sincere thanks to **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her invaluable support. We wish to thank **Dr. M. Pushpalatha**, Professor & Head, Department of Computing Technologies, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

We are extremely grateful to our Panel Head, **Dr. R.I. Minu**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for his / her inputs during the project reviews. We register our immeasurable thanks to our Faculty Advisor, **Mr. U.M. Prakash**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to or guide, **Dr. R.I. Minu**, Associate Professor, Department of Department of Computing Technologies, for providing us an opportunity to pursue our project under her mentorship. She provided me the freedom and support to explore the research topics of my interest.

We sincerely thank staff and students of the Computing Technologies Department, SRM Institute of Science and Technology, for their help during my project work. Finally, we would like to thank our parents, our family members and our friends for their unconditional love, constant support and encouragement.

Adhiraj Bhagawati

Rohit Ramesh

ABSTRACT

This paper aims to introduce a culmination of Reinforced Learning (RL) techniques used for simulations of self learning cars and other agents or scenarios, and then comparing the results of the simulation in Augmented Reality (AR). The main idea is to use Unity3D and C-sharp to first create a rendition of a RL-learned self learning car simulation, and then use Unity ML agents to simulate other aspects of cars and RL to diversify, and then create a user-friendly autonomous explorations system in AR for educating students in grade school.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

AR Augmented Reality

ANN Artificial Neural Network

AI Artificial Intelligence

B2C Business-to-Consumer

DBO Derivative-based Optimization

DFO Derivative-free Optimization

GA Genetic Algorithm

IL Imitation Learning

ML Machine Learning

NN Neural Network

NPC Non-Playable Character

PCG Procedural Content Generation

PPO Proximal Policy Optimization

RL Reinforcement Learning

RTX Ray Tracing Texel eXtreme

SDK Software Development Kit

VR Virtual Reality

XR Extended Reality

CHAPTER 1

INTRODUCTION

1.1 Augmented Reality and Reinforcement Learning Systems

AR (Augmented Reality) is an interactive experience of a real-world environment in which computer-generated perceptual information is utilised to enhance the objects in the actual world, sometimes spanning many sensory modalities such as visual, auditory, haptic, somatosensory, and olfactory. The capacity to learn correlations between inputs, actions, and the occurrence of pleasurable or painful events, referred to as rewards or punishments, is required for reinforcement learning (RL). Combining the two new technologies is an innovative approach to addressing the drawbacks of each and also simplifies the interpretation of data obtained from two quite distinct systems.

Recent advancements in both subsections of the Computer Science field have prompted scientific-technology industry experts to work cordially with both Augmented Reality and Machine Learning. Especially in the field of Augmented Reality, where applications are no longer consumer-oriented computer vision applications. In the world of B2B applications, visualization plays a very important role of quote and proposal determinations, and Machine Learning helps in effectively determining prerequisite conditions for effective deployment of model onto environments, and based on evident research, we have observed that such visualization does not exist yet for elucidating the benefits of autonomous driving, which inclined us to perform the same.

Autonomous driving has been a contentious field of study and debate since its inception. It has been exceptionally scrutinized during the COVID-19 pandemic, due to its benefit of upholding social distancing practices, however it has always been held in contempt by vehicular safety activists and governmental transportation departments

alike. However, even with the penetration of automated vehicles into the automobile market in droves, only 2 accidents of the 187 recorded in automated vehicles were due to faults in the driving algorithm, implying that human intervention is still one of the largest threats to accidents in the automation industry.[1].

Using Machine Learning algorithms and running our test environments in Augmented Reality [2], we hope to ameliorate the poor name that autonomous navigation systems have cultivated over the past decade.

However due to software discrepancies and the arduous nature of the problem statements in AR, we decided to create a B2C version of our simulation targeting lower grade school students - teaching them about autonomous vehicles [3]. We aim to provide a suitable, user-friendly environment for students in grade school and above to educate them about the coming revolution in the autonomous driving vehicle industry and the impact that they will have on the next generation.

1.2 Brief module description

In our project, we have taken various scenarios in which a neural network could be used for applications in which self-learning cars are involved - like autonomous cars, accident prevention, and parking. This coupled with trying to simulate the environments in an AR environment, which would give a better understanding of the simulation is our focus. For the former application i.e. autonomous vehicles, we are using Unity3D to create a genetic algorithm (GA) to train the weights of the neural network (NN) that will be responsible for driving the car and avoid obstacles [13]. Sensor data is converted into actions by our system. This allows the automobile to respond to its changing surroundings and make the best decision possible in real time. Artificial Neural Networks (ANNs) utilize a variety of optimization methods, including derivative optimizations and derivative-free optimizations.

Derivative-based optimizations (DBOs) [13] enhance parameters iteratively by utilizing the derivative to determine the optimal search direction, effectively climbing the hill towards the global maxima. However, DFO (derivative free optimization) is

also a mathematical optimization approach that does not need the use of derivative information to improve a function. DFOs have been utilized to overcome some of the issues associated with DBO approaches, but they have remained mostly unexplored. Therefore, we chose to utilize a prominent DFO technique (a genetic algorithm) to identify the best weights of a neural network that would be used to drive a car in Unity because of the promise of this growing area.

We also had a prototype (see (Fig.1.1).) for the first application that we also made in Unity 3D, but chose not to go forward with. For that we used the math.net C-sharp library for the functions and data types needed to construct the neural network, but there were certain limitations and we couldn't implement a complex version of it using elevation and other factors and therefore it was scrapped.

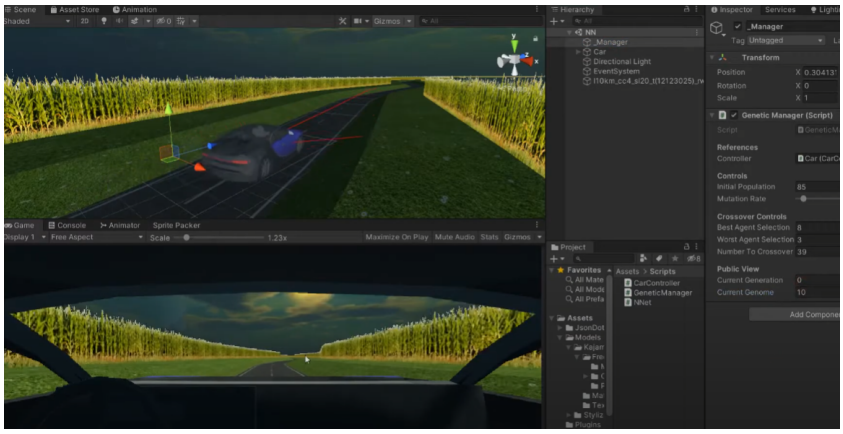


Figure 1.1: The first prototype made using math.net and Unity

For the applications in obstacle avoidance and parking, we have chosen to use Unity's state-of-the-art ML agents and Imitation Learning (IL), which is also a form of RL.

1.3 Conclusion

Throughout this paper, we show how genetic algorithms may be used in combination with neural networks to drive an automobile in this research. We'll also go through a set of Unity experiments that we ran using Unity's ML agents and illustrate various other examples related to autonomous vehicles. Our paper includes some instances of comparable work that has been done in the area, the design of our simulation, an

explanation of our technique, the findings we gained from running our simulation, and a conclusion.

CHAPTER 2

LITERATURE REVIEW

2.1 Existing Systms

In the research paper made using Unreal Engine 4 [1], the simulation platform appears to offer customizable sensor suite definition, climatic conditions, full control of all static and dynamic actors, map production, and much more.

Q-learning is also a technique used in the [1] paper, which can be further understood from the paper [22]. We also concluded that replicating the results shown in that paper would be next to impossible since it would require extremely fast computing power and also would not be entirely possible to implement with only 1 RTX nVidia card.

The paper [2] showed that the ML Agents' behavior became more human-like the more it was trained, since we know that the agents were being kept training until they reached human-like behavior and then they were deployed to an AR game.

In the [3] paper, we discovered that an AR interface was used in conjunction with a LEGO® SPIKE Prime robot to solve a treasure seeking RL issue in order to teach middle and high school students about RL ideas. By merging physical robots with a virtual AR interface, this strategy proved to be beneficial in providing students with more interactive and immersive AI learning. This is also in line with what we want to do by introducing AR into the foray.

[4] proposed a Procedural Content Generation (PCG) technique for creating new settings for Virtual Reality (VR) learning applications based on a Neural Network RL methodology. It was successful in mitigating the possible novelty effects by presenting them with new virtual environments.

The MEC-based AR system in [5] improved optimization, reduced space requirements, and improved adaptability of the AR application. We however could not

find a suitable solution for the RL aspects of our project to use the above-mentioned novel solution.

[6] had a similar idea to using another medium to showcase what the simulation could do in the real world by using a robot. The robot was successfully able to interact with obstacles thanks to the RL training of the agent and the Pixy camera sensor.

[7] illustrates the need of our project and the fact that it attempts to solve a very prevalent problem in today's world.

[8][9] are assets found in the Unity Asset store which we have used for our project. [10] helped us setting up the car in the first module with proper wheel colliders that are separate from the chassis of the car model, and thus behave differently with physics.

[11] and [12] are the foundation for the second and third modules, since we took the help of the YouTube videos and their implementation of the subject matter whereas [13] similarly helped us with implementing the first module.In [14], we can understand the various aspects of what RL and how they are implemented for various problems.

[15] provides us with information regarding autonomous trials from Cruise, which are evolving into autonomous services and how these services will create massive opportunities for the sensors market which in the next 20 years.

We can also see in [16], the fact that the authors used Unreal Engine 4 to craft NPC AI (Non-playable characters) that are used in video games, which do attest to the fact that we are also trying to use a 3D game engine to make automated AI, and to relate both of these papers, we can go even further to state that open-world games like GTA will be using similar methods for simulating their vehicle traffic. Finally, as per [17], the RGB images from a single camera, as well as their semantic segmentation, were provided to the driving policy.

They largely leveraged synthetic data, with tagged real-world data only appearing in the segmentation network's training. The use of RL in simulation and synthetic data reduced costs and engineering effort.

[18] is a fairly interesting application of Augmented Reality and Google's location API in the fact that, it is trying to provide users with a more intuitive way of navigating

the real world but in AR, with points of interests and many other features. [19] is fairly similar in concept to [18] but has a far more complex and intuitive system in it.

[20] involves the usage of computer vision, image processing techniques along with AR to process and relay information from real-world objects to the user's screen in AR. [21] tried to provide an inexpensive solution to reduce distraction for driver's by utilizing LEDs and AR to indicate turns on the road to drivers.

Q-learning is also a technique used in the [21] paper, which can be further understood from the paper [22].[23] used fairly similar methods by using ML agents along with Unity 3D to train their agents in the game. We also attribute a lot of our understanding of the concepts for RL in Unity 3D from the book [24].

[25][26] are similar to our thesis in the fact that they are using the same concepts in Unity 3D, in [25] the authors tried to shorten the time to train agents in a sparse environment, not unlike like our own. [27] is a fairly complex rendition of RL in the fact that, in the paper, their agent outperforms by averaging 880 percent human performance than the Atari.

[28] is very different than the other papers based on the fact that the paper is talking about using C++ for machine learning on larger projects that is more efficient and fast. [29][30] are extremely complex papers with the usage of RL, where in [29] there is usage of LidARs and GPS-GNSS systems and in [30] the agent learns to play a ping pong video game.

2.2 Inference from Surveys

S. no.	Objective	Tool(s) Used	Results	Remarks
1	Developed from the ground up to support development, training, and validation of autonomous driving systems.	Unreal Engine 4	The simulation platform supported flexible specification of sensor suites, environmental conditions, full control of all static and dynamic actors, maps generation and much more.	Requires extremely fast computing power and not entirely possible to implement with only 1 RTX nVidia card.
2	The agents were being kept training until they reached human-like behavior and then they were deployed to an AR game.	Unity 3D, Unity ML Agents, AR Core/ARKit	The results showed that the ML Agents' behavior became more human-like the more it was trained	NA
3	In this paper, an AR interface was combined with a LEGO® SPIKE Prime robot to solve a treasure hunting RL problem to introduce RL concepts to middle and high school students.	LEGO SPIKE Prime, ESP8266 microcontroller, Unity 3D, Vuforia SDK	Provided students with more interactive and immersive AI learning by combining physical robots with a virtual AR interface.	Hardware used.
4	Introduces a Procedural Content Generation (PCG) method based on a Neural Network RL approach that generates new environments for Virtual Reality (VR) learning applications.	Unity 3D, Unity ML agents, Google VR SDK	Mitigated possible novelty effects by presenting them with new virtual environments	Algorithm used is extremely complex.
5	Set up an MEC-based wireless AR system to meet the challenge of computing and storage resources in the AR application. The edge server provided generation of the AR content and assisted the AR device to learn the environment adaptively. Objective was "optimization"	Unity 3D, ARCore SDK, MEC-based web server for data.	Improved optimization, reduced space requirements, and improved adaptability of the AR application.	NA
6	To provide a robot and agent that can interact with both real-world obstacles and virtual obstacles in Unity.	Unity 3d, Vuforia SDK, ARDUINO UNO, CMU Pixy Camera Sensor, Google Cardboard/Hololens	The robot was successfully able to interact with obstacles thanks to the RL training of the agent and the Pixy camera sensor.	Hardware used.
7	To provide a resource for number of accidents	NA	Illustrates the need of our project and the fact that it attempts to solve a very prevalent problem in today's world.	NA

S. no.	Objective	Tool(s) Used	Results	Remarks
8	To make use of the unity asset store for 3D models.	NA	Were used to set up the race track for our hill track in the first module	Are assets found in the Unity Asset store which we have used for our project.
9	To make use of the unity asset store for 3D models.	NA	Were used to become the basis of the car models / agents for all modules	NA
10	To make use of the youtube instructional video for setting up wheel physics for the car in the first module	NA	Helped us set up the car in the first module with proper wheel colliders that are separate from the chassis of the car model, and thus behave differently with physics.	Usage of wheel colliders brought in the concept of wheel physics different to the car physics
11	Is the foundation for the third module, since we took the help of the YouTube videos and it's implementation of the subject matter	Unity 3D, ML Agents	Helped us to successfully set up the parking sim	Algorithm used is extremely complex. Used ML agents
12	Is the foundation for the second module	Unity 3D, ML Agents	Helped us to successfully set up the obstacle avoidance sim	Used Imitation Learning
13	Is the foundation for the first module which is a master's thesis from the University of Calgary	Unity 3D, C#	Helped us to set up the real-time learning autonomous driving sim	Complex and hard to train
14	To provide a resource for RL and it's uses	NA	Illustrates the various novel ways where RL could be used and it's basic concepts	NA

S. no.	Objective	Tool(s) Used	Results	Remarks
15	Provides us with information regarding autonomous trials from Cruise, Waymo which are evolving into autonomous services.	NA	Discussed about the strategies of OEMs and their attitudes towards autonomizing their models, what features they are bringing and what sensor suites.	These services will create massive opportunities for the sensors market in the next 20 years.
16	To make use of Unreal Engine to craft NPC AI	Unreal Engine 4	Authors used Unreal Engine 4 to craft NPC AI (Non-playable characters) that are used in video games	Does attest to the fact that we are also trying to use a 3D game engine to make automated AI
17	To make use of the RGB images from a single camera, as well as their semantic segmentation, for autonomous driving	Unreal Engine 4, CARLA simulator	The use of RL in simulation and synthetic data reduced costs and engineering effort.	NA
18	It is a fairly interesting application of Augmented Reality and Google's location API	Unity 3D, Google Maps / Location API, ARCore	It is trying to provide users with a more intuitive way of navigating the real world but in AR	NA
19	Tries a similar approach to the previous paper	Unity 3D	Trying to provide an ambitious rendition of AR and real-time location data	Algorithm used is extremely complex.
20	Usage of computer vision, image processing techniques along with AR to relay in AR.	Unity 3D	Usage of computer vision along with AR SDKs is pretty novel and got great results	Novel method
21	To provide an inexpensive solution to reduce distraction for driver's by utilizing LEDs and AR to indicate turns on the road to drivers.	Car, LED	Turned out to be extremely efficient	Hardware used

S. no.	Objective	Tool(s) Used	Results	Remarks
22	To understand the use of Q-learning in the field of RL	NA	The paper shows that Q-learning converges to the optimum action-values with probability 1.	Purely mathematical paper
23	Used ML agents along with Unity 3D to train the agents in their game	Unity 3D, ML Agents	The agents were properly trained and were able to fulfill requirements	Used fairly similar methods
24	A book which talks about using ML Agents and Unity3D	NA	A fairly comprehensible book with great concepts and reasoning regarding RL	NA
25	Thesis project that uses similar methods like us to train their agents	Unity 3D, ML Agents	The agents were properly trained and were able to fulfill requirements	NA
26	Thesis project that uses similar methods like us to train their agents	Unity 3D, ML Agents	The agents were properly trained and were able to fulfill requirements	NA
27	Introduction of an agent that also maximises many other pseudo-reward functions simultaneously by reinforcement learning.	NA	Their agent outperforms by averaging 880 percent more human performance than the Atari.	Extremely complex paper
28	Aims to use C++ for machine learning on larger projects that are more efficient and fast.	C++, IDEs	The results are extremely fast and efficient	Very different paper
29	Autonomous driving using very complex algorithms and solutions	LidARs and GPS-GNSS	Agents are extremely well trained	NA
30	Agent learns to play a ping pong game	Unity 3D	Agent is extremely well trained	Extremely complex paper

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 Introduction and brief explanation of existing systems

There aren't a lot of existing systems which include the use of Reinforcement Learning (RL) in Augmented Reality (AR). Adding to this fact, there are also not many applications out there that are willing to solve the issue of educating children and students about the emerging technologies that will be prevalent in their era. However, we do have a couple of papers talking about automation and the future of automation, and how it would look like when applied in the real-world scenario.

Let us begin by discussing one of those studies that discusses automation and its relation to autonomous cars - study [31] exemplifies this point. [31] is tasked with the responsibility of developing a suitable architecture for autonomous cars.

Today's academia, which focuses on similar subjects, presents past autonomous driving studies or implementations that are limited to certain places. As a result of building or evolving the autonomous vehicle, solutions arise.

One of the method's primary drawbacks is that requests cannot be tracked back to functional components, despite the fact that the majority of functionality is separated across many components. As a consequence, adopting recommendations without inside expertise is often difficult.

In today's world, we cannot classify autonomous driving as just a research topic. Automobiles are becoming more complex systems as manufacturers such as Tesla compete to increase the amount of vehicle automation.

Software is becoming the primary engine of innovation, as obsolete mechanical components become commodities and planning algorithms are held responsible for

critical decisions. Recent trends indicate that lowering human participation and error will increase transportation safety and efficiency.

The Society of Automotive Engineers (SAE) classifies the transfer of total control from humans to machines as a step-by-step process on a scale of 0 to 5, with 0 indicating no automation and 5 indicating full-time performance of all driving aspects by an automated driving system under all roadway and environmental conditions.

In layman's terms, what these basically means is the fact that, cars today are becoming more and more automated and therefore require heavy emphasis on utilizing good software and algorithms. As stated in the SAE (The Society of Automotive Engineers), we are giving more and more control over to machines, with Tesla cars today being a 3 or 4 on the scale of 0 to 5.

This literally elucidates the fact that we need greater reliability of software and as AI (Artificial Intelligence) takes over, and cars start being completely being a 5 on the scale, we need to teach the coming generation about the pros and cons they bring and how software and automation plays a big part in it - hence also reaffirming our stance on educating children and students with AR - as it tends to be easier and more efficient to explain them.

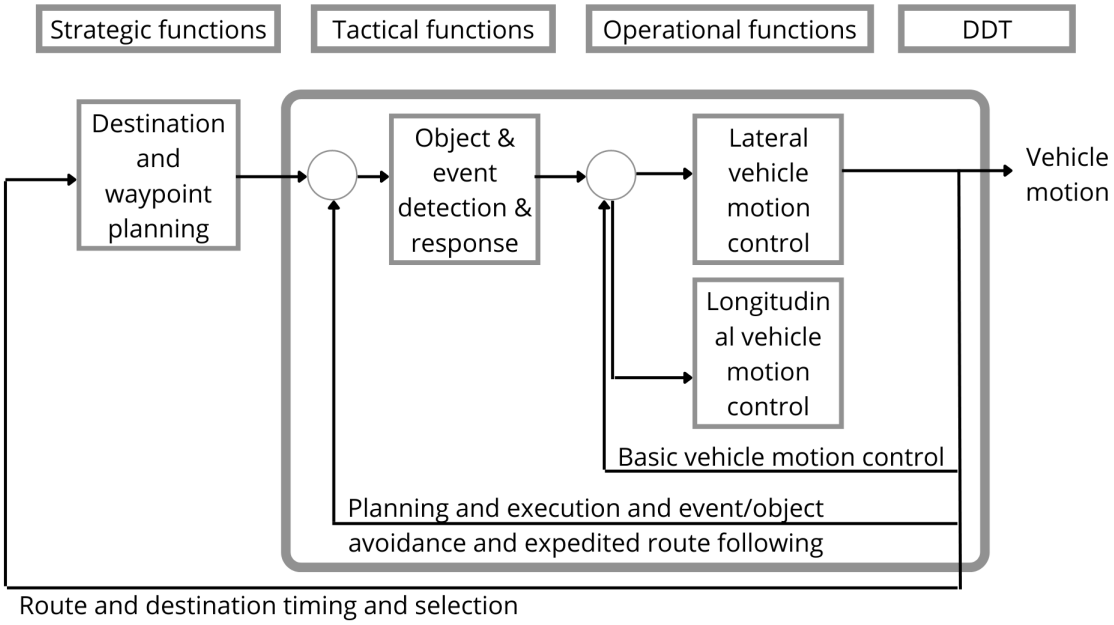


Figure 3.1: Functional component classification according to SAEJ3016

While not full, the list of components and their intended behaviour provided by

SAE J3016 is a good starting point. Each OEM is free to choose the implementation method for any requested functionality. For example, object detection, identification, and classification may be accomplished by a number of methodologies and software systems.

Because one OEM may choose to utilise several sensors and sophisticated sensor fusion algorithms, while another may prefer to produce a single component capable of performing all functions, this choice may have an effect on the final set of functional components. [31] sought to decompose each function into its constituent atomic components. This proposal will remain unchanged if any of these components are combined into higher level components.

A thorough investigation of such trade-offs is desperately required, since they are typically impacted by the distributed software development life cycle. According to the authors of [31], task automation is a control loop that gathers data from sensors, does some reasoning, and then acts on the environment (possibly through actuators).

Advanced task automation requires a deeper (semantic) interpretation of sensor data in order to generate higher-order judgements or plans. On the other hand, the loop behaviour is preserved. The SAE categorization of functional components follows the similar approach, as seen in 3.1. Each component type may be implemented as a control loop that accepts and acts on sensor input.

The amount of semantic knowledge necessary for decision-making varies according to job, from operational to strategic. Additionally, as we proceed from left to right, the components cease to control actuators and begin to control other components. For example, tactical components will provide directions to operational components.

Similarly, strategic functions cannot operate directly on actuators; rather, they must interact with tactical functions, which will then command operational functions.

3.2 Proposed System

3.2.1 Reinforcement Learning (Module 1)

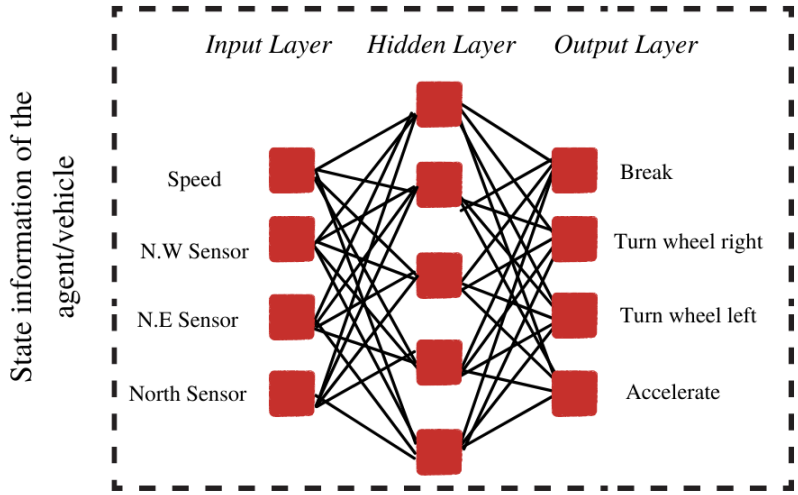
An environment, a genotype, a phenotype, operators, and a fitness function are all required by genetic algorithms. The 3D track we created will be the setting in which each prospective solution will interact.

$$weight1, weight2, weight3, ..., weightn, bias1, bias2, bias3, ..., biasn \tag{3.1}$$

$$where, weight, bias \in R$$

The genotype is a list of floats (3.1), each of which corresponds to a weight or bias in the neural network.

We will further illustrate how the NN looks later. (see Fig. 3.2 and Fig. 3.3).



Neural Network Structure for the autonomous driving simulation (Actor Network)

Figure 3.2: Architecture of NN used in the first application

To further explain (Fig. 3.2) and (Fig. 3.3), we need to understand how this NN

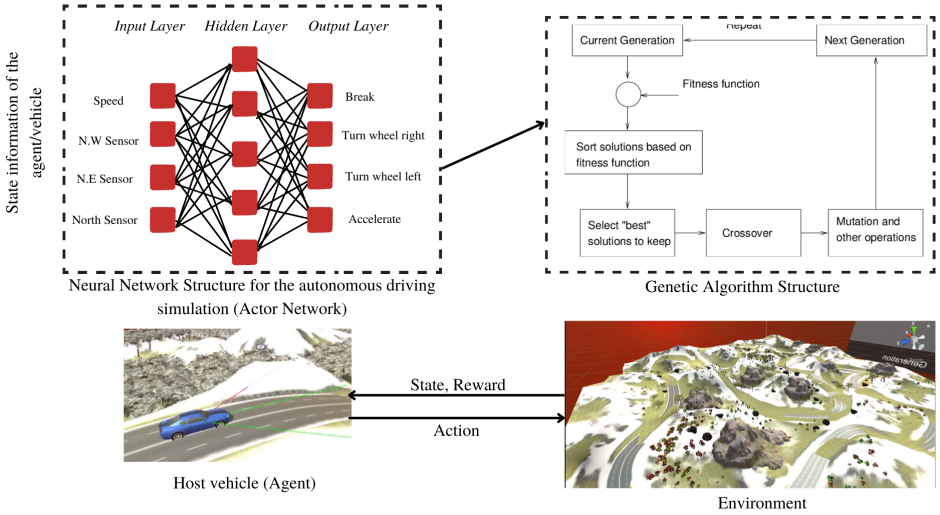


Figure 3.3: Architecture of the first application

(Neural Network exactly functions). In this very case, what we are essentially using is a "Multilayer feed forward neural network". This network has an internal hidden layer that has no direct interface with the exterior layer. The presence of one or more hidden layers makes the network computationally more powerful. There are no feedback links, therefore the model's outputs do not feed back into itself.

$$w_i^{(new)} = w_i^{(old)} + x_i y \tag{3.2}$$

where, (3.2) is known as the "Hebbian learning rule algorithm". In this rule, when two neurons on opposite sides of a synapse (connection) are activated at the same time, the strength of that synapse is selectively raised, resulting in the output neuron activating and vice versa.

There are four output neurons that indicate the desire to accelerate, brake, turn the wheel left, or right.

To figure out which output neuron has the greatest stimulus, we utilize argmax. The automobile will be driven by the urge linked with the most activated neuron. The above listed criteria are used to determine a vehicle's fitness.

When an automobile passes through a checkpoint, its fitness score is increased by one. As a result, the car that passes the most checkpoints is also the car that has driven

the greatest distance and so has the highest fitness score.

Our implementation just has a mutation function in terms of operations. To better understand this, we can provide an example with the help of (3.2).

$$\begin{aligned} w_{11}^{(new)} &= w_{11}^{(old)} + (speed)_1 y/ \\ w_{41}^{(new)} &= w_{41}^{(old)} + (N.sensor)_4 y \end{aligned} \tag{3.3}$$

where (speed) and (N.sensor) are input neurons which directly affect (break) and (accelerate) respectively (which are output neurons) with the 11 and 21 weights.

Similarily,

$$\begin{aligned} w_{33}^{(new)} &= w_{33}^{(old)} + (N.Esensor)_3 y/ \\ w_{22}^{(new)} &= w_{22}^{(old)} + (N.Wsensor)_2 y \end{aligned} \tag{3.4}$$

where (N.Esensor) and (N.Wsensor) are input neurons which affect (turn wheel right) and (turn wheel left) and are output neurons with the 33 and 22 weights.

(3.3) and (3.4) also show us that the input neurons can also affect other output neurons like for example (speed) with weights 14 can directly affect the (acceleration) output neuron. The inputs will be sent forward through the neural network, with tanh serving as the activation function in each neuron [13].

From this we can infer that, since all input neurons are connected to all output neurons, the cumulative effect in the output of the agent is dependent on all the input neurons and their respective weights.

In layman's terms, if the speed decreases, then there would be no need to break, but the 14 weight would be activated to activate more acceleration. To figure out which output neuron has the greatest stimulus, we utilized argmax. The automobile will be driven by the urge linked with the most activated neuron.

3.2.2 Imitation Learning (Module 2 and 3)

For the second and third applications [11][12], there are certain algorithms that Unity has implemented in their ML Agents toolkit, it just needed to prepare the car movement scripts, the spawners that were needed for imitation learning.

For obstacle avoidance, we first set up the cars to go in a straight path and made them agents that have the "agent" script to work off of.

We then implemented a spawner that always instantiates cars on the other end in regular time intervals. We then train the cars (agents) by starting the training environment as shown in (3.5).

$$mlagents - learntrainer\_config.yaml - -run - id = AI\_2 - -env = ../Build/build.app \\ - -time - scale = 10 - -quality - level = 0 - -width = 512 - -height = 512 \tag{3.5}$$

We can train the agents multiple times, using the anaconda and the tensorflow environment along with ML Agents. This results in the cars learning and models getting saved in order to be used later in the main session. These NN models are recognized by Unity as ".onyx" which can be read and utilized by Unity3D.

For the self-parking system [11], we constructed custom hyperparameters, along with a new yaml file, as it's visible in (Fig. 3.4), the name is SelfParking.yaml. As we can see from (3.5), we need a .yaml file to run the commands in the terminal in cmd or in our case the anaconda prompt.

We had to tune the hyperparameters a bit since the available ones were not doing a good job in training the agents and were very less efficient. Although, even our current simulation turned out to be less efficient too, albeit more efficient in training the agents than the former hyperparameter configurations.

The implementation is a bit similar to the first application of autonomous driving, in the fact that we are using ray-casts (Fig. 3.4) to determine where the car should park and where the car shouldn't hit.

D: > Unity Projects > CS Final Project > Parking Sim > Assets > Config > ! SelfParking.yaml

```yaml
behaviors:
  CarBehavior:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 2048
      learning_rate: 0.0003
      beta: 0.01
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 10000000
    time_horizon: 64
    summary_freq: 5000
    threaded: true
```

Figure 3.4: The new .yaml file with the custom hyper-parameters

Similar to the first application, all those ray-casts are sensors and they are responsible for the agent to realize where the car should head in real-time after training.

We have included a rough figure (Fig. 3.6) to illustrate the number of sensors, degrees, and length of the rays. Before that, we set a couple of transforms and quaternions (positions and 3D rotations respectively in the Unity system) to get the places where the car / agent should be parked.

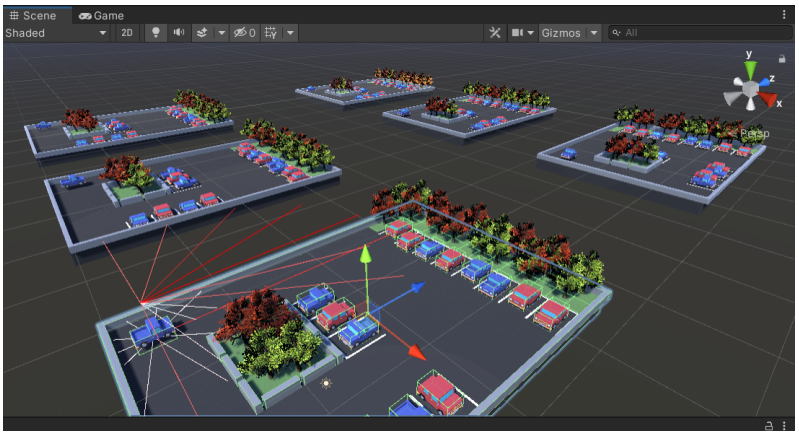


Figure 3.5: The ray-casts can be seen protruding from the agent

We have used imitation learning, and the agent learned through rewards, for example - we gave the agent a positive reward if it parked in the parking lot / space that we gave it, and if it collided with the walls or any other thing, we gave it a negative reward of -1.

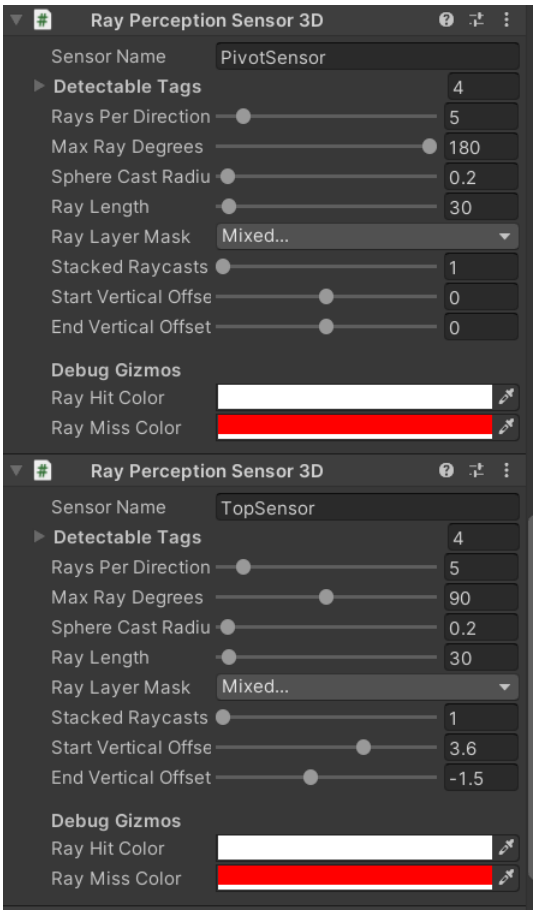


Figure 3.6: The ray-casts info for the third application

The agent and the academy learn with the help of the sensors / raycasts put in place in the agents. We can illustrate various ways regarding how the imitation learning and RL works in ML Agents but we shall properly illustrate them using diagrams. (Fig. 3.7).

We shall also illustrate the influence of the python API and Anaconda in the training process of the neural network. (Fig. 3.8) and how the second simulation looks (Fig. 6.1).

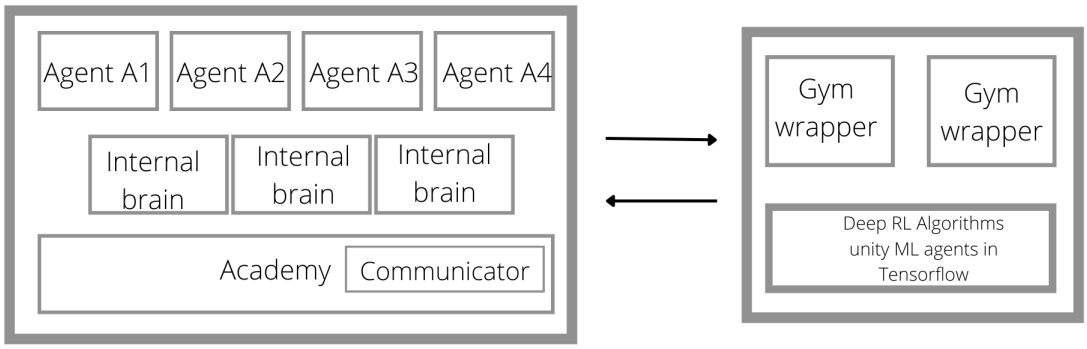


Figure 3.7: State diagram of the RL system in Unity ML Agents

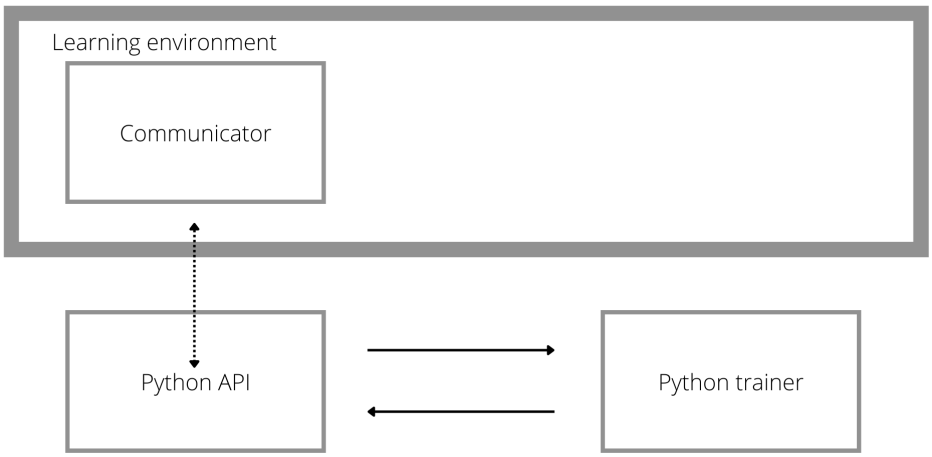


Figure 3.8: State diagram of the Python API in Unity ML Agents

3.2.3 AR Architecture (Module 4)

(Fig. 3.10) elucidates the gist of our AR architectural system. We are using Unity's built in XR (Extended Reality) developer tools (the XR interaction toolkit) which comprises the AR Foundation package.

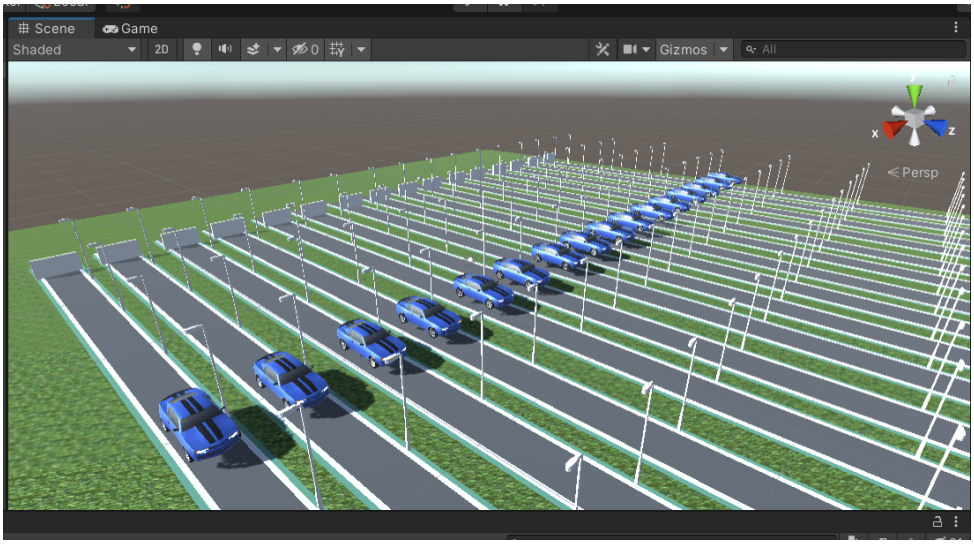


Figure 3.9: Simulation of obstacle avoidance using ML-Agents

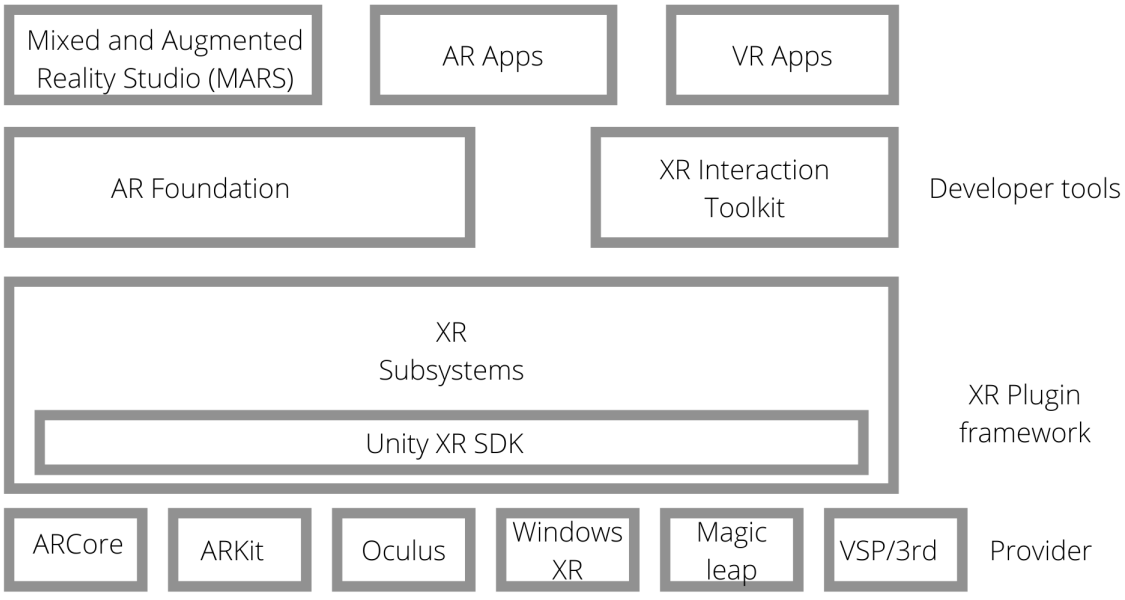


Figure 3.10: Unity 3D's complex XR architectural system

To leverage the power of AR Foundation, we have used ARCore SDK which provides the connection between the AR features and AR Foundation as a base framework.

As we can see in (Fig. 6.1), the XR plugin framework for Unity 3D is quite diverse. To explain how we have used it, we'll go into detail here:

1. Display, Input and Environment: For the display, we are using the AR Camera found in the AR Foundation and ARCore SDK to detect the camera of the mobile phone / device using the application. In our case, it's using the back camera to feed-in the real-world environment to the display.

2. Faces, Raycast, Camera and Planes: We aren't using plane detection to detect faces however we are using raycast to detect horizontal and vertical planes inside the room / real-world environment. This is why the 3D model / simulation of our environment is augmented horizontally in front of the camera.

3. Object Tracking and Meshing: ARCore requires OpenGL as a renderer to use and therefore Vulcan has to be excluded. In our cases, we took the simulation of the second and third applications, and put them inside a single empty game-object, and made them a prefab. This prefab containing the entirety of the simulation is treated as a single game-object by ARCore and AR Foundation and is thus tracked and meshed with the rendered and is augmented in real-time.

CHAPTER 4

METHODOLOGY

For the first application, which is autonomous driving, we set out to genetic algorithm (DFO) and Unity's pre-built physics engine. For the second and third applications - obstacle avoidance and parking, we chose to use Unity's ML Agents toolkit and Anaconda and other parts involved in the ML Agent process. For module 4, we used Unity-compatible Google ARCore SDK to create the required scenes to anchor the testing environments onto an augmented space.

We used Unity to create the simulation because of its pre-built physics engine. We were able to develop a realistic driving experience [10] with reduced overhead as a result of this. Our simulation includes a 3D track [13] with a road and obstacles on both sides of the road that was loaded from the Unity Asset store [8] (see Fig. 13).

4.1 Module 1: Reinforcement Learning

The main goal of the first iteration [13] and application was to train a neural network to travel a track independently without crashing. Back propagation is the most popular method for training neural networks, however it requires a labeled dataset, which is difficult and time-consuming to produce for this problem.

A genetic algorithm (DFO technique) was chosen as the most appealing method since it does not require a labeled dataset and simply requires a testing environment to evaluate potential solutions.

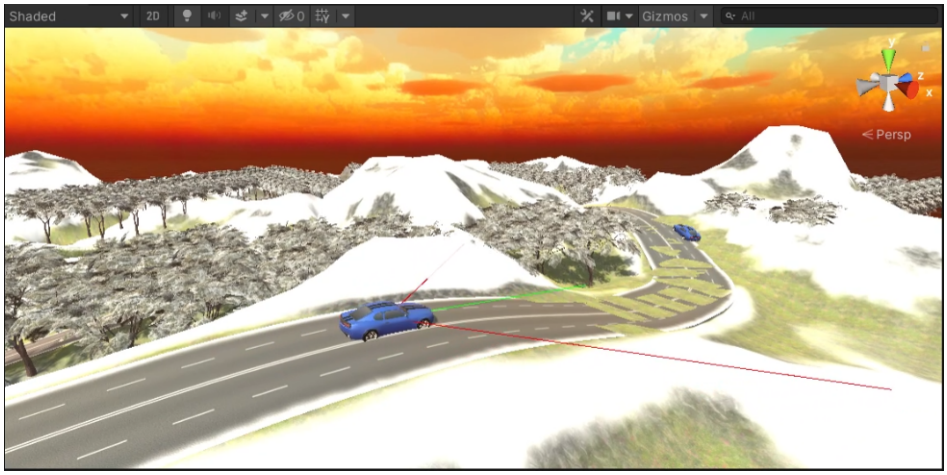


Figure 4.1: 3D track and the car asset equipped with proximity sensors

The road is the surface on which each automobile may drive, and the barriers are intended to keep cars in their designated lanes. When an automobile collides with a barrier, it comes to a complete halt.

We also included checkpoints, which are invisible indicators that are evenly distributed across the route. Sharp bends and steep slopes on the circuit presented an intriguing set of obstacles to which potential solutions had to respond.

An automobile that accelerates too quickly around a bend will not be able to complete it. To properly ascend the steep slopes, a car must acquire enough speed and apply appropriate brakes when rolling down to prevent acquiring too much speed. We used the Unity store to import a vehicle [2][9] that had the car's body and wheels (see Fig. 1) but no movement routines.

We were able to recreate the car's acceleration, braking, and turning using the techniques described in [1]. After that, we attached proximity sensors to the automobile, allowing it to investigate its surroundings.

Three proximity sensors followed the automobile for 20 meters to the north, northeast, and northwest. These proximity sensors use Unity raycasting and return a number between 0 and 20 depending on which barrier the raycast collides with.

The car's fitness is determined by the checkpoints described above. A car's fitness score is increased by one when it passes through a checkpoint. As a result, the car that passes the most checkpoints is also the car that has driven the greatest distance and so

has the highest fitness score.

We've built a Gaussian mutation function, in which each gene has a chance to be modified based on the mutation rate set by the user. We generate a random number r_i $\in$ (-mutation radius, mutation radius)
for each element in the genome, with the mutation radius being a float that the user may adjust [13]. The algorithm is as follows [13]:

1. A population of n number of cars is created using NNs (Neural Networks) that have been randomly initialized.
2. All of the vehicles will be populated within the track in the 3D environment.
3. The simulation will halt until all the cars have stopped or have hit the borders of the road in that generation.
4. The cars will be sorted in accordance with their fitness and will be populated in the next generation.
5. The chosen cars will be cloned again and again.
6. Until the track is completed by a car, step 2 will keep reoccurring.

As we have stated earlier, we also had a prototype of the first application that we developed in Unity 3D but using the math.net C-sharp library, which was scrapped. In that, we didn't have complex elevation and couldn't perform complex calculations and computations. content

4.2 Module 2 and 3: Imitation Learning

We also wanted to use the same principle of neural networks for the other two applications i.e. obstacle avoidance and parking, however, that turned out to be harder without using ML Agents. So, in this case, we used imitation learning (IL) by first playing that game and the agents kept learning as we will be illustrating later in this paper through figures and data.

The TensorFlow library, which is used to efficiently train the agents, is utilized via the Python API. Agents, Brains, and the Academy are the three core components of the toolkit. The agents can be connected to several brains or share a single brain, which is

then connected to the academy, which uses the knowledge base for successful decision-making once again.

However, the more agents you utilize, the faster the training process will be. That is precisely why we have multiple agents to speed up the process.

We also wanted to use the same principle of neural networks for the other two applications i.e. obstacle avoidance and parking, however, that turned out to be harder without using ML Agents. So, in this case, we used imitation learning (IL) by first playing that game and the agents kept learning as we will be illustrating later in this paper through figures and data.

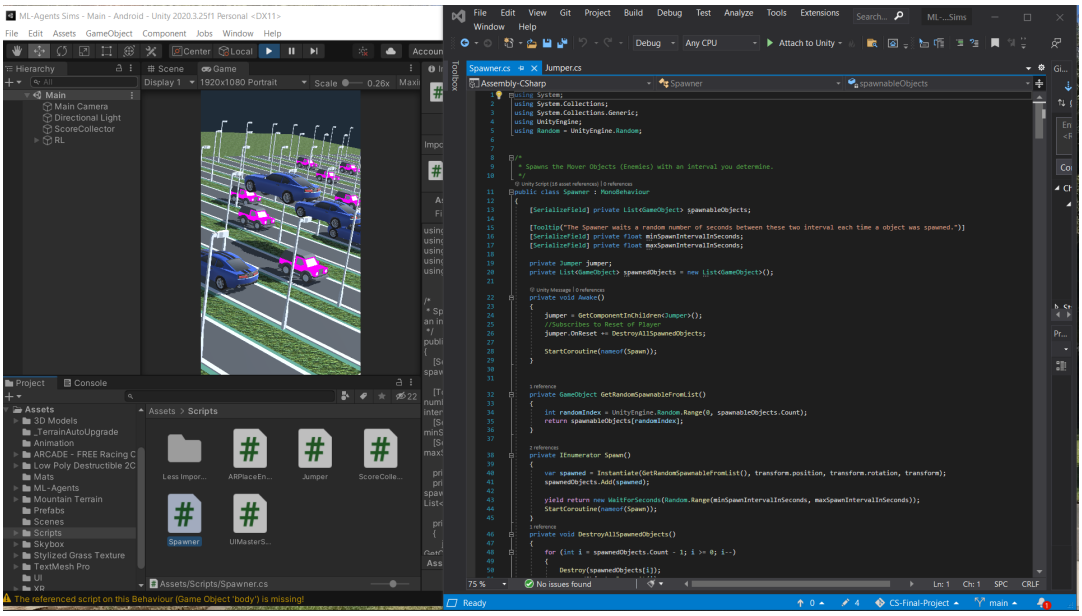


Figure 4.2: General method for module 2

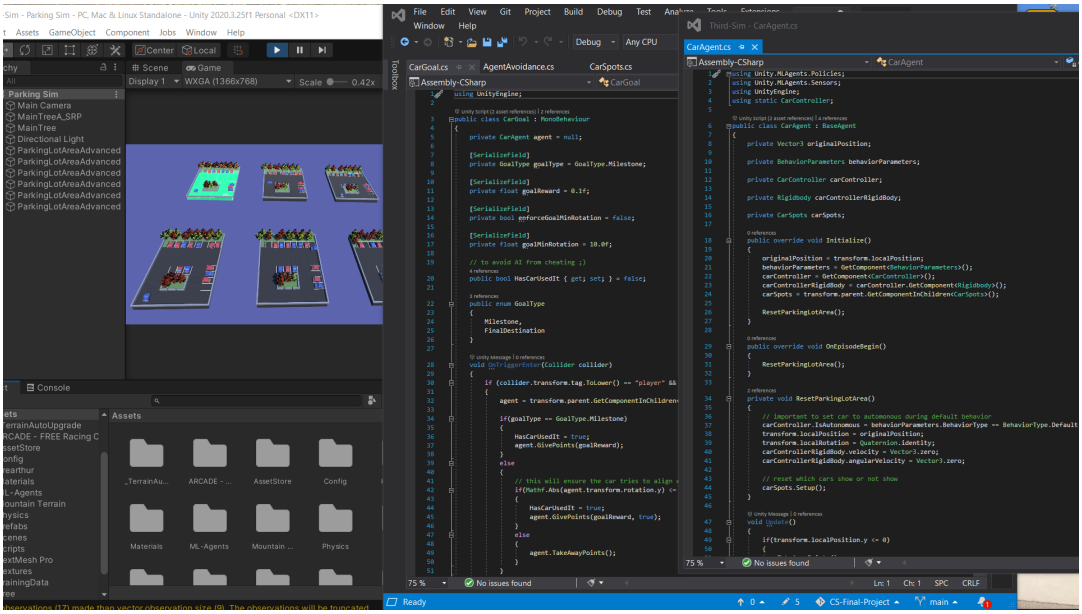


Figure 4.3: General method for module 3

4.3 Module 4: Implementation into Augmented Reality

For our last module, which is our AR application for students in grade-school and as mentioned above, [2][3]we used Unity 3D, Google ARCore SDK, and AR Foundation. Google AR Core SDK includes native APIs for motion tracking, environmental comprehension, and light estimates, among other AR capabilities.

With these tools, we can create totally new AR experiences or add AR functionality to existing apps, as we have done the latter for this paper. Within Unity, the AR Foundation lets us deal with augmented reality systems across several platforms.

This package provides an interface for Unity developers to employ, but it does not include any AR functionality.

To leverage AR Foundation on a target device, we additionally need extra packages for the Unity-supported target platforms: On Android, there's an AR Core XR plugin, while on iOS, there's an AR Kit XR plugin, an Magic Leap Windows XR Plugin for HoloLens.

We then aim to create a user-friendly application for students in grade school [3] and above for teaching them about these systems and the impact they will have on the

coming generations.

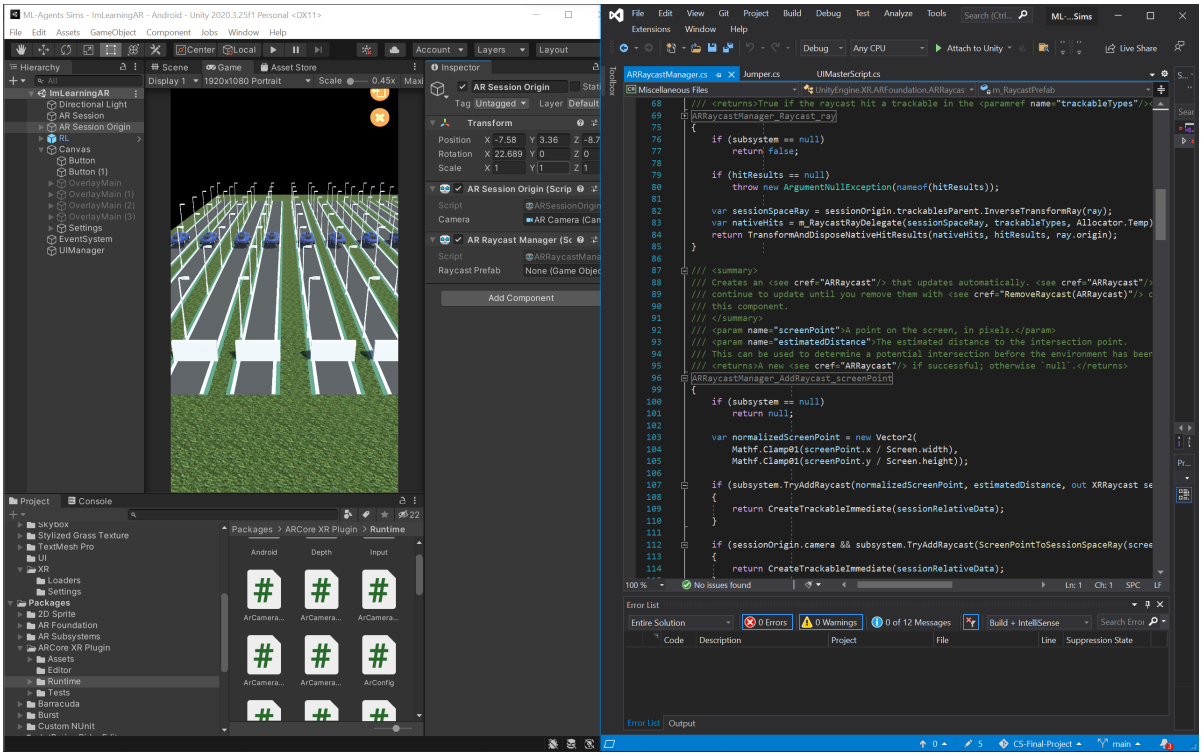


Figure 4.4: General application idea for module 4

CHAPTER 5

CODING AND TESTING

5.1 Module 1

```
using system.Collections.Generic;
using UnityEngine;

public class {Name of script}: Monobehaviour
{
  public int mut_rate;
  public float mut_rad;
  public int pop_size;
  public str weight_file;

  private Genepool genePool;

  start{
    if (pop_size mod 2 is not equal to 0 | pop_size is lesser than 2)
    {
    pop_size is equal to 50;
    }
    if (weight_file is null)
    {
      try{
        genepool = new Genepool(new list, size etc)
      }
      catch{
        genepool = new Genepool(new list, size etc)
      }
     else
     {
      repeat
     }
     invoke(checkongeneration());
     dashboard.initialize(getgenerationdata());
    }
  }
}
 void checkongeneration()
  {
     if (genePool is still alive == false)
     {
        {sort genepool} & report the best one
        {update dashboard with new genepool}

        GameObject[] allPlayers = {find all gameobjects with tag “player”}
        for (int i = 0; i < allPlayers.Length; i++)
        {
          {destroy all players}
        }
        generationNumber++;
        dashboard.UpdateGeneration(generationNumber);
        genePool.NextGeneration();
     }
  }
```

Figure 5.1: Pseudo code for module 1

In this pseudo code, we’ll be talking about our "Simulation" script.

Firstly, we are importing UnityEngine and systems.collections.generic in the script for Visual Basic to recognize that the code will be executed for Unity 3D. Since, C sharp is a purely OOPS-based language, we begin evert script by declaring a class with the same name as the script name in the environment, and then derive it from "Monobehaviour", where "Monobehaviour" is the main class in Unity 3D which contains all the functions, variables and other things recognized by Unity 3D. For example, void start() and void update() are two functions that are executed before the frames start and during the frame / simulation respectively.

We declare all our required variables such as mutation rate, radius, population size and the weight file that contains all the weights for the neural network. Then, in void start() method, we first try to get a even and positive populations size with which we later populate them in the genepool itself where Genepool is another script that has the same class name. If the user has chosen to load weights with checkongeneration(), then we shall initialize the dashboard.

In checkongeneration() method, what we are basically doing is sort all the population based on the best fitness, report the best one and populate it to the next generation again. Also, if genepool isn't alive and has crashed, then we destroy all gameobjects by finding them with the tag "player".

5.2 Module 2

```
using Unity.MLAgents;
using Unity.MLAgents.Sensors;

public class Jumper : Agent
{
    [SerializeField] private float jumpForce;
    [SerializeField] private KeyCode jumpKey;

    private bool jumpIsReady = true;
    private Rigidbody rBody;
    private Vector3 startingPosition;
    private int score = 0;

    public event Action OnReset;

    private void OnCollisionEnter(Collision collidedObj)
    {
        if (collidedObj.gameObject.CompareTag("Street"))
            jumpIsReady = true;

        else if (collidedObj.gameObject.CompareTag("Mover") || collidedObj.gameObject.CompareTag("DoubleMover"))
        {
            AddReward(-1.0f);
            EndEpisode();
        }
    }

    private void OnTriggerEnter(Collider collidedObj)
    {
        if (collidedObj.gameObject.CompareTag("score"))
        {
            AddReward(0.1f);
            score++;
            ScoreCollector.Instance.AddScore(score);
        }
    }

    public override void OnActionReceived(float[] vectorAction)
    {
        if (Mathf.FloorToInt(vectorAction[0]) == 1)
            Jump();
    }

    private void Jump()
    {
        if (jumpIsReady)
        {
            rBody.AddForce(new Vector3(0, jumpForce, 0), ForceMode.VelocityChange);
            jumpIsReady = false;
        }
    }
```

Figure 5.2: Pseudo code for module 2

In this pseudo code, we'll be talking about our "Jumper" script.

Here we import everything that we imported in the first module, but we also import unity.mlagenents and unity.mlagents.sensors which are the driving force behind the ML Agents brain and academy to recognize the C sharp script. After that, we declare the class Jumper which is also the script's name and declare certain variables like jump force and jump keys which we used to train the agents through imitation learning (where

we played the game ourselves for the agents to learn). We also declare Vector3 and rigidbody which come from monobehaviour in Unity and represent 3D position and body physics respectively in the Unity 3D engine.

Next, we declare Ontrigger() and Oncollision() methods which basically represent two or more gameobjects colliding together in the simulation. In those methods, we have defined rules such as when the object collided with is the "street" which has the tag "street", then the object is ready to jump (which is a bool) and in case the object does jump and does not trigger the oncoming obstacle / cars, then we add reward of 1 or -1 and vice versa depending on if the agent performs well or not.

5.3 Module 3

```
using UnityEngine;

public class CarGoal : MonoBehaviour
{
    private CarAgent agent = null;

    [SerializeField]
    private GoalType goalType = GoalType.Milestone;

    [SerializeField]
    private float goalReward = 0.1f;

    [SerializeField]
    private bool enforceGoalMinRotation = false;

    [SerializeField]
    private float goalMinRotation = 10.0f;

    // to avoid AI from cheating ;)
    public bool HasCarUsedIt { get; set; } = false;

    public enum GoalType
    {
        Milestone,
        FinalDestination
    }

    void OnTriggerEnter(Collider collider)
    {
        if (collider.transform.tag.ToLower() == "player" && !HasCarUsedIt)
        {
            agent = transform.parent.GetComponentInChildren<CarAgent>();

            if(goalType == GoalType.Milestone)
            {
                HasCarUsedIt = true;
                agent.GivePoints(goalReward);
            }
            else
            {
                // this will ensure the car tries to align when parking
                if(Mathf.Abs(agent.transform.rotation.y) <= goalMinRotation || !enforceGoalMinRotation)
                {
                    HasCarUsedIt = true;
                    agent.GivePoints(goalReward, true);
                }
                else
                {
                    agent.TakeAwayPoints();
                }
            }
        }
    }
```

Figure 5.3: Pseudo code for module 3

In this pseudo code, we'll be talking about our "CarGoal" script.

Here, we again import the required and then declare the class with the same name as the script. We then declare a couple of variables with [Serializefield] which means that the variables are secure and private, but can also be edited inside the Unity 3D editor (which is very convenient). We also declare goaltypes such as final destination and milestone for the agents to recognize. Then we use the ontriggerenter() method

which gives us a way to get information about the agent/ car and whether it has reached the goal and then get it's children. If the car has done the needful, we then reward it with a certain amount. We are also using math.abs to calculate the agent's rotation and quaternion to see whether it has parked in it's given spot properly, and reward it accordingly.

5.4 Module 4

```
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

[RequireComponent(typeof(ARAnchorManager))]
[RequireComponent(typeof(ARRaycastManager))]
[RequireComponent(typeof(ARPlaneManager))]
public class ARPlaceEnvironment : MonoBehaviour
{
   [SerializeField] private GameObject _prefabToPlace;
   private ARRaycastManager _raycastManager;
   private ARAnchorManager _anchorManager;
   private static readonly List<ARRaycastHit> Hits = new List<ARRaycastHit>();

   protected void Awake()
   {
      _raycastManager = GetComponent<ARRaycastManager>();
      _anchorManager = GetComponent<ARAnchorManager>();
      //_planeManager = GetComponent<ARPlaneManager>();
   }
   protected void Update()
   {
      Touch touch;
      if (Input.touchCount < 1 || (touch = Input.GetTouch(0)).phase != TouchPhase.Began) { return; }

      const TrackableType trackableTypes =
         TrackableType.FeaturePoint |
         TrackableType.PlaneWithinPolygon;

      if (_raycastManager.Raycast(touch.position, Hits, trackableTypes))
      {
         CreateAnchor(Hits[0]);
      }
      ARAnchor CreateAnchor(in ARRaycastHit hit)
      {
         ARAnchor anchor;
         if (hit.trackable is ARPlane hitPlane)
         {
            var oldPrefab = _anchorManager.anchorPrefab;
            _anchorManager.anchorPrefab = _prefabToPlace;
            anchor = _anchorManager.AttachAnchor(hitPlane, hit.pose);
            _anchorManager.anchorPrefab = oldPrefab;
         }
         else
         {
            var instantiatedObject = Instantiate(_prefabToPlace, hit.pose.position, hit.pose.rotation);

            // Make sure the new GameObject has an ARAnchor component.
            if (!instantiatedObject.TryGetComponent<ARAnchor>(out anchor))
            {
               anchor = instantiatedObject.AddComponent<ARAnchor>();
            }
         }
         return anchor;
```

Figure 5.4: Pseudo code for module 4

In this pseudo code, we'll be talking about our "AR placer" script.

Firstly, we shall import XR.ARFoundation and XR.ARSubsystems which are necessary to utilize the variables, methods that are dependent on XR, AR, VR in Unity 3D and the IDE. We shall also use requirecomponent to put the ARAnchorManager, RayCastManager and PlaneManager scripts inside the game object where this very script will be attached. This is an example of bending the engine to your will and saves

valuable time of the developers that spend on interacting the UI of the engine by dragging and dropping. Now talking about the awake() method, this is used when we want the code inside this method to execute at the first and before any other calculations, simulations etc. Inside awake(), we are initializing raycast (which are basically rays from the camera that hit the ground etc and understand if there is ground there or just open), anchormanager that anchors the gameobject in a specific place and AR placer simply places the object in AR. Lastly, we are using Touch which comes under mobile lingo and it basically detects screen touches and whenever it detects screen touches, the object is placed in AR. Also, prefab in this case are gameobjects, that cannot be changed anymore but are configured in a particular way to allow multiple re-uses.

CHAPTER 6

RESULTS AND OBSERVATIONS

While we were not able to successfully compare the results drawn from the simulations run on the PC and mobile AR environment, we were able to successfully simulate the autonomous navigation system's modules as standalone applications on both PC and mobile AR environments.

As previously mentioned, due to hardware limitations on our PC devices, the training time and hence results were affected. Furthermore, due to mobile hardware limitations, we were not able to retrieve any valuable simulation results and had to resort to simply implementing the simulation as a B2C-friendly application.

However, we have inferred particular results from the simulations which do give a proper conclusion.

6.1 Module 1 Results

(Fig. 6.1) illustrates the various values including mutation, generation, speed, and fitness and the graphs involved in the first simulation that gives us an idea about the performance of the simulation.

If we have to explain "fitness", it is simply a function that takes a candidate solution to a problem as input and outputs how "fit" or "excellent" the answer is with regard to the problem at hand.

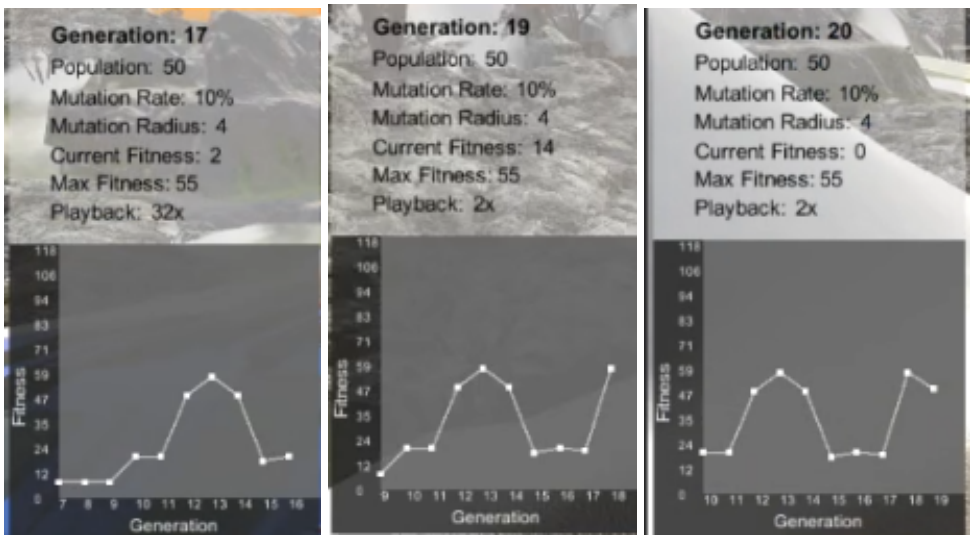


Figure 6.1: Fitness-generation graph of the self-learning algorithm (first application)

6.2 Module 2 Results

(Fig. 6.2) gives us the various graphs and observations that we got from training the neural network in the tensorflow-anaconda-MLAgents environment.

Also we can notice in (Fig. 6.2), that our second graph is noticeably different. This is because more agents have allowed us to handle many more steps in the same amount of time. This is why we start basic - instead of taking hours of training time, we were able to validate that our environment was operating in only a few minutes, instead of a few hours.

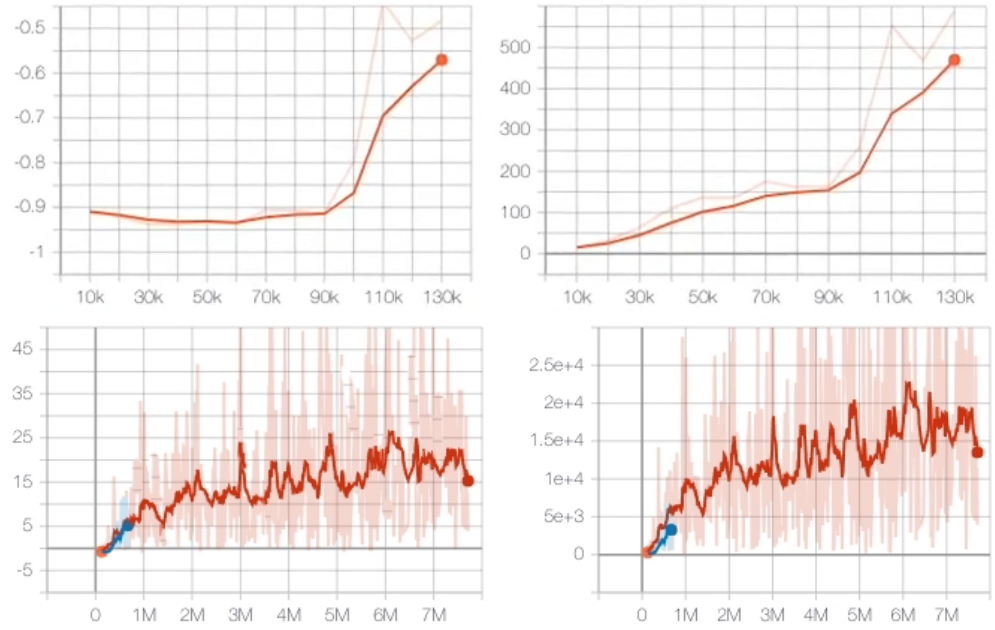


Figure 6.2: The graph on the left reveals the reward accumulated over time and the graph on the right for episode length over time for the second application of obstacle avoidance

6.3 Module 3 Results

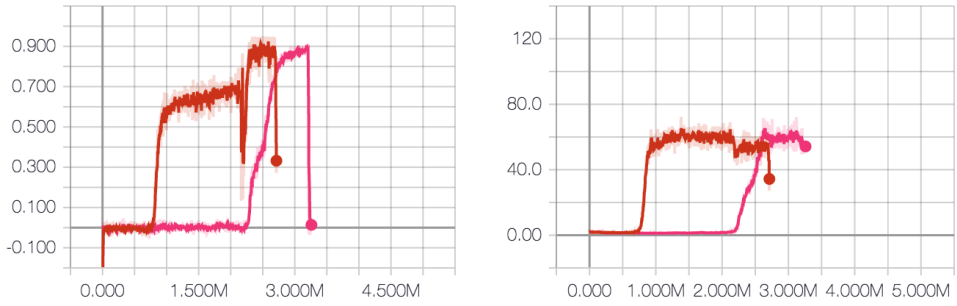


Figure 6.3: The graph on the left reveals the reward accumulated over time and the graph on the right for episode length over time for the third application of the parking simulation

(Fig. 6.3.) illustrates the performance of the agents trained for the third application. As we can see from (Fig. 6.1), we are using the default training configuration for PPO, and the rewards have been normalized between -1 and 1.

As we discussed earlier, our agents turned out to be less efficient and that is why we can see the cumulative rewards for the agents falling down after some time during training.

To explain (Fig. 6.2) and (Fig. 6.3) better we need to look at these terms and understand what they are: The agent that we have specified about, receives the state of the environment at each time step, and the agent picks an appropriate response in accordance.

If the agent behaves adequately and has good results in accordance with the state, then the agent receives a reward and a new state one time step later. The objective of all agents is to maximize the predicted cumulative reward [14].

1. Episodic vs. Continuing Tasks:
   (a) The term "continuing tasks" refers to tasks that go on indefinitely.
   (b) Tasks having a well-defined beginning and end point are known as episodic tasks. A whole sequence of interaction, from beginning to end, is referred to as an episode in this circumstance.
   * When the agent achieves a terminal state, episodic tasks come to an end.
2. Cumulative Reward:
   (a) At time step t, the discounted return is

$$G(t) = R(t+1) + \gamma * R(t+2) + \gamma^2 * R(t+3) + \dots \tag{6.1}$$

   (b) The agent chooses activities to maximize the predicted (discounted) return.

6.4 Module 4 Results

(Fig. 6.4) and (Fig. 6.5) elucidates the entry and main application for a baseline application we created aimed at elucidating the benefits of autonomous navigation systems to an inquisitive younger audience, mostly young grade school students.

On entry, users are greeted with a login screen (Fig. 6.4) which validates user credentials and connects the user to the main application scene (Fig. 6.5).

On entering the application, an AR Session is started, which prompts the object in the scene to be placed on the first plane that it detects (unfortunately due to directory issues in Unity ML-Agents and Unity ARCore SDK, there is no proper plane detection and object placement in the project).

Along with the simulation being run in an AR environment, there are also indicators that explain the proceedings of the simulation, in a rather rudimentary fashion, making it more palatable for a younger audience.

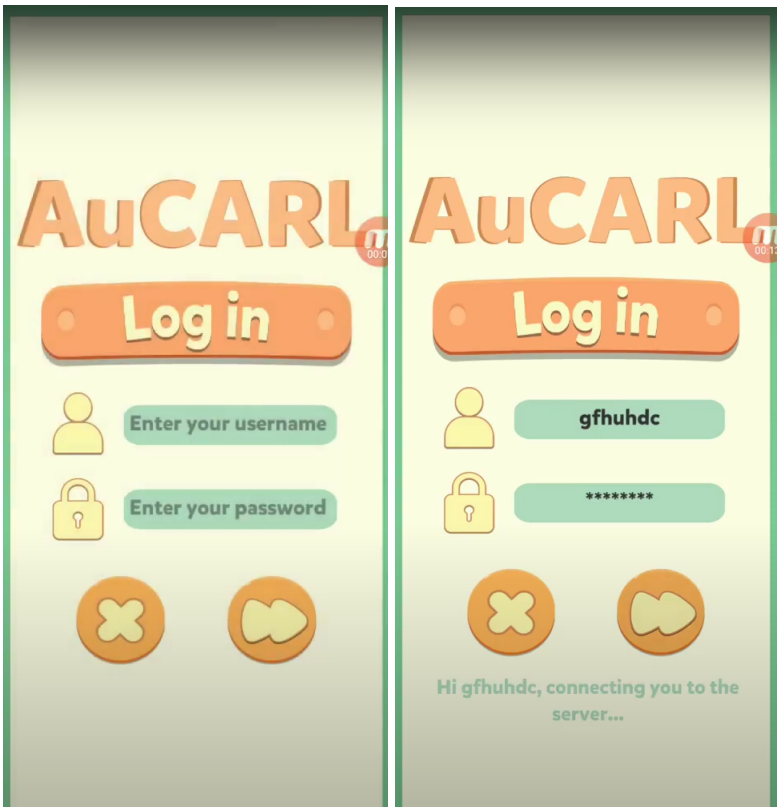


Figure 6.4: AR application login screen

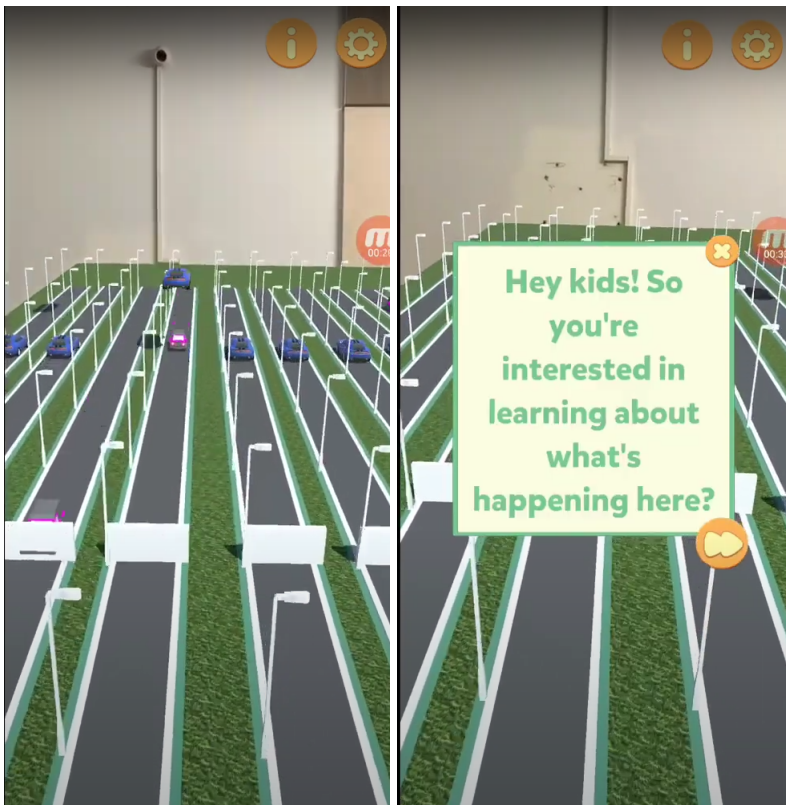


Figure 6.5: AR application main screen

The other scenes which display the other simulation environments will also be made available in the application to explain to students the process of how the models train our test GameObject through the environment and highlight the benefits of autonomous terrain maneuvering and parking to invested students.

CHAPTER 7

CONCLUSION

We were able to successfully infer results from the three working modules of the autonomous vehicular navigation system, and developed a seemingly robust educational application to demonstrate the working and benefits of the same, which can be easily distributed to an invested audience.

It was observed that by successfully implementing a genetic algorithm (DFO) using Unity's physics engine, we were able to effectively establish a high-reward achieving self-driving car algorithm (Fig. 1.1), which has the potential to achieve near-perfect accuracy in inclement terrain traversal and object detection and avoidance using attached proximity sensors.

Furthermore we were also able to develop an imitation learning based object avoidance and parking simulation using Unity's ML-Agents module, Anaconda and Tensorflow Python package, with moderately- high reward accumulation and low error rate for both obstacle avoidance (Fig. 6.2), but with low reward accumulation and moderately-high error rate for the parking actions (Fig. 6.3), with fluctuating object spawning as well as inhospitable parking scenarios.

While it is true that there has been a significant decline in road accidents in India (18.9% from 2019 to 2020 [7]), traffic accidents owing to human negligence account for 91.8% (354796 recorded cases) of said accidents.

With the algorithm intervention, not only can that number be effectively minimized, the scenarios that led up to the accidents can be simulated and can be used as a learning checkpoint or benchmark for the algorithm to grow and improve on the error caused by human negligence, for not only situations dealing with obstacle detection and avoidance (89% of cases including animal crossing cases) but parking (0.7% of cases owing to vehicles being parked at road shoulders) as well.

REFERENCES

[1] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V., 2017, "CARLA: an open urban driving simulator," *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16.

[2] Magdy, R., Rashad, S., Hany, S., Tarek, M., Hassan, M. A., and Mohammed, A., 2021, "Deep Reinforcement Learning Approach for Augmented Reality Games," *International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*.

[3] Zhang, Z., Akai-Nettey, S. M., Addo, A., Rogers, C., and Sinapov, J., 2021, "An Augmented Reality Platform for Introducing Reinforcement Learning to K-12 Students with Robots," arXiv:2110.04697 [cs.RO].

[4] Christian E López B, C. T., Omar M Ashour, 2019, "Reinforcement Learning Content Generation for Virtual Reality Applications," *ASME 2019 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*.

[5] Qin, R., Liu, D., Wu, J., and Zhu, F., 2019, "Reinforcement Learning Based Wireless Augmented Reality on Mobile Edge Computing," *IEEE International Conference on Industrial Internet (ICII)*.

[6] Venkata, V., Murali, R., Muvva, K. R., Adhikari, N., and Ghimire, A. D., 2017, "Towards training an agent in augmented reality world with reinforcement learning," *17th International Conference on Control, Automation and Systems (ICCAS)*.

[7] Bureau, N. C. R., 2020, "Causes-wise Number of Accidents due to 'Other Causes' during 2019- 2020 (All India) report and Cause – wise Distribution of Road Accidents and Unmanned Railway Crossing Accidents during 2020," Government of India.

[8] Awan, S., "Mountain race tracks," .

[9] 1Poly, "Low Poly Destructible 2 Cars no. 8," .

[10] Coders, R., "Unity3D How To: Driving With Wheel Colliders," `https://youtu.be/j6_SMdWeGFI`

[11] Valecillos, D., 2020, "How To Park A Car With Unity Machine Learning ML-Agents," `https://youtu.be/IcatcC9Rikk`

[12] Schuchmann, S., 2020, "Create your own A.I." `https://youtu.be/2Js4KiDwiyU`

[13] Peralta, J., Choi, A., and Habtegergesa, N., 2020, "Self-Driving Cars Using

Genetic Algorithms and Neural Networks," Master's thesis, University of Calgary Archives.

[14] YTech, 2019, "Reinforcement Learning Introduction," Medium.

[15] Jeffs, J., 2021, "Autonomous Cars, Robotaxis Sensors 2022-2042," IDTechEx.

[16] Boyd, R., 2017, "Implementing Reinforcement Learning in Unreal Engine 4 with Blueprint," Master's thesis, Tennessee State University.

[17] Osiński, B., Jakubowsk, A., Zięcina, P., Miłoś, P., Galias, C., and Homoceanu, S., 2020, "Simulation-Based Reinforcement Learning for Real-World Autonomous Driving," *2020 IEEE International Conference on Robotics and Automation (ICRA)*.

[18] Piarsa, I. N., Buana, P. W., and Mahasadhu, I. G. A., 2018, "Android Navigation Application with Location-BasedAugmented Reality," International Journal of Computer Science Issues, Volume 13, Issue 4.

[19] Narzt, W., Pomberger, G., and Ferscha, A., 2005, "Augmented reality navigation systems," Universal Access in the Information Society.

[20] Deshpande, V., 2019, "Unity Gaming and Augmented Reality Applications," Master's thesis, Shri Ramdeobaba Kamla Nehru Engineering College.

[21] Palinko, O., Kun, A. L., Cook, Z., Downey, A., Lecomte, A., Swanson, M., and Tomaszewski, T., 2013, "Towards Augmented Reality NavigationUsing Affordable Technology," *Proceedings of the 5th International Conference on Automotive User Interfaces andInteractive Vehicular Applications*.

[22] Dayan, P. and Watkins, C. J., 1992, "Machine learning," Q-learning.

[23] Nirmal Baby, B. G., 2019, "Implementing Artificial Intelligence Agent Within Connect 4 Using Unity3d and Machine Learning Concepts," International Journal of Recent Technology and Engineering (IJRTE).

[24] Majumder, A., 2021, *Deep Reinforcement Learning in Unity*, Apress.

[25] Brown, H., 2020, "Applying Imitation and Reinforcement Learning to Sparse Reward Environments," Master's thesis, University of Arkansas.

[26] Zifrid, R., 2019, "Smart Driving Agent based on DeepReinforcement Learning," Master's thesis, Freie Universität Berlin.

[27] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K., 2017, "Reinforcement Learning with Unsupervisedauxiliary Tasks," *5th International Conference on Learning Representations*.

[28] Fleury, N. and Nayrolles, M., 2018, "Better C++ using Machine Learning on Large Projects," *cppCON'18*.

[29] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., and Pérez, P., 2021, "Deep Reinforcement Learningfor Autonomous Driving: A Survey," arXiv:2002.00444v2 [cs.LG].

[30] Makarov, I., Kashin, A., and Korinevskaya, A., “Learning to Play Pong Video Game via Deep Reinforcement Learning,” Master’s thesis, School of Data Analysis and Artificial Intelligence.

[31] Serban, A., Poll, E., and Visser, J., 2020, “A Standard Driven Software Architecture for Fully Autonomous Vehicles,” Journal of Automotive Software Engineering.

APPENDIX A

CODE REPOSITORY AND LINKS

Link to the GitHub repository of the project:

```
https://github.com/ReanSchwarzer1/CS-Final-Project
```

Link to working video of the first module (Autonomous Driving):

```
https://youtu.be/hkzUavNsofs
```

Link to working video of the second module (Obstacle Avoidance):

```
https://youtu.be/1ywn9I_MHYs
```

Link to working video of the third module (Parking Simulation):

```
https://youtu.be/OKNEAgtciDo
```

Link to working video of the last module (AR Simulation):

```
https://youtu.be/n5nY7W0T2vw
```

APPENDIX B

PROOF FOR PLAGIARISM REPORT

AU-CARL: AUTONOMOUS VEHICULAR NAVIGATION WITH AUGMENTED REALITY AND REINFORCED LEARNING

ORIGINALITY REPORT

5%	1%	3%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Alex Serban, Erik Poll, Joost Visser. "A Standard Driven Software Architecture for Fully Autonomous Vehicles", Journal of Automotive Software Engineering, 2020 Publication	2%
2	Submitted to College of Haringey, Enfield and North East London Student Paper	<1%
3	Ramez Magdy, Seif Rashad, Seif Hany, Mahmoud Tarek, Mennat Allah Hassan, Ammar Mohammed. "Deep Reinforcement Learning Approach for Augmented Reality Games", 2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC), 2021 Publication	<1%
4	Submitted to Visvesvaraya Technological University, Belagavi Student Paper	<1%
5	Submitted to University of Lancaster Student Paper	

		<1%
6	Christian E. Lopez, Omar Ashour, Conrad S. Tucker. "Reinforcement Learning Content Generation for Virtual Reality Applications", Volume 1: 39th Computers and Information in Engineering Conference, 2019 Publication	<1%
7	Blazej Osinski, Adam Jakubowski, Pawel Ziecina, Piotr Milos, Christopher Galias, Silviu Homoceanu, Henryk Michalewski. "Simulation-Based Reinforcement Learning for Real-World Autonomous Driving", 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020 Publication	<1%
8	Submitted to CSU, San Jose State University Student Paper	<1%
9	Emilia Vassileva, Petko Mandjukov. "3. Trace Elements in the Environment", Walter de Gruyter GmbH, 2017 Publication	<1%
10	www.coursehero.com Internet Source	<1%
11	www.researchgate.net Internet Source	<1%

APPENDIX C

PLAGIARISM REPORT

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY (Deemed to be University u/s 3 of UGC Act, 1956)		
Office of Controller of Examinations		
REPORT FOR PLAGIARISM CHECK ON THE SYNOPSIS/THESIS/DISSERTATION/PROJECT REPORTS		
1	Name of the Candidate (IN BLOCK LETTERS)	Rohit Ramesh
2	Address of the Candidate	34, 80A, Brindavan Society, Majiwada, Thane - 400601 **Mobile Number :** 7824019446
3	Registration Number	RA1811003010055
4	Date of Birth	7 October 2000
5	Department	Computer Science and Engg.
6	Faculty	Mr. U.M. Prakash
7	Title of the Synopsis/ Thesis/ Dissertation/Project	Au-CARL: Autonomous vehicular navigation with Augmented Reality and Reinforced Learning
8	Name and address of the Supervisor / Guide	Dr. R.I. Minu SRM Nagar, Kattankulathur - 603 203 Chengalpattu District, Tamil Nadu. **Mail ID :** minur@srmist.edu.in **Mobile Number :** +91 94435 29372
9	Name and address of the Co-Supervisor / Co- Guide (if any)	N/A Address: N/A **Mail ID :** N/A **Mobile Number :** N/A
10	Software Used	Unity Engine
11	Date of Verfication	4 May 2022

12	Plagiarism Details: (to attach the final report)			
Chapter	Title of the Chapter	Percentage of similarity index (including self citation)	Percentage of similarity index (Excluding self citation)	% of plagarism after excluding Quotes, Bibliography, etc.,
1	Introduction	7%	0%	0%
2	Literature Review	0%	0%	0%
3	System Architecture and Design	6%	0%	0%
4	Methodology	2%	0%	0%
5	Coding and Testing	0%	0%	0%
6	Results and Observations	4%	0%	0%
7	Conclusion	0%	0%	0%
8				
9				
10				
Thesis abstract		0%	0%	0%
Appendices		0%	0%	0%
I / We declare that the above information have been verified and found true to the best of my / our knowledge.				
 Signature of the Candidate		SSignature of the Supervisor / Guide		
Signature of the Co-Supervisor/Co-Guide		Signature of the HOD		

SRM	**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY** **(Deemed to be University u/s 3 of UGC Act, 1956)**	
Office of Controller of Examinations		
REPORT FOR PLAGIARISM CHECK ON THE SYNOPSIS/THESIS/DISSERTATION/PROJECT REPORTS		
1	Name of the Candidate **(IN BLOCK LETTERS)**	Adhiraj Bhagawati
2	Address of the Candidate	House No. 41, Vidyamandir Rd., Bhetapara, Guwahati - 781028 **Mobile Number :** 6000587986
3	Registration Number	RA1811003010023
4	Date of Birth	19 December 2000
5	Department	Computer Science and Engg.
6	Faculty	Mr. U.M. Prakash
7	Title of the Synopsis/ Thesis/ Dissertation/Project	Au-CARL: Autonomous vehicular navigation with Augmented Reality and Reinforced Learning
8	Name and address of the Supervisor / Guide	Dr. R.I. Minu SRM Nagar, Kattankulathur - 603 203 Chengalpattu District, Tamil Nadu. **Mail ID :** minur@srmist.edu.in **Mobile Number :** +91 94435 29372
9	Name and address of the Co-Supervisor / Co- Guide (if any)	N/A Address: N/A **Mail ID :** N/A **Mobile Number :** N/A
10	Software Used	Unity Engine
11	Date of Verfication	4 May 2022

12	Plagiarism Details: (to attach the final report)			
Chapter	Title of the Chapter	Percentage of similarity index (including self citation)	Percentage of similarity index (Excluding self citation)	% of plagarism after excluding Quotes, Bibliography, etc.,
1	Introduction	7%	0%	0%
2	Literature Review	0%	0%	0%
3	System Architecture and Design	6%	0%	0%
4	Methodology	2%	0%	0%
5	Coding and Testing	0%	0%	0%
6	Results and Observations	4%	0%	0%
7	Conclusion	0%	0%	0%
8				
9				
10				
Thesis abstract		0%	0%	0%
Appendices		0%	0%	0%
I / We declare that the above information have been verified and found true to the best of my / our knowledge.				
 Signature of the Candidate		SSignature of the Supervisor / Guide		
Signature of the Co-Supervisor/Co-Guide		Signature of the HOD		

APPENDIX D

PROOF OF PAPER PUBLICATION

KI - Künstliche Intelligenz

Au-CARL: Autonomous vehicular navigation with Augmented Reality and Reinforced Learning

--Manuscript Draft--

Manuscript Number:	KUIN-D-22-00040
Full Title:	Au-CARL: Autonomous vehicular navigation with Augmented Reality and Reinforced Learning
Article Type:	Technical Contribution
Corresponding Author:	Minu R I SRM Institute of Science and Technology Kattankulathur, Tamil Nadu INDIA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	SRM Institute of Science and Technology
Corresponding Author's Secondary Institution:	
First Author:	Adhiraj Bhagawati
First Author Secondary Information:	
Order of Authors:	Adhiraj Bhagawati
	Rohit Ramesh
	Minu R I
Order of Authors Secondary Information:	
Funding Information:	
Abstract:	This paper aims to introduce a culmination of Reinforced Learning (RL) techniques used for simulations of self-learning cars and other agents or scenarios, and then compare the results of the simulation in Augmented Reality (AR). The main idea is to use Unity3D and C# to first create a rendition of an RL-learned self-learning car simulation, and then use Unity ML agents to simulate other aspects of cars and RL to diversify, and then create a user-friendly autonomous explorations system in AR for educating students in grade school.
Suggested Reviewers:	Sivabalan S siva.seennu@gmail.com He is working in this area of research
	SonyPriya s ssp.sonypriya@gmail.com This researcher is working in this area of research