INTERIM REPORT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING

Mixed Traffic Simulation for Autonomous Systems in Shared Spaces

Author:
Frederik Tobias Hillier (CID: 01412520)

Supervisors:
Prof Y.K. Demiris
Dr X. Zhang

Second Marker:
Prof D. Gunduz

Date: February 1, 2021

Contents

Chapter 1

Introduction

1.1 Report Outline

This is the interim report for my master's thesis titled "Mixed Traffic Simulation for Autonomous Systems in Shared Spaces". The report aims to present the initial background research and early implementation results [1]. The report will contain the following sections as suggested in the project guide:

The **Introduction** chapter will establish the project outline and the expected deliverables. The chapter will also describe the context of the underlying problem the project will try to resolve.

The **Background** chapter will first introduce the technical tools and theory needed to understand the project. This will include looking at what a game engine is, as well as briefly looking at the two major game engines used by most simulators. Thereafter comes an analysis of the simulators that have been studied with a conclusion for which simulator was chosen for this project.

The **Implementation** chapter will describe what work has been done in regards to the chosen simulator. The chapter will also contain an evaluation plan for how the project deliverables will be evaluated. In addition, the chapter will show the estimated timeline and possible extensions.

And finally the **Ethical, Legal and Safety Considerations** chapter which will look if there are any ethical legal or safety concerns both in regards to the project as is, but also briefly with potential extensions.

1.2 Project Specification

1.2.1 Objectives

The objectives of the project are:

- Survey the current state of the art in vehicle and mobile robot simulation and evaluate current simulators (e.g. CARLA, CrowdSim3D, Gazebo), list and understand the underlying simulation methodologies (e.g. types of physics engine), and understand and document their advantages and disadvantages, particularly in terms of their sensing and control abilities.
- Select the most suitable current simulator that offers potential for incorporating mixed traffic (for example, cars, mobile robots/wheelchairs, humans, cyclists, and so on), and 3D environments/maps.
- Create a process for incorporating and controlling agent models with an API (e.g. ability to add control abilities to the human model, for example to move their bodies, and to grasp objects).
- Add sensors to the simulated agents (e.g. cameras, LIDARS) and a perception API, where simulation users can obtain sensing data from the simulator, for example what can the agents "see" in their environment.

(Source: Project Specification as written by Prof. Y. K. Demiris)

Features that want to have in our simulator:

- Be able to place agents
- Control the agents' behavior
- Access to what the agent sees as well as body movement
- Looking to plot a path for pedestrians
- Able to drive off the road, e.g. on the pavement and interact with pedestrians
- Want to easily create custom maps which are not whole worlds
- Does not need to handle a large number of pedestrians and vehicles
- Not looking to be able to navigate indoors
- Animation and realistic car movement is not important

(Source: After discussion in the supervisor meetings, 22/10 and 10/11)

1.2.2 Deliverables

The main deliverables for this project will be:

- An analysis of available simulators with a conclusion for which one to use for the project.
- A simulator with the ability to easily add additional models and simulate mixed traffic.
- A simulator with the incorporated sensing and controlling APIs.
- Documentation for how to use the two above points.

If time allows for it there could be a possibility of extending the project, but that is still to be determined (Section 3.4)

1.3 Context

For the last decade there has been a lot of talk about self-driving cars and other autonomous systems [2]. Back in 2010 most autonomous systems were only driven on closed circuits, but today there are a variety of car manufacturers offering autopilot on their cars. In 2021 Tesla have announced that all new cars they produce will have the hardware needed for full automation in almost all circumstances [3].

Compared to 2010 when the autonomous cars were mainly driven on closed tracks, the modern-day self-driving car will have to operate alongside other cars on the road as well as alongside pedestrians. This is where the concept of shared spaces comes in. The autonomous system must now take other entities into account when making decisions.

As other entities' actions can be complex to forecast, machine learning models can be used to predict the outcome of a situation more easily. These models can be trained much faster on a simulator than in real life. This is because the simulator can simulate many different configurations at once. An autonomous system in real life cannot replicate the exact same environment, whilst this can easily be done in a simulation.

Another advantage of using a simulator is that you can learn from doing mistakes, whilst in real life, mistakes can be both dangerous and costly. In the simulator, the autonomous system can learn without the risk of colliding with an actual person or damaging other cars. Running simulations is therefore a good way to train the machine learning model much faster and safer than on a real road.

The autonomous system does not have to be a vehicle, but anything that can artificial control itself. These could for example be mobile robots/wheelchairs or artificial controlled humans and cyclists.

This project will consist of three main parts. The first one being to research available simulators (Section 2.2). This will be necessary to get a good starting point for the project. Working on an existing simulator that is too complex would mean that it would take too long to get basic features working. Whilst working on a simulator that does not have enough features will mean that too much time will be spent implementing something that another simulator will already have implemented. Exactly what these features are will be clarified in the Project Specification chapter (Chapter 1.2). The second part will be to implement the missing features from the simulator that we selected to work on. And the last part will be to determine what to do next with the simulator after the missing features have been implemented. This is still open-ended, but some of the potential use cases will be discussed in the Implementation section (Section 3.2).

The product generated from this project could be used for a variety of different use cases. Firstly it could be used to test the software for an autonomous system. Testing it in a simulator would be much faster than trying to check for corner cases in real life. Another example would be to train a machine learning model. The simulator will aim to be a diverse tool to accommodate a variety of needs.

Chapter 2

Background

2.1 Background Theory

2.1.1 Physics Engines

The purpose of a physics engine is to allow programs to easily implement physics, without the creators having to implement their own from scratch every time. An example could be, if a program needed to calculate a trajectory, this formula could be implemented directly inside the physics engine [4]. This is quite a simple example, but simulating basic particle distribution or fluid dynamics would be much more complex.

There are a large variety of different game engines, but the ones that are most commonly used in the simulators listed below are PhysX and ODE.

Open Dynamics Engine (ODE)

Open Dynamics Engine is a free physics engine most commonly used in simulation projects [5]. ODE is written in C++ and is a popular choice for simulating robotics.

PhysX

PhysX is an open source-physics engine developed by Nvidia. This physics engine is commonly used in a lot of computer games. Both Unity and Unreal Engine use PhysX (Section 2.1.2). PhysX is quite unique among other physics engines. This is because most scientific-targeted simulation software would be more accurate. This physics engine however truncates the calculations leading to more efficient but less accurate simulations [6]. The physics engine is also non-deterministic, meaning there could be some variation between runs.

2.1.2 Game Engines

The purpose of a game engine is to work as a fundamental back-end for most video games and simulators. A game engine is a software tool that allows interactive dig-

ital content to be made using a framework that easily allows the user to run it on different platforms such as computers, smartphones and consoles [7]. Game engines are complex and consist of many components such as a physics engine, a rendering engine, and other interactive tools to help the creator. They give the creator the freedom to control everything from lighting and audio to animation and character behavior in their game.

There exists many different game engines, but the two main ones that are currently in existence are Unity and Unreal Engine. These allow beginners to easily create their first games, as well as professional companies creating their games for millions of users [8, 9].

Unity

Unity is currently the most commonly used game engine with over 2.5 million registered developers [10]. Unity is developed by Unity Technologies and was first released in June 2005. Over 60% of augmented- and virtual reality games are created using Unity [7].

Unity is free for personal and educational projects. To create the game, Unity allows a lot of features to be used through the GUI, but for full freedom, the creator would need to program in C#. Unity however provides lots of resources to learn C#. Unity also combines with Visual Studio where the IntelliSense can help with recommending functions to use as well as spot errors[1].

Unreal Engine

Unreal Engine is a fast-growing game engine created by Epic Games [11]. The first version of Unreal Engine was released in 1998, but Unreal Engine 4 (UE4), which was released in September 2014, is their latest game engine. They have however announced that they will release Unreal Engine 5 (UE5) at the end of 2021 [12].

UE4 has a variety of different use cases as well as for games. TV broadcasters such as Sky Sports and BBC are using Unreal Engine for their graphical match analysis [13]. Professional architects are also using the product to illustrate their creations [14].

Unreal Engine 4 is free for personal use and businesses making less than $1 million in gross revenue [12]. The code for UE4 is available on GitHub[2] (For access to the repository you would need to sign up for an Epic Games account[3]). UE4 allows two different ways for developers to program their games. Either by using C++,

[1] `https://docs.microsoft.com/en-us/visualstudio/gamedev/unity/get-started/getting-started-with-visual-studio-tools-for-unity`

[2] `https://github.com/EpicGames/UnrealEngine`

[3] `https://www.unrealengine.com/en-US/ue4-on-github`

which links with Visual Studio[4], or by using blueprints as illustrated in Figure 2.1. Blueprints allow the creator most of the freedom code gives them, but in an easier drag and drop layout.

The majority of the simulators that will be looked at later use Unreal Engine as their game engine.

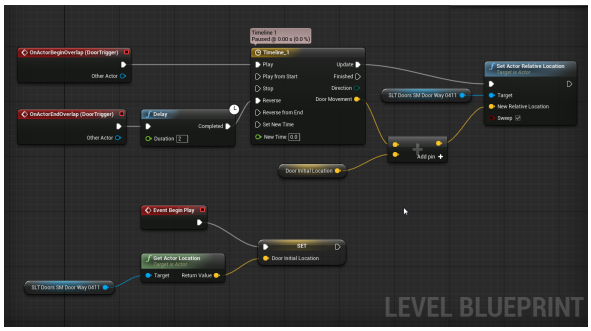

Figure 2.1: Source: `https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Timelines/Examples/OpeningDoors/index.html`

2.1.3 Levels of Automation

When talking about autonomous systems, it is important to know the different levels of autonomy. The modern day Tesla cars are level 3, which means they can be driven autonomously for the majority of the time, but do need a human driver as a fallback system [3, 15].

Level	Name	Description
0	No Automation	The driver is solely controlling the vehicle
1	Driver Assistance	Cruise control is an example of this
2	Partial Automation	Automatic parking
3	Conditional Automation	Modern day Tesla cars
4	High Automation	Vehicle almost fully automated
5	Full Automation	Vehicle never needs human input

Table 2.1: Source: `https://link.springer.com/article/10.1007/s40534-016-0117-3#Sec3`

For this project all our simulations will be level 5.

2.1.4 Mixed Traffic Simulation

The purpose of a mixed traffic simulation is to model the interaction between different types of agents, for example the the interaction between pedestrians, au-

[4] `https://docs.unrealengine.com/en-US/ProductionPipelines/DevelopmentSetup/VisualStudioSetup/index.html`

tonomous robots and vehicles. This is important because having a fully automated agent has to interact with a lot more than only other fully automated agents [16].

A mixed traffic simulator can be used to simulate different situations where an autonomous system has to react to other agents behavior. This will mean the implemented algorithm will have the opportunity to try out a lot more complex scenarios a lot faster than in real life. This will help speed up development time.

2.1.5 API

API stands for Application Programming Interface. This will allow a program to communicate with one-another. The API will define what kind of requests another program can make, how to make them, and the what the format should be. APIs also need to be documented so that other developers know how they work [17].

In the case of our simulators, APIs will be used to communicate between an external program and the simulator itself. These communications could contain information for how to control the agent, or it could contain information from what the agent observes.

2.2 Analysis of Relevant Literature

In this section we will look at a large variety of different simulators, to determine which one best suits our purpose. We will be looking at which operating system and game engine the simulator uses, whether or not it is open source, and the pros and cons of each simulator. We will be particularly looking at the simulators sensing abilities, ability to add additional entities, map customisability, available APIs, and how user-friendly the simulator is. Aspects of the simulator which is not as important as how realistic the simulator physics is, and how visually good looking it is. These are criteria formed by the project specification (Section 1.2)

The purpose of this section is to get a good understanding of the different simulators currently in existence. The aim is to find 3-4 simulators that would be worth looking closer at.

2.2.1 4DV-Sim

Description: 4DV-Sim[5] is a simulator that is designed to emulate the hardware and sensors in autonomous systems. This is a professional product and has a variety of use cases from simulating farming to military equipment [18].
Open Source: No
Operating System: Linux
Game Engine: Non, but it does use PhysX for the physics engine

[5]Website: `https://www.4d-virtualiz.com/en/automotive-simulator`

Pros: The simulator has a lot of available APIs. The simulator also comes with a configurable GUI to set up the simulation environment how you would like it. Also, as it is professionally made, it looks very good.
Cons: It is not designed to train machine learning implementations on the simulator, but rather emulate a current hardware setup. Also, as it is not open source, it will not be something that we could modify or expand upon to suit our purposes.
Conclusion: As 4DV-Sim is not an open-source product it is not something that we can use for this project. It is however interesting to see that simulators like this are needed not just for research purposes, but for customers who want to try out their hardware setup in an emulated environment.

Figure 2.2: Source: `https://www.4d-virtualiz.com/en/automotive-simulator`

2.2.2 AirSim

Description: AirSim[6] is a simulator for cars and drones. It is open-source and works as a plugin for Unreal Engine, which means the simulator can be used with any environment which has been modeled inside the game engine. According to their website [19], the goal of the simulator is to create a platform for AI research to experiment with deep learning, computer vision, and reinforcement learning algorithms for autonomous systems.
Open Source: Yes
Operating System: Any operating system
Game Engine: Primarily Unreal Engine, but it also offers a prototype version in Unity
Pros: Offers a large range of existing APIs. The simulator also has an active community on both discord and Github. It also gives the option to add drones. It is also designed to train machine learning models on it.
Cons: The simulator is not as realistic as other simulators. The vehicle physics is not as good as some of the other simulators, for example the handling and collisions. Also, currently, there are no pedestrians in the game.
Conclusion: AirSim is worth looking closer into. As it is built using a game engine it should not be too hard to add the missing features, like for example adding and

[6]Website: `https://microsoft.github.io/AirSim`

controlling pedestrians. In addition, as it is a plugin for Unreal Engine means that we can use other tools to import for example maps. Also, realistic vehicle physics was determined not to be an important factor for this project.

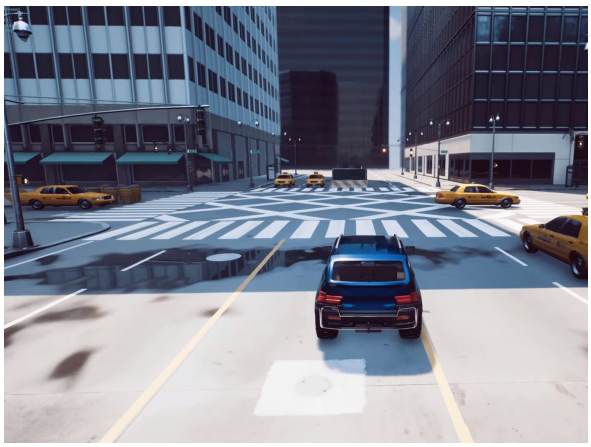

Figure 2.3: Source: `https://microsoft.github.io/AirSim`

2.2.3 Apollo

Description: Apollo[7] is a simulator that is designed to emulate the hardware in autonomous vehicles so that it can be trained for machine learning models. According to their website [20], Apollo is a flexible architecture that accelerates the development and testing of autonomous vehicles.
Open Source: Yes
Operating System: Any system that can run Docker
Game Engine: Unity
Pros: Accurately models the vehicle physics to help improve the machine learning model's accuracy. The simulator is also actively being worked on by a large community.
Cons: It looks like quite a complex simulator, and it does therefore not seem like it will be easy to modify. The product is really specific towards training autonomous vehicles.
Conclusion: Due to the complexity of this simulator, it does not look like something that we could build upon for this project.

[7] `https://github.com/ApolloAuto/apollo`

Figure 2.4: Source: Slide deck from Apollo Game Engine Based Simulation Talk at GDC 2019 - `https://bit.ly/2VSzwlF`

2.2.4 Autoware

Description: Autoware[8] is an open-source software for autonomous vehicles. It comes with a large variety of APIs [21]. Autoware is however not a simulator, but can be used on a simulated vehicle to make it autonomous.
Open Source: Yes
Operating System: Robot Operating System (ROS)
Game Engine: Na
Pros: As it runs on ROS it can easily be adapted to work on a real autonomous vehicle.
Cons: Currently it only works with a specific car model and sensor set up. The software also seems quite complex, and combining it with a simulator will probably be quite challenging.
Conclusion: As this is not a simulator this is not something that we can use for this project. We will see with the LGSVL simulator (2.2.11), this software can be used alongside a simulator to model the autonomous system.

2.2.5 Carla

Description: Carla[9] is an open-source simulator for developing autonomous vehicles. It contains a variety of APIs and is actively being developed. Carla is also designed for training machine learning models [22].
Open Source: Yes
Operating System: Primarily Linux, but also Windows
Game Engine: Unreal Engine
Pros: Has a lot of features already implemented, such as sensors, vehicle API, and the ability to add new objects. The simulator also has the ability to add pedestrians

[8] `https://gitlab.com/autowarefoundation/autoware.auto/AutowareAuto`
[9] `https://carla.org/`

and plot their movement. Active community. Well documented and lots of information online.
Cons: Difficult to add new and custom maps. Vehicle handling is not as realistic as some of the other simulators.
Conclusion: Carla is worth looking into further as it has most of the features that we are looking for.

Figure 2.5: Source: `https://www.unrealengine.com/en-US/spotlights/carla-democratizes-autonomous-vehicle-r-d-with-free-open-source-simulator`

2.2.6 CrowdSim3D

Description: CrowdSim3D[10] is primarily a simulator for modeling large crowds, but can also be used for vehicle traffic.
Open Source: No (£180)
Operating System: Any operating system
Game Engine: Not specified
Pros: Has the ability to control several pedestrians and vehicles in a shared space.
Cons: Does not look to be designed for machine learning models. Difficult to add new models. Also, as it is not open source it will not be possible to customise the product.
Conclusion: CrowdSim3D is not worth considering as we cannot adapt the product as it is not open source.

[10] `https://crowdsim3d.com`

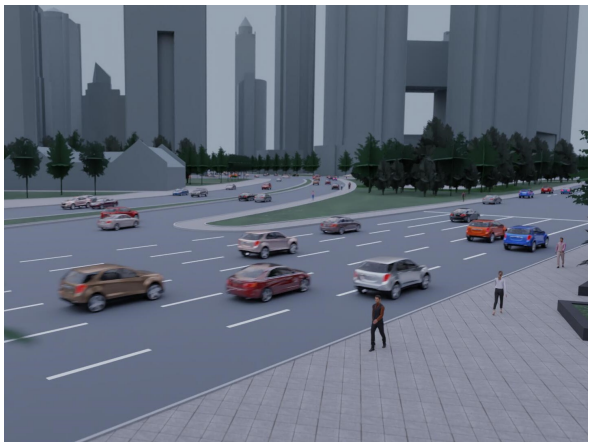

Figure 2.6: Source: `https://crowdsim3d.com`

2.2.7 Deep Drive

Description: Deep Drive[11] is an open-source simulator aimed to train neural networks for self-driving cars [23]. They also provide you with a large data set to train your autonomous vehicle on.
Open Source: yes
Operating System: Any operating system
Game Engine: Unreal Engine
Pros: Is able to handle a variety of different sensors and vehicle setups. It also has an active community and a leader board where you can compare your trained neural network against other developers.
Cons: Currently there are three maps available, but it is not easy to add your own maps. Also, it is not designed to be customised. Currently, there are no pedestrians and no APIs.
Conclusion: Deep Drive is not a simulator that is worth looking at as it is not designed to be customised. It also lacks most of the features we are looking for.

Figure 2.7: Source: `https://deepdrive.voyage.auto`

[11] `https://deepdrive.voyage.auto/`

2.2.8 Donkey Car Simulator

Description: Donkey Car Simulator[12] is a simulator for the Donkey Car. The car itself costs roughly £200 and can be ordered online, or you can download the schematics for free. The simulator can be used to train a neural network in Python which can then run on your car.
Open Source: Yes
Operating System: Any operating system
Game Engine: Unity
Pros: Easy to use
Cons: Not what we are looking for as it is only used to train a real toy car.
Conclusion: Donkey Car is an interesting project to read about as it is a fun kit that introduces people to autonomous cars. However, it is not what we are looking for for this project as the simulator is not customisable and there is only one vehicle.

Figure 2.8: Source: `https://docs.donkeycar.com`

2.2.9 Gazebo

Description: Gazebo[13] is a robotics simulator designed to design robots, test algorithms and train AI systems [24]. Gazebo offers the ability to simulate several robots at once in both indoor and outdoor environments.
Open Source: Yes
Operating System: Any operating system
Game Engine: ODE, but also Bullet, Simbody and DART which are additional physics engines.
Pros: Very versatile, can be used for much more than traffic simulations.
Cons: Due to the large number of different physics engines it might be complicated to fix something if a feature breaks. The documentations is not as clear as some other simulators.
Conclusion: Gazebo is worth looking at as it is widely used for robotics simulations. All the sensing APIs are already implemented.

[12] `https://docs.donkeycar.com/guide/simulator`
[13] `http://gazebosim.org`

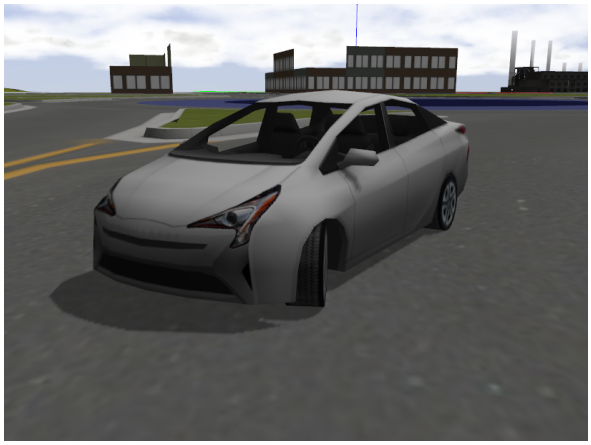

Figure 2.9: Source: `http://gazebosim.org/blog/vehicle%20simulation`

2.2.10 LPZRobots

Description: LPZRobots[14] is a robotics simulator created by the Robotics Group for Self-Organisation of Control at the University of Leipzig in Germany [25, 26]. The simulator aims to have an open environment to simulate the robot's physics.
Open Source: Yes
Operating System: Primarily Linux, but now also supports Windows
Game Engine: ODE
Pros: Free to add any kind of robot.
Cons: The code has not been updated since 2018 and the last release was in 2016. Unlike other simulators, this one does not seem to have an active community.
Conclusion: As the development of the LPZRobots simulator has been inactive for several years, makes it not worth considering. The simulator also lacks most of the features we are looking for.

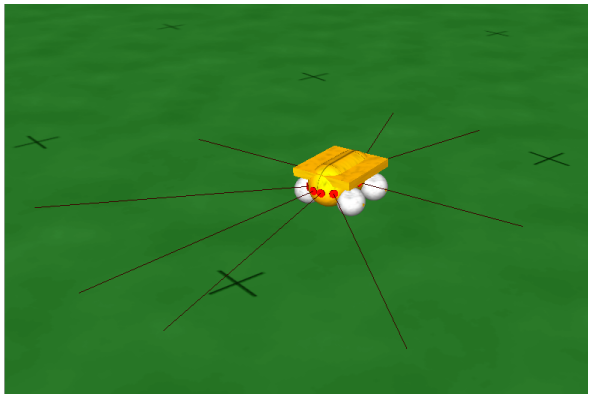

Figure 2.10: Source: `https://itp.uni-frankfurt.de/~gros/StudentProjects/Robots_2016_ObstacleAvoidance`

[14] `https://github.com/georgmartius/lpzrobots`

2.2.11 LGSVL Simulator

Description: LGSVL[15] is a simulator created by the Advanced Platform Lab at the LG Electronics America R&D Center [27]. The simulator combines the vehicle from Autoware (Section 2.2.4) with Apollo (Section 2.2.3) which emulates the hardware. The LGSVL has lots of available APIs such as cameras, LiDAR, RADAR, and GPS. Some environmental parameters can also be changed such as the map, weather, and pedestrians.
Open Source: Yes
Operating System: Windows 10
Game Engine: A variety due to its complexness, but both Unreal Engine and Unity.
Pros: Has most of the features we are looking for.
Cons: Relies on a lot of different components. The codebase is therefore large and complex and it looks difficult to add our own features such as our own entity.
Conclusion: The LGSVL Simulator would probably not work for us as the code-base seems too large and complex.

Figure 2.11: Source: `https://www.lgsvlsimulator.com/about`

2.2.12 Marilou

Description: Marilou[16] is a simulator created by ANYKODE [28]. Marilou is an open map simulator where the user can add objects and hindrances for the robot to navigate around. The simulator is designed to simulate simultaneous localisation and mapping (SLAM) and other localisation techniques.
Open Source: No (£350)
Operating System: Windows and Linux
Game Engine: Unknown
Pros: Accurately simulates sensors and robot behavior. Easy to add new objects and other controllable entities.
Cons: As the simulator is not open source, we cannot modify its behavior. Latest release 2018.

[15] `https://github.com/lgsvl/simulator`
[16] `http://www.anykode.com/index.php`

Conclusion: Marilou is not a simulator that we can use for this project as it is not open source. In addition, it is not designed with APIs in mind and controlling multiple objects at once looks impossible in the current program.

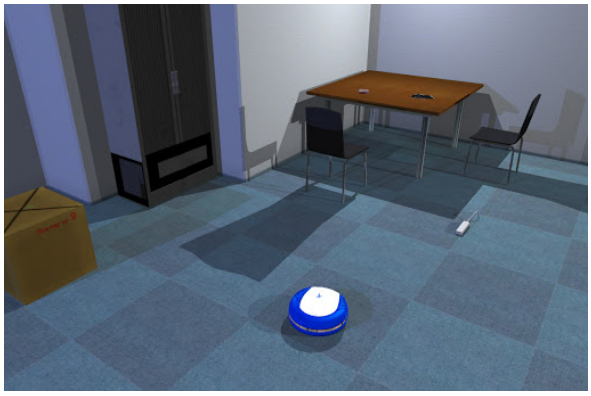

Figure 2.12: Source: `http://www.anykode.com/downloads.php`

2.2.13 rFpro

Description: rFpro[17] is a driving simulation software which focuses on road-vehicle simulation [29]. rFpro allows for a variety of use cases, from training machine learning models for autonomous driving [30], to motor racing.
Open Source: No
Operating System: Windows
Game Engine: ISIMotor - A game engine created by Image Space Inc [31]. The game engine is used for F1 and other racing games.
Pros: A professionally made simulator that has most of the features we are looking for. Very good graphics and accurate vehicle behavior.
Cons: rFpro does not give us access to any of the source code. The price is only available upon request, but it is most likely too expensive for this project.
Conclusion: Even though rFpro contains just about all of the features we are looking for, and is probably the best-looking simulator, it will not work for this project as it is not open source.

[17] `https://www.rfpro.com`

Figure 2.13: Source: `http:https://www.rfpro.com/driving-simulation`

2.2.14 Rigs of Rods

Description: Rigs of Rods[18] (RoR) is an open-source physics simulator primarily designed to simulate vehicle physics. The simulator uses soft-body physics which means that if the vehicle collides, its structure will be deformed. This will result in a more accurate simulation.
Open Source: Yes
Operating System: Windows and Linux
Game Engine: Non, creates its own soft-body physics engine.
Pros: There is an active community creating modifications for the simulator. Also, the only simulator on the list which uses soft-body physics.
Cons: There are no APIs currently available.
Conclusion: As for this project, we are not looking for realistic features, but rather certain APIs. This simulator will require too much work to add the missing features. It is therefore not a simulator that is worth considering.

Figure 2.14: Source: `https://docs.rigsofrods.org/gameplay/beginners-guide`

[18] `https://www.rigsofrods.org`

2.2.15 TORCS - The Open Racing Car Simulator

Description: TORCS[19] is an open-source racing car simulator. It can be used as an ordinary racing game or as an AI racing research platform. The creators of TORCS used to host competitions on its website among players for who could create the best artificially intelligent racing car [32].
Open Source: Yes
Operating System: Linux and Windows
Game Engine: Non, implemented from scratch.
Pros: Easy to add and create new content.
Cons: Latest release 2016 but mainly developed in 2008. Also, has no available APIs and no pedestrians.
Conclusion: This simulator is lacking most of the features we are looking for. It is also quite outdated. TORCS is therefore not worth considering.

Figure 2.15: Source: `https://sourceforge.net/projects/torcs`

2.2.16 Webots

Description: Webots[20] is an open-source robotics simulator developed by Cyberbotics [33]. Webots provides a complete developing environment to model, program and simulate the robots [34].
Open Source: Yes
Operating System: Any operating system
Game Engine: ODE
Pros: A lot of documentation on how to add new features. Seems to be able to generate new maps easily. Available chat page.
Cons: There does not seem to be many APIs to control multiple entities. All the APIs are mainly used for sensing.
Conclusion: Webots has most of the features that we are looking for, however seems to lack the ability to easily control multiple entities at once. The simulator also seems less advanced than Gazebo (Section 2.2.9), so as an open area robotics simulator, we will rather look at that one first. We will therefore not be considering Webots.

[19] `https://sourceforge.net/projects/torcs`
[20] `https://www.cyberbotics.com`

Figure 2.16: Source: `https://www.cyberbotics.com/doc/automobile/city-night`

2.2.17 Conclusion

After looking at the different simulators we decided to look further into AirSim (Section 2.2.2), Carla (Section 2.2.5) and Gazebo (Section 2.2.9). These are the simulators that will hopefully be the easiest to adapt and modify to suit our needs. In the next section, Analysis of Competing Product (Section 2.3), we will look at which of these three simulators to go for.

2.3 Analysis of Competing Products

In this section, we will look at which of the three simulators to use from Section 2.2.

2.3.1 AirSim vs Carla vs Gazebo

For this section, I started off building the different simulators from source. The aim was to build the simulators in Ubuntu, but due to my outdated graphics card, I was unable to build Unreal Engine. I was however able to run Unreal Engine on Windows by downloading the binary distribution.

AirSim: The AirSim plugin built successfully on Windows[21], and I was able to use it in Unreal Engine. The only minor inconvenience was that the simulator has to be built using the Visual Studio 2019 development terminal. However, after installing Visual Studio 2017 which is used to build Carla, I was no longer able to build AirSim.

As AirSim is built using a game engine, it would hopefully mean it is an easier environment to set up missing features. AirSim is also designed to simulate traffic, unlike Gazebo which is designed for a variety of simulations.

[21] `https://microsoft.github.io/AirSim/build_windows`

Carla: Carla built successfully on Ubuntu following this guide[22] and after doing these alterations:

- As I was using Ubuntu 20.04 I had to install Python2/Pip2 using curl
- Clang-8 was outdated so I installed Clang
- Clang-Tools-8 was outdated so I installed Clang-Tools
- lld-8 was outdated so I installed lld

However, as Unreal Engine did not build on Ubuntu I was unable to proceed from here and instead tried on Windows. Carla claims to work on Windows, but I ran into a build error which I was unable to resolve. This seems to be a known issue, and as of the time of writing, is still an unresolved issue on GitHub[23].

Another issue with Carla was the ability to import custom maps [35]. Carla requires the map to consist of two layers. The first one being the map structure with buildings and roads, whilst the second one will consist of road rules, such as traffic lights, where the cars are allowed to drive, pedestrian crossings, and so on. RoadRunner[24] can be used to import maps, but this is not a free product.

Gazebo: Gazebo built successfully on Ubuntu[25].

Gazebo has all the sensing features we are looking for [36], such as GPS, LiDAR, RADAR, and Ultrasonic. One main drawback with Gazebo is the lack of existing APIs to interact with multiple entities at once. Even though the Gazebo platform has a lot of features, it is not as rich as Unreal Engine [37].

Conclusion: As we are not able to build Carla, as well as it being very difficult to change maps, we will discard that one and look at comparing AirSim and Gazebo.

- AirSim and Unreal Engine is more feature-rich than Gazebo. Using Unreal Engine will also minimise chances of finding out something is not possible later on.
- Both have the ability to import maps quite easily
- Both have the ability to train machine learning models.
- AirSim has GPS and Lidar, but it is not clear if it has Ultrasonic or not. Gazebo has more sensing APIs in any case.
- Having multiple entities seems easier in AirSim than in Gazebo.

[22] `https://carla.readthedocs.io/en/latest/build_linux`
[23] `https://github.com/carla-simulator/carla/issues/3605`
[24] `https://uk.mathworks.com/products/roadrunner.html`
[25] `http://gazebosim.org/tutorials?tut=install_ubuntu&cat=install`

- AirSim is designed for vehicles, whilst Gazebo is designed to be a more generic robotics platform.
- Gazebo is more commonly used than AirSim.

After looking at the information above we have chosen to go with AirSim for this project. This is because AirSim is a plugin for Unreal Engine and that gives us the option to extend the program to anything Unreal Engine allows us to do.

Chapter 3

Implementation

3.1 Current Work

This section will describe what implementation work has been done thus far, and what work will be done in the near future.

3.1.1 AirSim

AirSim is missing pedestrians, and this has been the first and main priority for the time being. The plan is to add pedestrians in such a way that it will be easy to add cyclists and other mobile robots soon after that. As can be seen in the timeline (Section 3.3) this is expected to take roughly 3 weeks. This is also due to the fact that other commitments have been pushed back slightly due to this Interim Report. In AirSim currently we have got a character added, but the character does not yet have any ability to be spawned in or controlled.

AirSim has a lot of available APIs, and the hope is to either incorporate them, or follow their specification when implementing our own. AirSim has a lot of documentation to help with this. There is also an active community where developers can go and ask questions.

AirSim is built using both Unreal Engine and also Unity. For the time being the plan is to only use Unreal Engine. This is to prioritise rapid development rather than cross game engine compatibility.

An advantage of using AirSim is that it could later allow us to use drones in the simulation. This is however not something that will be looked at yet and the feature will be ignored for the time being.

3.1.2 StreetMap

As mentioned before, the main advantage of using AirSim is that it works as a plugin. This means that we can load any map into Unreal Engine then drive around it.

This is what StreetMap allows us to do. Figure 3.1 shows the area around the South Kensington Campus loaded into Unreal Engine with the AirSim plugin also added. The roads are coloured green to make them easily distinguishable from the ground and the buildings.

Currently I am looking at ways of updating and improving certain parts of the map. As can be seen in Figure 3.1, Exhibition road and the Royal Albert Hall are missing. The vehicle can drive off the road, so it is not very important to have that part working however.

Another thing that has to be fixed is that currently buildings do not have a collision box. This means that cars can drive straight through them. This should however hopefully be an easy fix as the buildings are showing up on the existing vehicle sensing APIs in AirSim.

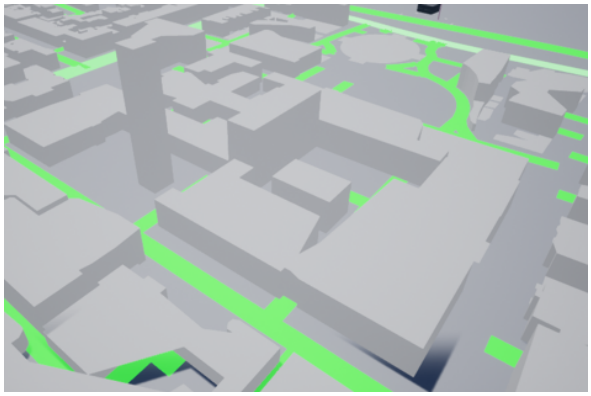

Figure 3.1: Source: Loaded the map of the South Kensington Campus into Unreal Engine

3.2 Future Work

3.2.1 Development

After pedestrians have been added we will need to look at implementing APIs to control them. This should try to follow a similar style to the APIs currently available to control the vehicle. Thereafter we could make minor modifications to the pedestrians so that other mobile robots could be controlled as well.

The next step would then be to implement the sensing APIs for our new entities. First and foremost, a camera illustrating the pedestrians' point of view would be the most important one. And once this is done extending it so that it would work for mobile robots as well. Following that, we would like to have other kinds of sensors for the mobile robots working. These could for example be ultrasonic and LiDAR.

When this is done, we will have most of the features we are looking for in the

simulator. This will hopefully be done sometime in the middle of March.

3.2.2 Evaluation Plan

When evaluating the simulator there will be some key features to look at:

The main one will be **usability**. It is important that it is easy to use the simulator. When evaluating the usability we will look at:

- How to set up the simulator.
- How easy it is to import maps.
- How to add agents into the simulation.
- How to configure these agents.
- How to control these agents.
- Are the APIs easy to use?

Documentation should be written to make it clearer in regards to the elements listed above. It could also be useful to ask someone else to try using these features to evaluate if the documentation is clear and intuitive.

The simulator could also be evaluated on **feature richness**:

- How much of the objectives (Section 1.2.1) have been implemented.
- Are there any other needed features that have not been implemented.
- Is there a variety of available agents, such as pedestrians and autonomous robots?

Finally, it is also worth evaluating the **performance**:

- Memory usage
- Frame rate
- Responsiveness

Some simulators had a clear impact on performance when several agents were spawned at once. It is important to make sure that the simulator is still usable for a small number of entities.

3.3 Future Timeline

When looking at the timeline, it is worth noting that future work might be pushed back if the AirSim development takes longer than anticipated.

In addition, after the last exam at the end of march, a lot more time will be spent on this project.

3.4 Extensions

The extensions for the project have not yet been defined, but there are a variety of different options. The current plan is to implement the features required in AirSim and then use them in an extension task. Listed below are some of the suggestions which have been discussed in the supervisor meetings.

One option would be to try to use the simulator alongside another final year project to simulate the live feed from traffic cameras. There are several use cases for this. Firstly, it could detect dangerous driving and report it to the police. This could both make people drive more safely as they know they are being surveyed, as well as making it possible to stop dangerous driving early on. It could also be used to track dangerous traffic junctions to monitor and detect near-collisions.

Another option could be to simulate an autonomous wheelchair which exists in the Imperial Robotics Lab. By modeling the wheelchair in AirSim we could try to create an autonomous system and then compare it to the behavior in real life.

A third example could be to try to train a machine learning model for an autonomous robot so that it could navigate around crowded places with cars and pedestrians [38].

Chapter 4

Ethical, Legal and Safety Considerations

4.1 Ethical Considerations

As this is just a simulation, there are not really any ethical concerns as such.

If we however decide to use the knowledge we learn in our simulator on a physical system there are a few things worth thinking about. Ethical considerations for autonomous systems have been a discussion for a long time [39, 40]. There are a variety of different ethical concerns. To name just a few, will an autonomous system be responsible enough [40], who would be responsible for an accident involving a self-driving car [41, 42], and how could autonomous vehicles impact peoples behavior [43]. These are not ethical concerns for the project as is, but could become an issue once we decide what the plan for the future is (Section 3.2).

4.2 Legal Considerations

AirSim has an MIT license which means that we can use the simulator however we like [44]. This is the same license that is used for StreetMap. In regards to the game engine, as we have chosen to use AirSim which uses Unreal Engine, we are free to distribute the simulator as long as we do not make a gross profit of more than $1 million [12].

It could also be worth considering what the UK rules for autonomous vehicles are[1], if in the future work we decide to introduce a physical system [45, 46].

[1] https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/929352/innovation-is-great-connected-and-automated-vehicles-booklet.pdf

4.3 Safety Considerations

In regards to the project itself, there are no safety issues as it is a simulator being worked on from home.

As mentioned in the other two sections. If information from the simulator ever gets used in a physical product there are a few things worth considering. Firstly, it is important to remember these are only simulations. Training a machine learning model on the simulator will not accurately reflect the behavior in real life. Secondly, it is important when testing a physical system that it does not for example collide with someone. This could be prevented by driving slowly or not driving where other people or objects might be in the way.

Bibliography

[1] T. Clarke. (2020, Oct) Project guide. [Online]. Available: https://intranet.ee.ic.ac.uk/t.clarke/projects/general/Project%20Guide.pdf

[2] J. Markoff, "Google cars drive themselves, in traffic," Oct 2010. [Online]. Available: https://www.nytimes.com/2010/10/10/science/10google.html

[3] Tesla. (2021) Autopilot. [Online]. Available: https://www.tesla.com/autopilot

[4] I. Millington, **Game physics engine development**. CRC Press, 2007. [Online]. Available: https://books.google.no/books?id=d0NZDwAAQBAJ&lpg=PP1&ots=2LlCAMdy58&dq=Physics%20engine&lr&hl=no&pg=PP1#v=onepage&q=Physics%20engine&f=false

[5] Yıldırım, Şahin and Arslan, Erdem, "Ode (open dynamics engine) based stability control algorithm for six legged robot," **Measurement : journal of the International Measurement Confederation**, vol. 124, pp. 367–377, 2018.

[6] J. C. Martinez-Franco and D. Alvarez-Martinez, "Physx as a middleware for dynamic simulations in the container loading problem," in **2018 Winter Simulation Conference (WSC)**. IEEE, 2018, pp. 2933–2940.

[7] B. Nicoll and B. Keogh, **The Unity Game Engine and the Circuits of Cultural Software**. Cham: Springer International Publishing, 2019, pp. 1–21. [Online]. Available: https://doi.org/10.1007/978-3-030-25012-6_1

[8] G. Dotan. (2015, Jan) Top 10 Unity Games Ever Made. [Online]. Available: https://blog.soomla.com/2015/01/top-10-unity-games-ever-made.html

[9] J. Drake. (2020, Jun) 15 Great Games That Use The Unreal 4 Game Engine. [Online]. Available: https://www.thegamer.com/great-games-use-unreal-4-game-engine/

[10] Arnia Software, "What Makes Unity So Popular in Game Development?" [Online]. Available: https://www.arnia.com/what-makes-unity-so-popular-in-game-development/

[11] A. Kumar, **Introduction to the Software**. Berkeley, CA: Apress, 2020, pp. 9–13. [Online]. Available: https://doi.org/10.1007/978-1-4842-6077-7_2

[12] Epic Games. (2020, Jun) A first look at Unreal Engine 5. [Online]. Available: https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5

[13] ——, "Broadcast Live Events." [Online]. Available: https://www.unrealengine.com/en-US/industry/broadcast-live-events

[14] ——, "Automotive & Transportation." [Online]. Available: https://www.unrealengine.com/en-US/industry/automotive-transportation

[15] S. A. Bagloee, M. Tavana, M. Asadi, and T. Oliver, "Autonomous vehicles: challenges, opportunities, and future implications for transportation policies," **Journal of Modern Transportation**, vol. 24, no. 4, pp. 284–303, Dec 2016. [Online]. Available: https://doi.org/10.1007/s40534-016-0117-3

[16] B. S. Kerner, "Effect of autonomous driving on traffic breakdown in mixed traffic flow: A comparison of classical acc with three-traffic-phase-acc (tpacc)," **Physica A**, vol. 562, 2021.

[17] J. Wulf and I. Blohm, "Fostering value creation with digital platforms: A unified theory of the application programming interface design," **Journal of management information systems**, vol. 37, no. 1, pp. 251–281, 2020.

[18] 4D-Virtualiz. (2021, Jan) 4d virtualiz - robotics simulator. [Online]. Available: https://www.4d-virtualiz.com/robotics-simulator

[19] Air Sim. (2021, Jan) Welcome to AirSim. [Online]. Available: https://microsoft.github.io/AirSim

[20] ApolloAuto. (2021, Jan) Github - Apollo Simulator. [Online]. Available: https://github.com/ApolloAuto/apollo

[21] The Autoware Foundation. (2021, Jan) Gitlab - Autoware Documentation. [Online]. Available: https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto

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

[23] Voyage Deepdrive. (2021, Jan) DeepDrive Website. [Online]. Available: https://deepdrive.voyage.auto/

[24] Open Source Robotics Foundation. (2014) Gazebo Website. [Online]. Available: http://gazebosim.org

[25] Der, Ralf and Martius, Georg. (2021, Jan) LPZRobots Github. [Online]. Available: https://github.com/georgmartius/lpzrobots

[26] R. Der and G. Martius, **The LpzRobots Simulator**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 293–308. [Online]. Available: https://doi.org/10.1007/978-3-642-20253-7_16

[27] LG Electronics America RD Center. (2021, Jan) LGSVL Simulator. [Online]. Available: https://www.lgsvlsimulator.com/about

[28] AnyKode. (2019) Marilou Simulator. [Online]. Available: http://www.anykode.com/index.php

[29] rFpro. (2021, Jan) Driving Simulation. [Online]. Available: https://www.rfpro.com/driving-simulation

[30] ——. (2021, Jan) Supervised Learning Autonomous Driving. [Online]. Available: https://www.rfpro.com/virtual-test/dlad-deep-learning-autonomous-driving

[31] Image Space Inc. (2021) ISI Motor Software Engine. [Online]. Available: https://imagespaceinc.com

[32] The TORCS racing board. (2017) TORCS Competition. [Online]. Available: http://www.berniw.org/trb/events/eventlist.php

[33] Cyberbotics. (2021, Jan) Webots Github. [Online]. Available: https://github.com/cyberbotics/webots/tree/released

[34] ——. (2021, Jan) Cyberbotics. [Online]. Available: https://cyberbotics.com

[35] Carla. (2021, Jun) Tutorial - Add custom map. [Online]. Available: https://carla.readthedocs.io/en/latest/tuto_A_add_map

[36] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," **Sensors (Basel, Switzerland)**, vol. 19, no. 3, p. 648, Feb 2019, 30764486[pmid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/30764486

[37] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen, and U. P. Schultz, "A survey of open-source uav flight controllers and flight simulators," **Microprocessors and microsystems**, vol. 61, pp. 11–20, 2018.

[38] Q. Chao, Z. Deng, and X. Jin, "Vehicle–pedestrian interaction for mixed traffic simulation," **Computer animation and virtual worlds**, vol. 26, no. 3-4, pp. 405–412, 2015.

[39] R. C. Arkin, "Ethics and autonomous systems: Perils and promises [point of view]," **Proceedings of the IEEE**, vol. 104, no. 10, pp. 1779–1781, 2016.

[40] J. Borenstein, J. R. Herkert, and K. W. Miller, "Self-driving cars and engineering ethics: The need for a system level analysis," **Science and engineering ethics**, vol. 25, no. 2, pp. 383–398, 2019.

[41] A. Hevelke and J. Nida-Rümelin, "Responsibility for crashes of autonomous vehicles: An ethical analysis," **Science and Engineering Ethics**, vol. 21, no. 3, pp. 619–630, Jun 2015. [Online]. Available: https://doi.org/10.1007/s11948-014-9565-5

[42] A. Martinho, N. Herber, M. Kroesen, and C. Chorus, "Ethical issues in focus by the autonomous vehicles industry," **Transport Reviews**, vol. 0, no. 0, pp. 1–22, 2021. [Online]. Available: https://doi.org/10.1080/01441647.2020.1862355

[43] K. Miller, "Moral responsibility for computing artifacts: The rules," **IT Professional**, vol. 13, pp. 57 – 59, 07 2011.

[44] Open Source Initiative. (2020, Jun) The MIT License. [Online]. Available: https://opensource.org/licenses/MIT

[45] UK Gov. (2018) Automated and Electric Vehicles Act 2018. [Online]. Available: https://www.legislation.gov.uk/ukpga/2018/18/part/1/enacted

[46] ——. (2016, Oct) Robotics and artificial intelligence. [Online]. Available: https://publications.parliament.uk/pa/cm201617/cmselect/cmsctech/145/14506.htm