



Bachelor Thesis

# **Formula SAE – Simulation & Autonomous Control**

**Alistair Thorogood**

Supervised by: Dr Craig Lobsey & Dr Tobias Low

Bachelor of Engineering (Honours)  
Mechatronics Major



University of  
**Southern  
Queensland**

School of Engineering

September 2023



## Abstract

The Society of Automotive Engineers (SAE) have been running a student-based racing competition the Formula SAE (FSAE) since 1981, there are currently over 600 competing teams from universities all over the world. The competition has evolved, with the automotive industry, to include an electric vehicle class since 2013 and an autonomous vehicle class since 2017.

This thesis project is intended to serve as an initial entry into Autonomous FSAE development for the UniSQ team. To achieve this, three project aims were developed and successfully achieved.

### Project Aim 1: Set up and Test Simulation Environment

A range of existing Formula Student simulators were evaluated and the Formula Student Driverless Simulator was found to be most suitable for this project. The Simulator was installed, tested, and utilised for the development of basic self-driving.

### Project Aim 2: Demonstrate Basic Self-Driving

Basic self-driving was demonstrated utilising C++ and python with the ROS2 robotics framework. ROS2 nodes were created for traffic cone perception, steering angle determination, throttle position, and vehicle control. A final lap time of 82.21 seconds was achieved, a significant improvement from the initial lap time of 160.37 seconds

### Project Aim 3: Establish UniSQ FSAE Autonomous Development Platform

A git repository containing the ROS2 workspace was utilised as the development platform. This contained all nodes for basic self-driving, improved perception and visualisation tools used throughout the development.

Above the aims of this project, an effective and robust LiDAR based perception algorithm was developed based off the Velodyne VLP-16 LiDAR. Using the VLP-16 LiDAR allowed for easier integration into a real-world vehicle. Within the simulator the perception range was improved from 7m to greater than 20m. The improved simulation perception allows for future simulated autonomous development in; motion estimation and mapping, and vehicle control.



## Disclaimer



University of  
**Southern  
Queensland**

Faculty of Health, Engineering & Sciences

## ENG4111 & ENG4112 Research Project **Limitations of Use**

The Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering and Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitles “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and any other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.




## Candidates Certification

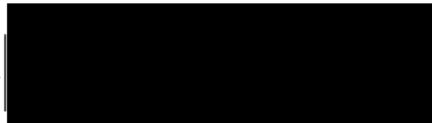
I certify that the ideas, designs and experimental work, results, analysis and conclusions set out in this dissertation are entirely me own efforts except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course of institution, except where specifically stated.

Alistair Thorogood

Student Number: 

Signed

A large solid black rectangular box used for redaction, covering the signature area.

Date: 22 / 09 / 2023



## **Acknowledgments**

I would like to express my gratitude to my supervisors, Dr. Craig Lobsey & Dr Tobias Low, for their guidance, support, and encouragement throughout my thesis research project. Their expert advice and feedback have been instrumental in shaping the direction of my work and helping me achieve more than I initially thought possible.

I would also like to acknowledge the previous Formula SAE and Formula Student teams who generously shared parts of their research and autonomous control designs. In particular, I am grateful to the AMZ racing team, Monash Motorsport team, and Czech Technical University for their contributions to my work.

I would also like to acknowledge everyone who has contributed to the Formula Student Driverless Simulator as this project would not have been possible if it were not for their continued dedication.

Most of all, I would like to thank my beautiful wife, Emily, for her support and understanding during this demanding period. Her love and encouragement have kept me motivated and focused, even during the most challenging of late nights. She has taken on the responsibility of caring for our newborn Kai while I spent countless hours programming and writing my thesis report. I could not have completed this work without her.



# Table of Contents

Abstract .....	i
Disclaimer .....	ii
Candidates Certification.....	iii
Acknowledgments.....	iv
List of Figures .....	vii
List of Tables .....	viii
List of Equations .....	ix
List of Appendices .....	ix
Abbreviations.....	x
1 Introduction.....	12
1.1 Formula SAE / FSAE / Formula Student.....	12
1.2 FSAE Autonomous Competition Rules .....	13
1.3 Autonomous Control Software.....	13
1.4 Project Aim & Objectives .....	14
2 Literature Review .....	16
2.1 SAE Competition .....	16
2.2 Successful Formula SAE / Formula Student teams .....	18
2.3 Simulation .....	18
2.4 Perception.....	22
2.5 Motion Estimation and Mapping.....	25
2.6 Control.....	28
2.7 Software Frameworks .....	31
3 Project Design.....	36
3.1 Safety, Ethics and Environment.....	36
3.2 Project Risk .....	36
3.3 Resources .....	37



3.4	Timeline .....	37
3.5	Project Tasks .....	38
3.6	Assumptions and Limitations.....	39
4	Autonomous Software Development.....	40
4.1	Simulator Setup .....	40
4.2	Complete a Basic Lap of Simulation Track .....	44
4.3	Perception (LiDAR) .....	49
4.4	System Synthesis.....	63
5	Contributions – Git Repository.....	65
6	Conclusion .....	66
6.1	Future Work .....	66
6.2	Links.....	67
7	References.....	68
8	Appendices.....	71
	Appendix A1 - Project Specification.....	71
	Appendix A2 - Personal and Property RMP .....	73
	Appendix A3 - Project Gantt Chart .....	75
	Appendix A4 - Running the Simulator – Cheat Sheet.....	76



## List of Figures

FIGURE 1: TRACKDRIVE LAYOUT (FSG 2022A).....	17
FIGURE 2: EUFS_SIM SIMULATOR SCREENSHOT (EDINBURGH-UNIVERSITY N.D.) .....	19
FIGURE 3: FSSIM SIMULATION SCREENSHOT.....	20
FIGURE 4: FORMULA STUDENT DRIVERLESS SIMULATOR SCREENSHOT .....	21
FIGURE 5: VLP-16; 16 CHANNELS WITH 2° RESOLUTION (OKUNSKY & NESTEROVA 2019) .....	23
FIGURE 6: RACE PATH CENTRELINE - DELAUNAY TRIANGULATION (KURUVILLA 2022) .....	26
FIGURE 7: POINT MASS MODEL (EDINBURGH-UNIVERSITY N.D.) .....	27
FIGURE 8: BICYCLE MODEL (KRITAYAKIRANA & GERDES 2012).....	27
FIGURE 9: PROPORTIONAL STEERING ERROR ANGLE .....	28
FIGURE 10: PURE PURSUIT STEERING CONTROLLER (DING 2020).....	29
FIGURE 11: STANLEY STEERING CONTROLLER.....	30
FIGURE 12: ROS2 COMMUNICATIONS (ROS N.D.).....	32
FIGURE 13: VERSION CONTROL SURVEY (DONOVAN 2023) .....	33
FIGURE 14: UPGRADED PC COMPONENTS .....	40
FIGURE 15: GPU PARAMETERS.....	42
FIGURE 16: RECORDED GPU TEMPERATURES .....	42
FIGURE 17: SIMULATOR RUNNING.....	43
FIGURE 18: BASIC PATH PLANNING FLOWCHART .....	45
FIGURE 19: DESIRED LOCATION ERRORS .....	45
FIGURE 20: DESIRED LOCATION ERROR RADIUS.....	46
FIGURE 21: SIMULATOR EXAMPLE PERCEPTION ERROR.....	49
FIGURE 22: SIMULATOR LIDAR OPTIONS (FORMULA-STUDENT-DRIVERLESS-COMMUNITY N.D.).....	50
FIGURE 23: IMPROVED PERCEPTION LIDAR SETTINGS .....	52
FIGURE 24: PERCEPTION FLOWCHART - MAIN CODE .....	52
FIGURE 25: PERCEPTION FLOWCHART - FILTER INVALID POINTS.....	53
FIGURE 26: PERCEPTION FLOWCHART - FIND GROUND PLANE .....	55
FIGURE 27: CUMULATIVE ERROR SCATTER PLOT.....	56
FIGURE 28: PERCEPTION FLOWCHART - FILTER GROUND & AIR POINTS .....	57
FIGURE 29: PERCEPTION FLOWCHART - CLUSTER & FILTER.....	58
FIGURE 30: PERCEPTION ALGORITHM RUN TIME TESTS .....	59
FIGURE 31: LIDAR POINTS RESULTS .....	59
FIGURE 32: VISUALISATION OF GROUND PLANE REMOVAL .....	60
FIGURE 33: IMPROVED PERCEPTION ALGORITHM SCREENSHOT .....	60
FIGURE 34: IMPROVED PERCEPTION ALGORITHM SCREENSHOT .....	61





## List of Tables

TABLE 1: FORMULA STUDENT MAXIMUM AVAILABLE POINTS (FSG 2023) .....	13
TABLE 2: TRAFFIC CONE SPECIFICATIONS (SAE 2022b) .....	17
TABLE 3: VLP-16 HORIZONTAL RESOLUTION (VELODYNE 2019).....	23
TABLE 4: PROJECT RISK - CUSTOMISED RISK MATRIX.....	36
TABLE 5: PROJECT RISK ANALYSIS .....	37
TABLE 6: PROPOSED PC UPGRADES .....	37
TABLE 7: BREAKDOWN OF PROJECT PHASES .....	38
TABLE 8: SIMULATOR EXAMPLE LAP TIMES.....	47
TABLE 9: BASIC LAP ALGORITHM LAP TIMES, MAX SPEED = 4 M/S .....	47
TABLE 10: BASIC LAP ALGORITHM LAP TIMES, MAX SPEED = 6.5 M/S .....	47
TABLE 11: DISTANCE TRAVELLED (M) AT DIFFERENT LIDAR FREQUENCY AND VEHICLE SPEEDS.....	50
TABLE 12: VLP-16 LIDAR RESOLUTION AND POINTS.....	51
TABLE 13: VISUALISATION OF HORIZONTAL RESOLUTION AT MIN AND MAX FREQUENCY SETTINGS.....	51
TABLE 14: IMPROVED PERCEPTION ALGORITHM LAP TIMES, MAX SPEED = 6.5 M/S .....	61
TABLE 15: IMPROVED PERCEPTION ALGORITHM LAP TIMES, MAX SPEED = 9 M/S .....	61
TABLE 16: ROS2 WORKSPACE SUMMARY .....	63
TABLE 17: ROS2 LAUNCH FILE SUMMARY .....	64



## List of Equations

EQUATION 1: PROPORTIONAL STEERING CONTROL EQUATIONS .....	28
EQUATION 2: PURE PURSUIT STEERING CONTROLLER EQUATION .....	29
EQUATION 3: STANLEY STEERING CONTROLLER .....	30
EQUATION 5: CALCULATING TWO VECTORS FOR GROUND PLANE DETECTION .....	53
EQUATION 6: NORMAL VECTOR CALCULATION FOR GROUND PLANE DETECTION .....	53
EQUATION 7: PARAMETER A, B & C FOR GROUND PLANE DETECTION.....	54
EQUATION 8: PARAMETER D FOR GROUND PLANE DETECTION.....	54
EQUATION 9: CALCULATING DISTANCE TO PLANE FOR GROUND PLANE DETECTION.....	54

## List of Appendices

- Appendix A1 - Project Specification
- Appendix A2 - Personal and Property RMP
- Appendix A3 - Project Gantt Chart
- Appendix A4 - Running the Simulator – Cheat Sheet



## Abbreviations

AV – Autonomous Vehicle

CPU – Central Processing Unit

EKF – Extended Kalman Filter

FSAE – Formula Society of Automotive Engineers

FSG – Formula Student Germany

FOV – Field of View

GSS – Ground Speed Sensor

GPS – Global Positioning System

GPU – Graphical Processing Unit

LHS – Left Hand Side

LiDAR – Light Detection and Ranging

PC – Personal Computer

RAM – Random Access Memory

RHS – Right Hand Side

RMP – Risk Management Plan

ROS – Robot Operating System

SAE – Society of Automotive Engineers

SLAM – Simultaneous Localisation and Mapping

USQ/UniSQ – University of Southern Queensland

# 1 Introduction

Racing has long served as a testing ground for new vehicle technologies. Competitions such as Formula 1, Indy, World Rally Championship, and many more have been pioneering vehicle innovation. Examples of this innovation include disc brakes, the turbocharger and more recently hybrid vehicle technology (Betz et al. 2022).

In parallel, recent advancements in computing, artificial intelligence, and robotics have led to remarkable progress in the field of autonomous driving. As a result of this several autonomous racing competitions are emerging to develop and test cutting edge autonomous driving technologies. These competitions include Indy Autonomous Challenge, Roborace and more. Autonomous racing student competitions such as Formula SAE and Formula Student provide the foundation for which this thesis project is based.

## 1.1 Formula SAE / FSAE / Formula Student

The Society of Automotive Engineers (SAE) have been running a student-based racing competition the Formula SAE (FSAE) since 1981, there are currently over 600 competing teams from universities all over the world. The European competition is named 'Formula Student' for the remainder of the report Formula SAE, FSAE and Formula Student may be used interchangeably. The primary purpose of the competition is for the development of university students in fields such as engineering and project management.

Over the years the competition has developed, with the automotive industry, to include an electric vehicle class since 2013 and a driverless vehicle class since 2017. for the remainder of this report the words driverless and autonomous will be used interchangeably in relation to vehicle control.

The autonomous competition is relatively new. Despite being held in Europe from 2017, the first autonomous vehicle to meet all requirements and complete a lap of the track was the Monash University team in 2022 (Monash-Motorsport 2022). There have been several teams from Europe that have met all requirements to compete and completed the required events, in general these are teams with much higher budget and student participation than the current UniSQ team. In total there are 123 teams registered for autonomous competition, five of which are Australian teams (FSG n.d.). Being a student competition, some of these teams are willing to share their research and design after the competition year. Most of the literature review and guidance for this design will be a result of this shared information.

## 1.2 FSAE Autonomous Competition Rules

Due to infancy of the FSAE autonomous competition it is currently run as a non-scored demonstration event including Trackdrive, Emergency Braking System Test, Inspection and Manual driving (SAE 2022b). The more matured European Formula Student competition provides a likely framework for future autonomous competition scoring with the maximum available points from the 2023 competition outlined in Table 1 below.

**Table 1: Formula Student Maximum Available Points (FSG 2023)**

<b>Static Events:</b>	
Engineering Design	150 Points
<b>Dynamic Events:</b>	
Skidpad	75 Points
Acceleration	75 Points
Autocross	100 Points
Trackdrive	200 Points
<b>Overall:</b>	600 Points

For the purpose of this project only the more advanced autocross and trackdrive events have been considered. Both these events utilise the same track, which is further outlined in section 2.1. The autocross event consists of two single-lap runs and the use of any prior track data is forbidden. The Trackdrive event consists of a single 10-lap race. There is no stipulation on prior track data for the trackdrive event however, D2.4 states no map data is provided by the officials (SAE 2022b).

## 1.3 Autonomous Control Software

Autonomous control software can be categorised into three key sections, each requiring significant effort and expertise to develop a competitive autonomous control system.

Completing a competitive autonomous control software will require multiple bachelor thesis or similar research efforts dedicated to each section. The three sections are as follows:

1. Perception: Utilising machine/computer vision techniques to provide meaningful data from the vehicle's environment. In FSAE environment this will primarily be focused on identifying the traffic cones that outline the track.
2. Motion Estimation and Mapping: Localisation of the vehicle, mapping the track, and determining optimised racing path. These processes are essential for the vehicle to understand its position relative to the track and plan its trajectory accordingly.
3. Control: Controlling the steering, acceleration, and braking parameters to direct the vehicle along the desired path.



## **1.4 Project Aim & Objectives**

The UniSQ team are in the process of developing a driven vehicle and have not yet made any progress into the autonomous competition. This thesis project is intended to serve as an initial entry into Autonomous FSAE development for the UniSQ team. Three project aims have been developed to meet this intention.

### **Project Aim 1: Set up and Test Simulation Environment**

The cost and resource requirements of developing a real-world autonomous vehicle are currently a major hurdle to autonomous software development for the UniSQ team. A simulation environment offers a solution that is both cost-effective and safer compared to real-world testing. A simulation environment will potentially justify and pave the way to development of a real-world vehicle.

This project will aim to satisfy the aim ‘Set up and Test Simulation Environment’ by completing the following objectives:

1. Research available simulation software and determine if existing simulators exist.
2. Determine preferred simulator from existing/new solutions.
3. Develop/Install and trial simulation environment.

### **Project Aim 2: Demonstrate Basic Self-Driving**

Once a simulation environment is validated the next logical step in an autonomous software development is to demonstrate basic self-driving behaviours. This will require a basic implementation of the three sections of autonomous control software being perception, motion estimation and mapping, and vehicle control.

This project will aim to satisfy the aim ‘Demonstrate Basic Self-Driving’ by completing the following objectives within a simulation environment:

1. Research, develop and test a basic perception algorithm.
2. Research, develop and test a basic Motion Estimation and Mapping algorithm.
3. Research, develop and test a basic vehicle control algorithm.

### **Project Aim 3: Establish UniSQ FSAE Autonomous Development Platform**

Once a simulation environment and basic self-driving are validated there will be significant work remaining to develop a competitive autonomous racing software. For the benefit of the UniSQ race team it will be important to hand over the design and results of this project in a way that will allow future students to easily access, understand and utilise.



This project will aim to satisfy the aim 'Establish UniSQ FSAE Autonomous Development Platform' by completing the following objectives:

1. Establish a version control and collaborative repository.
2. Compile all completed objectives, algorithms, and relevant documentation into a handover package.

## 2 Literature Review

### 2.1 SAE Competition

There are four sources for competition rules and design standards. Each of these refer to and relate to each other. The latest available documents have been used, at the time of starting this project this is a mix of 2022 and 2023 resources:

1. Formula SAE Rules 2023 (SAE 2022a).
2. Autonomous vehicle addendum (SAE 2022b).
3. Formula Student Rules 2023 (FSG 2023).
4. Formula Student Germany (FSG) Handbook 2023 (FSG 2022a).

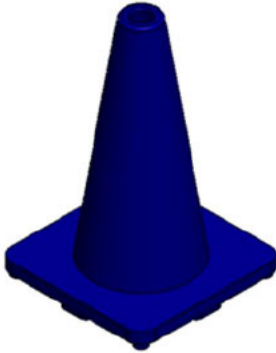

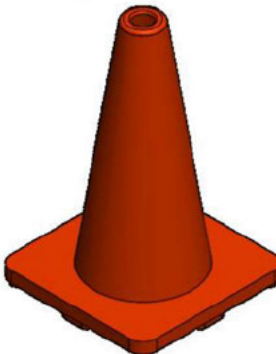
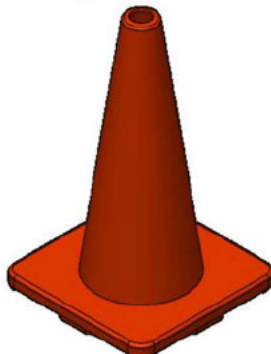
The addendum states that the main dynamic autonomous competition will be the trackdrive event. Requirements of the trackdrive event are outlined in the EFS rules. A summary of the most applicable trackdrive requirements is shown below:




- D8.1 The trackdrive layout is a closed loop circuit built to the following guidelines:
  - Straights: No longer than 80m
  - Constant Turns: up to 50m diameter
  - Hairpin Turns: Minimum of 9m outside diameter (of the turn)
  - Miscellaneous: Chicanes, multiple turns, decreasing radius turns, etc.
  - The minimum track width is 3m
- D8.1.2 The length of one lap is approximately 200m to 500 m.
- D8.2.6 After ten laps the vehicle must come to a full stop within 30m behind the finish line on the track and enter the finish-state described in T14.10.
- D8.2.7 There will be no last lap signal i.e. the vehicle should count laps itself.
- D9.1.7 a 2s penalty will apply for each traffic cone knocked down or out.
- D9.1.7 a 10s penalty will apply for each time the vehicle is off course. This occurs when all four wheels of the vehicle are outside the track boundary.

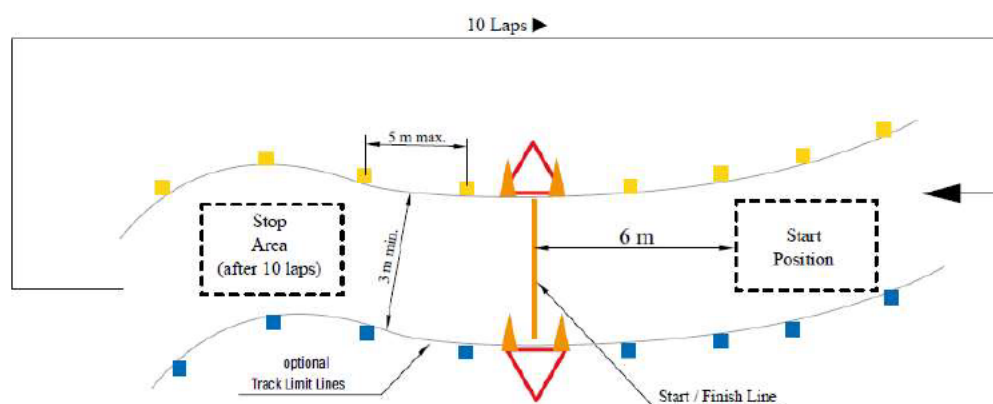
An understanding of Traffic cone purpose and dimensions are required to develop a functioning perception algorithm. Table 2 from appendix PDA-3 of the autonomous addendum outlines traffic cone specifications an important note is the removal of the stripes from the EFS competition. Figure 1 from the FSG Competition Handbook outlines the trackdrive layout.



**Table 2: Traffic Cone Specifications (SAE 2022b)**

<p>Small Blue Cone (Plain) 210mm x 210mm x 300mm <a href="https://grabcad.com/library/300mm-blue-road-cone-1">https://grabcad.com/library/300mm-blue-road-cone-1</a></p> 	<p>Small Yellow Cone (Plain) 210mm x 210mm x 300mm <a href="https://grabcad.com/library/300mm-yellow-road-cone-1">https://grabcad.com/library/300mm-yellow-road-cone-1</a></p> 
<p>Small Orange Cone (Plain) 210mm x 210mm x 300mm <a href="https://grabcad.com/library/300mm-orange-road-cone-1">https://grabcad.com/library/300mm-orange-road-cone-1</a></p> 	<p>Large Orange Cone (Plain) 270mm x 270mm x 450mm <a href="https://grabcad.com/library/450mm-orange-road-cone-2">https://grabcad.com/library/450mm-orange-road-cone-2</a></p> 

-  Yellow/Blue Cone
-  Small/Big Orange Cone
-  Red TK Marking & TK Equipment (Shape undefined)

**Figure 1: Trackdrive Layout (FSG 2022a)**



## 2.2 Successful Formula SAE / Formula Student teams

The aim of this project is to enhance the autonomous vehicle capabilities of the UniSQ FSAE team by leveraging the knowledge and experiences of successful teams. Out of the 122 registered teams in the autonomous competition, only a small percentage have successfully met the design requirements and completed a track drive event. For instance, in the 2022 Formula Student event, only five out of the nineteen registered teams scored points (FSG 2022b). To accomplish this aim, this project will primarily draw upon the design approaches of successful teams that follow an open-source model, openly sharing their research, design, and findings. The three teams predominantly referenced include:

- AMZ Motorsports – Lucerne University of Applied Sciences and Arts and ETH Zurich.
- Monash Motorsports – Monash University
- EFORCE FEE Prague Formula – Czech Technical University in Prague

## 2.3 Simulation

The UniSQ FSAE team does not currently possess a functioning vehicle, as such a simulator is required for all software development during this project. The use of a simulator has the potential to accelerate the development of an autonomous vehicle by removing initial cost barriers of vehicle development and providing a safe environment for vehicle testing.

Several universities have already developed functional lap time simulators, driver simulators and autonomous vehicle simulators to assist with their own vehicle development.

Lap time simulators such as the ones developed by The University of Glasgow (Broatch 2019) and Queen's University of Belfast (Doyle et al. 2019), utilise simplified vehicle and track models such as:

- Steady state simulators where the simulation is divided into discrete segments; either a corner, accelerating straight or de-accelerating straight,
- Quasi-static simulators which are similar to steady state except each corner is broken up into smaller corners of varying radius.
- Transient simulators where the vehicle is modelled as an integrated dynamic system.

This allows suspension, inertia, and damping effects to be taken into consideration.

Lap time simulators are still relevant to FSAE development, particularly for optimising path planning and parameter optimisation. However, a more complex simulator with sensor inputs and other features is required for full autonomous development.

Driver simulators such as the one built by Monash University (Behrendt 2017) may be able to be converted to an autonomous vehicle platform. However, purpose built autonomous vehicle simulators should be considered first.

There are currently at least three open-source autonomous vehicle simulators available for FSAE development:

- eufs\_sim developed by Edinburgh University (Edinburgh-University n.d.)
- FSSim developed by AMZ racing team (AMZ-Driverless n.d.)
- Formula Student Driverless Simulator developed as a collaboration between Formula Student Team Delft, MIT Driverless and FSEast as a substitute for the Formula Student competition during the Covid-19 restrictions in 2020 (Formula-Student-Driverless-Community n.d.)

### 2.3.1 eufs\_sim

eufs\_sim is built using the Gazebo open-source 3D robotics simulator. The simulator allows for customisable vehicle models, weather conditions and a random track generator.

The simulator does not utilise a standard Gazebo physics model and instead utilises a custom point mass or dynamic bicycle model depending on user choice. The simulator documentation states the dynamic bicycle model is better in almost every way and should be used except for special use cases where a simple model is preferred. The dynamic bicycle model is claimed as very accurate near the limits of dynamics but does not consider pitch and roll dynamics. The dynamic bicycle model is further explained in section 262.5.3.

The simulator is actively being maintained and may be a suitable simulation software for the project. Figure 2 Shows a screenshot of eufs\_sim simulation.

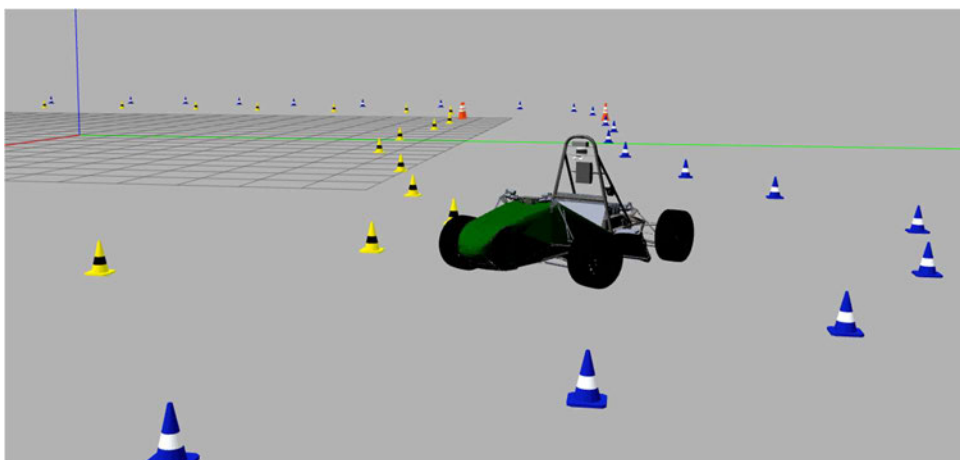


Figure 2: eufs\_sim Simulator Screenshot (Edinburgh-University n.d.)

### 2.3.2 FSSim

FSSim is also built using the Gazebo open-source 3D robotics simulator. FSSIM also uses a dynamic bicycle model instead of a Gazebo physics engine plugin. The bicycle model is incorporated into the simulator using Euler forward discretisation method to overwrite the current pose.

AMZ's FSSim has not been updated since 2019 however, a repository fork by the ARUS Formula student team appears to be updated in 2023 to allowing use with Ubuntu 20.04 and ROS Noetic. Not having an active community maintaining the simulator may present difficulties with integration into this project. Figure 3 shows a screenshot of the FSSim simulation.

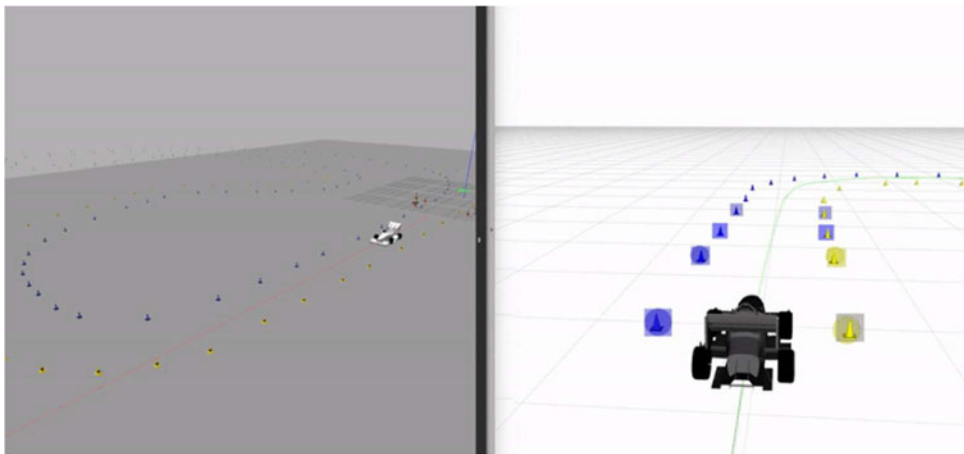


Figure 3: FSSim Simulation Screenshot

### 2.3.3 Formula Student Driverless Simulator

The Formula Student Driverless Simulator is built upon the Unreal Engine 4 development tool and makes use of the Microsoft AirSim plugin. The Unreal Engine is utilised for all physics, lighting, and world-building. The Microsoft AirSim plugin is used to connect Unreal Engine with ROS and the operator.

This simulator utilises an Unreal Engine vehicle physics model 'PhysXVehicles' developed by NVIDIA. This is a far more complex physics model than the bicycle model used by the other simulators. The physics model is beyond the scope of this project however can be researched through NVIDIA's PhysX documentation (Nvidia n.d.).

The Unreal Engine development tool allows the FSEA Simulator to provide a realistic testing environment specifically designed for autonomous formula student vehicles. The rules of the Formula Student are incorporated into the simulator with realistic start and stop signals available.

To assist with setup and familiarisation, the FSEA Simulator has its own dedicated GitHub page with detailed documentation (Formula-Student-Driverless-Community n.d.).

The Simulator is currently maintained by an active community and may be a suitable simulation tool for this project with a more realistic physics engine and graphic environment than the other simulator options. Figure 4 shows a screenshot of the Formula Student Driverless Simulator.



**Figure 4: Formula Student Driverless Simulator Screenshot**

## 2.4 Perception

Perception is the first step in developing autonomous control software for an autonomous vehicle. In the context of an autonomous vehicle, perception is the use of various sensors to perceive the surrounding environment. For this project the only required perception is the location of the traffic cones outlining the race track, this may evolve as the competition evolves to include obstacles or other vehicles.

For reliable perception, inputs from multiple sensors are used to verify each other. The successful AMZ, Monash Motorsport and EForce teams all utilise a combination of LiDAR sensors and camera sensors for traffic cone detection.

Other perception sensor options such as ultrasonic and RADAR are available, however, these sensor options are not utilised by successful formula student teams and are not available within the simulators. As a result, they are excluded from this project.

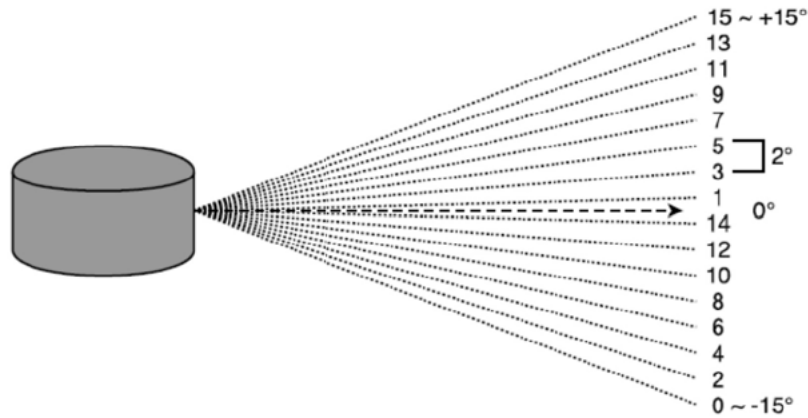
### 2.4.1 LiDAR

Light detection and radar (LiDAR) work on the principle of measuring the time of flight of a laser pulse. The LiDAR sensor emits pulsed light waves into the surrounding environment, these pulses bounce off objects and return to the sensor. The distance to an object can be determined by the time the pulse takes to return to the sensor. (Velodyne n.d.) The intensity of the returning light wave can also be used to analyse the surface properties of an object.

Advantages of LiDAR in autonomous racing include accurate mapping, object detection in varying conditions, and fast data acquisition. However, its disadvantages include high costs, and susceptibility to environmental conditions such as heavy rain and fog.

The UniSQ FSAE team have a Velodyne VLP-16 LiDAR available to use. This is a reputable and reliable LiDAR with the capabilities to perform FSAE vehicle traffic cone perception. Simulator setup and perception algorithm development will be based off the Velodyne VLP-16 LiDAR user manual (Velodyne 2019).

The VLP-16 is a rotating 'surround' LiDAR. These are characterised by a number of channels in the vertical direction for example Velodyne VLP-16 has 16 channels  $2^\circ$  apart, this results in a  $2^\circ$  horizontal resolution as per Figure 5.



**Figure 5: VLP-16; 16 Channels with 2° Resolution (Okunsky & Nesterova 2019)**

The laser emitter rotates allowing a 360° horizontal field of view (FOV). The rotational speed and laser frequency determines the horizontal resolution. Table 3 shows the horizontal resolution of the VLP-16 dependent on RPM setting.

**Table 3: VLP-16 Horizontal Resolution (Velodyne 2019)**

RPM	Resolution
300	0.1°
600	0.2°
900	0.3°
1200	0.4°

All the laser pulses and their corresponding measurements are combined to form a point cloud. The point cloud is an unordered set of datapoints representing (x, y, z) coordinates of a 3D space. In the FSEA simulator a PointCloud2 ROS2 message is published by the customisable vehicle LiDAR.

### 2.4.2 Camera

There are two types of cameras used for perception purposes mono and stereo. A mono camera is a single camera taking a single image. Mono cameras lack depth perception and can only be used to determine distances if existing parameters is known, such as the dimensions of the traffic cones. Stereo cameras utilise multiple cameras to take an image from slightly different viewpoints and have the advantage of being able to estimate distances, similar to human vision.



Advantages of cameras are their relatively low costs when compared to other sensing technology and their versatility in detecting a wide range of objects. A major disadvantage of camera-based perception is its complexity, the data acquired by cameras is up to 70 times greater than that acquired by LiDAR and GPS (Ovenden 2019). Additionally, the computational methods required to extract information from the data requires complex machine learning algorithms with large datasets required for accurate training.

Due to the complexity, computational requirements, and requirement of real-world testing camera vision; it will be excluded from this project and likely require a future thesis project of its own.



## **2.5 Motion Estimation and Mapping**

Motion estimation and mapping encompasses vehicle localisation, mapping and path planning: Vehicle localisation and mapping is a complex task and is often referred to as a chicken-or-egg problem as vehicle localisation requires a map and mapping requires vehicle localisation. This problem is often referred to as Simultaneous Localisation and Mapping (SLAM) (Large 2020). SLAM is further reviewed in 2.5.1.

Path planning is the process of determining a desired vehicle trajectory. This can be broken down into determining the racing path and the velocity profile of the race path. Determining a race path is further reviewed in 2.5.2 and the velocity profile is further reviewed in 2.5.3.

### **2.5.1 Simultaneous Localisation and Mapping**

Simultaneous Localisation and Mapping (SLAM) is the process of a robot, in this case a vehicle, building a map of an unknown environment while keeping track of its position. SLAM is not an exact process, instead it is a best estimation of localisation and mapping with the accuracy dependant on the quality of information from various sensors.

Inputs for the localisation of the vehicle will be both relative and absolute. Relative sensors being relative to the vehicle including input from wheel speed sensors, inertial measurement units (IMU), and ground speed sensors. Absolute sensors have a reference to the external environment including inputs from cameras, LiDAR and GPS.

There are numerous methods of SLAM implementation a review of Formula Student SLAM algorithms highlights three popular SLAM methods; extended Kalman filter (EKF) SLAM, fastSLAM and GraphSLAM.

The Monash Motorsport team initially implemented an EKF SLAM method using only LiDAR input prior to integration of GPS, IMU and stereoscopic cameras (Ovenden 2019). The initial SLAM implementation and the integration of additional sensors were separate thesis projects. Future work for the Monash Motorsport team includes implementation of a fastSLAM method utilising their EKF method.

The AMZ motorsport team state using fastSLAM 2.0, which is a revised Extend Kalman particle filter method (Kabzan et al. 2020).

A master's thesis by Nick Le Large summarises the trade-offs between the more accurate computational intensive GraphSLAM compared with the less accurate less computational intensive EKF SLAM (Large 2020).

### 2.5.2 Racing Path

The simplest race path for a FSAE vehicle is a centreline of the race track. The AMZ racing team and Monash Motorsport teams both mention the use of Delauney Triangulation to determine the centreline of the track. MathWorks® provide an example of using Delauney Triangulation to determine the centreline of a Formula Student track (Kuruvilla 2022), this example is visualised in Figure 6. Other methods of determining the centreline are available such as the traverse line method utilised by the Monash Motorsport team prior to Delauney Triangulation (Slomoi 2018). The traverse line method was found to be computationally intensive and more error prone than Delauney Triangulation.

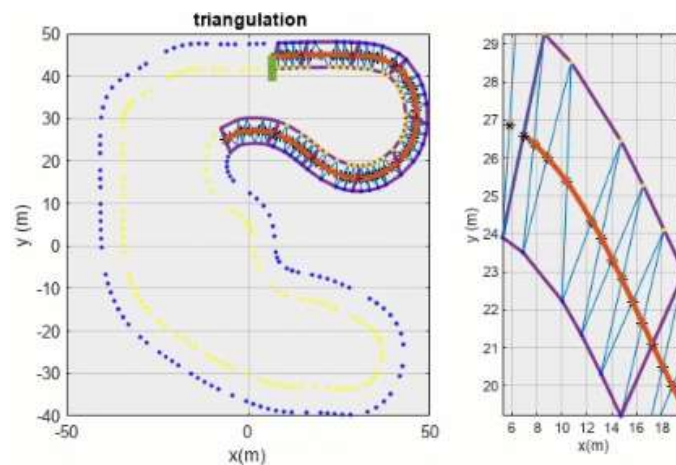


Figure 6: Race Path Centreline - Delaunay Triangulation (Kuruvilla 2022)

More advanced methods of optimised race paths are available. Publicly available repositories allow optimisation of race paths on a range of criteria including shortest path, minimum curvature, and minimum time (TUMFTM 2021). Given the narrow FSAE trackdrive track, the tolerances associated with SLAM and the penalties associated with impacting a traffic cone; there is little to gain from an optimised racing path. A centreline will provide sufficient autonomous development for the UniSQ team until a more advanced method is required.

### 2.5.3 Velocity Profile

A velocity profile provides a target velocity and acceleration for each point along the race track. Developing a velocity profile requires the development of a lap time simulation, lap time simulators are briefly discussed in section 2.3. To create a lap time simulation a dynamic model of the vehicle is required.

There are many options for vehicle dynamic models. Simple point mass models, two wheeled bicycle models, and more advanced four wheeled models or complete vehicle models that consider pitch, roll etc. such as the PhysXVehicles model referenced in section 2.3.3.

Both the AMZ racing team and Monash Motorsport team utilise the bicycle model rather than more advanced four wheel or complete vehicle models (Slomoi 2018; Kabzan et al. 2020). A point mass or bicycle model will allow sufficient autonomous development for the UniSQ team until a more advanced method is required.

A more advanced model is the two wheeled dynamic bicycle model as shown in Figure 8. This model considers vehicle properties such as dimensions, steering characteristics, yaw inertia and separates front and rear tyre frictional coefficients.

Both the AMZ racing team and Monash Motorsport team utilise the bicycle model rather than more advanced four wheel or complete vehicle models (Slomoi 2018; Kabzan et al. 2020). A point mass or bicycle model will allow sufficient autonomous development for the UniSQ team until a more advanced method is required.

## 2.6 Control

For this project, control of the FSEA autonomous vehicle is limited to the steering, braking and throttle position within the simulation.

The brake position and throttle position within the simulators are a value from [0:1] and will be dependent on the current vehicle velocity, desired velocity of the velocity profile, and the desired acceleration of the velocity profile.

The steering angle within the simulators is a value from [-1:1] with -1 being full steer left and +1 being full steer right. There are several options for autonomous steering control. The simplest of steering control is a bang-bang style controller, this controller simply turns set\_steering\_angle left if the race path is to the left of the vehicle or set\_steering\_angle right if the race path is to the right of the vehicle. This is a very simple yet erratic and ineffective method of race vehicle steering control.

A rudimentary yet slightly more effective method of steering is a proportional controller. In proportional steering control an error angle is calculated between the vehicle and a desired location on the race path, a set distance in front of the vehicle. Figure 9 shows the error angle  $\theta$ , where the red dot is the desired location with reference to the vehicle coordinate frame.

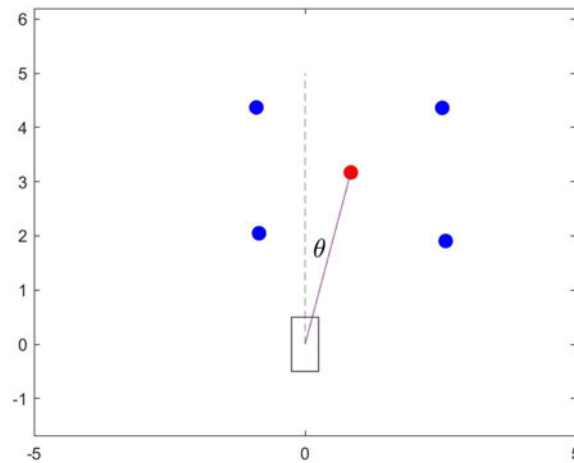


Figure 9: Proportional Steering Error Angle

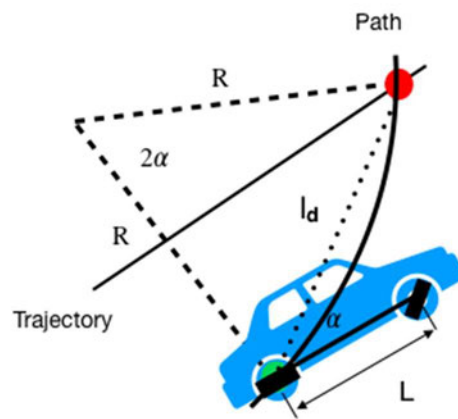
The steering angle  $\delta$  is then calculated by multiplying error angle  $\theta$  by a gain constant  $K_p$  as per Equation 1. This method may prove useful in development stages, however, will not result in a competitive autonomous vehicle.

### Equation 1: Proportional Steering Control Equations

$$\theta = \text{atan}\left(\frac{x_{desired}}{y_{desired}}\right)$$

$$\delta = K_p * \theta$$

A more effective proportional steering controller is known as a Pure Pursuit steering controller (Ding 2020). Pure Pursuit considers the arc required to move to the desired location and utilises a bicycle model to determine steering angle  $\delta$ . The arc is visualised in Figure 10.



**Figure 10: Pure Pursuit Steering Controller (Ding 2020)**

With incorporation of the bicycle model, the steering angle  $\delta$  of the Pure Pursuit controller is calculated as per Equation 2 where  $\alpha$  is the angle between the vehicles heading and the desired location,  $L$  is the length between the axles and  $l_d$  is the distance to the desired location.

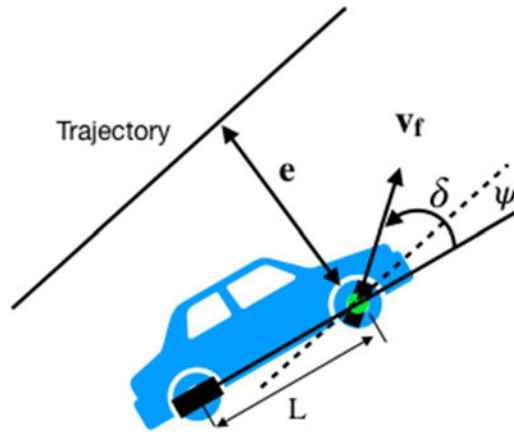
**Equation 2: Pure Pursuit Steering Controller Equation**

$$\delta = \arctan\left(\frac{2L\sin(\alpha)}{l_d}\right)$$

Tuning of the pure pursuit controller is required by adjusting the look ahead distance.

MathWorks® has a Pure Pursuit library available, however implementation appears simple.

An improvement on the pure pursuit controller is the Stanley controller (Ding 2020). The Stanley controller requires a race path with a known trajectory heading. And hence requires an established race path to function. Opposed to the Pure Pursuit method the Stanley method references the centre of the front axle. The heading error  $\psi$  is calculated by determining the angle between the trajectory heading (at closest point) and vehicle heading. A cross-track error  $e$  is the distance from the reference point to the closest point on the track. The heading error and cross-track error are visualised in Figure 11.



**Figure 11: Stanley Steering Controller**

A cross-track error gain  $K_e$  is applied and a minimum and maximum steering angle  $\delta_{\min}$  and  $\delta_{\max}$  is considered. Equation 3 shows the final equation for the Stanley controller.

**Equation 3: Stanley Steering Controller**

$$\delta = \psi + \arctan\left(K_e \frac{e}{v}\right), \delta \in [\delta_{\min}, \delta_{\max}]$$

More advanced steering control methods exist such as predictive Stanley control, model predictive control, and machine learning methods however a Pure Pursuit or Stanley controller will allow sufficient autonomous development for the UniSQ team until a more advanced method is required.

## **2.7 Software Frameworks**

Knowledge of a range of programming languages and software platforms was required as a part of this project. It is not practical to give a detailed literature review for each, instead a high-level overview has been provided. A familiarisation period was built in to the project schedule to ensure competency with all the required programming languages and software platforms.

### **2.7.1 Robotics Frameworks**

Robotics frameworks provide a set of software libraries such as communication protocols, sensor drivers, control algorithms and more. They have been developed to simplify the development and maintenance of robotic systems.

There are a range of robotics frameworks available such as NVIDIA Isaac, Orocos (Open Robot Control Software) and YARP (Yet Another Robotics Platform) and more. However, Robot Operating System (ROS) is the current industry standard, being the most used and most supported with the largest active community. ROS is required for startup and safety protocols within the formula student competition and is the obvious choice of robotics frameworks for this project.

ROS is an open-source robotics middleware suite developed for robot communication and control. There are many different ROS distributions (versions) available, for this project ROS2 Humble will be used as it is the latest non-development distribution and is compatible with Ubuntu 22.04. There is a dedicated ROS2 documentation and tutorial that will be used for familiarisation (ROS n.d.).

Although ROS2's functionality includes debugging, monitoring, visualisation and more the main functionality used in the development of an autonomous vehicle will be ROS2 communications. To understand ROS2 communications an understanding of; workspaces, packages, nodes, topics, messages, and services is required.

The root workspace for ROS2 projects is defined as a 'workspace' a single workspace will be created for this thesis project. Within the workspace are 'packages,' each package is an individual unit of software allowing the overall code to be developed modularly with structured organisation. Each package should have a defined purpose for example perception, mapping, path planning and control will all be separate packages.

Within a package can be a single or multiple ‘nodes’ each node is designed to be a processing algorithm designed for a specific task. A publish/subscribe model is utilised for communication channels known as ‘topics.’ A node can subscribe to a topic and receive ‘messages,’ alternatively a node can publish to a topic and send messages. As an example, a single node can be developed to read the LiDAR sensor and publish a LiDAR point cloud message. A separate node will read the LiDAR message, perform computations, and publish known traffic cone locations.

Nodes can subscribe to any open ros2 topic allowing communications between different packages and even different workspaces. As per Figure 12 many nodes can subscribe/publish to the same topic/s allowing one-to-many, many-to-one, and many-to-many communications. Each message is required to have a defined data structure to match the topic to which it is being published. It is also important to note that. For further information refer the referenced ROS2 documentation (ROS n.d.).

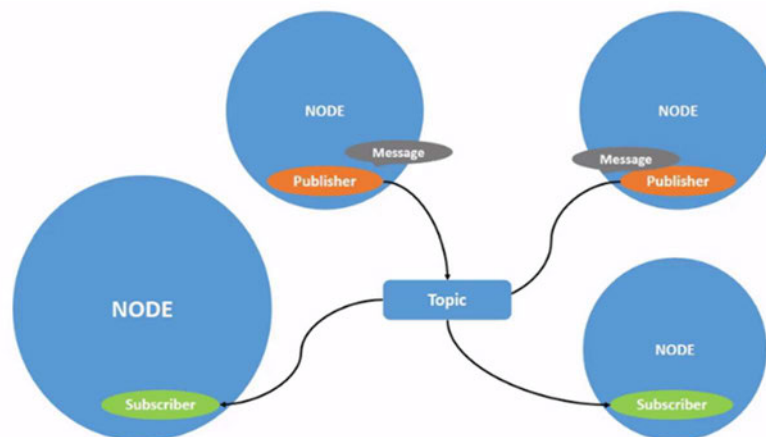


Figure 12: ROS2 Communications (ROS n.d.).

### 2.7.2 Visualisation Tools

Various visualisation tools will be utilised to assist with algorithm creation, testing and optimisation. These tools offer valuable capabilities for visualising data, analysing performance, and gain insight into system behaviour.

One widely adopted visualisation tool is MATLAB, a programming language developed and owned by MathWorks. MATLAB provides a suite of functions and libraries that facilitate algorithm development and analysis. Additionally, MATLAB benefits from standardized documentation and tutorials, contributing to its widespread adoption in the academic and industrial domains (MathWorks n.d.).



Another popular visualisation tool is Matplotlib, a plotting tool in the python language. Matplotlib is widely used for data visualisation in software development projects and is a viable open-source alternative to MATLAB. The Matplotlib community offer and maintain a range of tutorials and resources to help users learn and utilise the libraries effectively (matplotlib n.d.).

Another visualisation tool likely to be used within this research project is ROS visualisation package RViz. RViz is ROS graphical interface and is an open-source 3D visualisation environment. RViz is supported by ROS community, there is a range of tutorials and documentation available on the ROS website (ROS n.d.). It is important to note that RViz is not currently fully compatible with ROS2, and this may result in issues or crashes.

These visualisation tools are the most common for robotic/autonomous development and will be utilised throughout the project to assist with algorithm creation, testing and optimisation.

### 2.7.3 Git / Software Storage / Version Control

Effective software storage and version control are crucial for any software development project. For this project it will be a critical step in meeting Project Aim 3: Establish UniSQ FSAE Autonomous Development Platform. A software storage platform should allow for software back-up to prevent loss of work if there is an issue with the local system. A version control system should allow collaboration between multiple team members, ability to view and revert to previous versions and more. There are various version control systems including SVN and Mercurial however Git is by far the most widely used. Figure 13 shows a survey completed by software developers, showing more than 90% of developers are using Git for software storage and version control. As a result of this Git was the chosen software storage and version control system for this project.



Figure 13: Version Control Survey (Donovan 2023)

Git is an open-source distributed version control system that allows multiple developers to work on the same codebase at the same time. Git provides a record of all changes made to the codebase, allowing for easy tracking and identification of bugs and issues. Additionally, Git allows for collaboration and easy integration of code changes from multiple developers. To store the Git repository, the USQ Student Gitea instance (Gitea n.d.) was used.

Familiarisation with Git and Gitea will be essential for version control and future collaboration.

#### **2.7.4 Operating Systems**

An operating system is the software program that allows users to interact with computing hardware. The key functions of an operating system are hardware management, process management, memory management, file system management, device management and user interface. An operating system is required for any computing device, however, for the purpose of this research project only personal computer (PC) operating systems are relevant.

There are many different operating systems available for PC's including windows, Linux, macOS etc. The simulator and ROS2 require a Linux based operating system. Linux operating systems come in a range of different distributions however Ubuntu is the only distribution supported by the simulator and was used for this project. The report writing, and Gantt chart software used require a windows operating system.

It is possible to have both windows and ubuntu operating on the same PC, this is known as dual-booting (Hoffman 2014). Dual-booting requires partitioning the PC hard drive into two separate partitions. This allows one part of the hard drive to be devoted to windows and another part of the hard drive to be devoted to Linux.

#### **2.7.5 Programming Languages**

The selection of programming languages in any software development project plays a crucial role in achieving the desired outcomes. For autonomous vehicle development, various programming languages are utilised to address different aspects of the project.

In order to work with the visualisation tools, a solid understanding of both MATLAB and Python languages is required. These languages offer powerful capabilities for visualising data and analysing performance, making them valuable tools for this project.



For robotics frameworks, a combination of C++ and Python is commonly used. C++ is preferred for performance-critical tasks, while Python allows for rapid prototyping and algorithm development. These languages ensure compatibility with existing frameworks and benefit from extensive resources and community support.

Knowledge of shell scripting is essential for utilising the Linux terminal.

The simulator supports python integration and ROS2 communications. Proficiency in python and creating ROS2 packages with both python and C++ will be required for this project.

Where possible C++ will be selected as the primary language to maintain standardisation and ensure optimal performance across USQ FSEA projects. While lacking standardised documentation, online tutorials and guides are available for learning and development (cplusplus n.d.) (W3Schools n.d.).

### 3 Project Design

This section provides an overview of the project including; risk management, resources, timeline, and project planning. Due to the broad scope of the project, detailed methodology has been removed from this section and placed in section 4 ‘Autonomous Software Development’. Section 4 combines the detailed methodology, results, and discussion grouped by the three major contributions; Simulator Setup, Complete a Basic Lap of Simulation Track, Perception (LiDAR).

#### 3.1 Safety, Ethics and Environment

This project focuses on developing software within a simulated environment, which inherently reduces the scope of ethical and environmental considerations. This project is not directly impacted by the broader ethical and environmental considerations associated with general autonomous driving.

Given the nature of working in a simulated environment, the potential risks to personnel and equipment are expected to be minimal. A UniSQ risk management plan (RMP) has been implemented to address any potential personnel and property risks. The details of the RMP can be found in Appendix A2 - Personal and Property RMP. The assessment of residual risks, following the implementation of basic controls, indicates that the overall risk level associated with this project is low.

#### 3.2 Project Risk

In addition to personal and property risk there is potential for project-related risks, such as project delays or non-completion. A risk matrix has been developed with the consequences adjusted to align with the project as shown in Table 4. Using this risk matrix, a basic risk analysis has been completed for project risks shown in Table 5.

**Table 4: Project Risk - Customised Risk Matrix**

Probability	Consequence				
	Insignificant Delay <1 Day	Minor Delay <1 week	Moderate Delay <1 Month	Major Delay >1 Month	Catastrophic Project Failure
Almost Certain	M	H	E	E	E
Likely	M	H	H	E	E
Possible	L	M	H	H	H
Unlikely	L	L	M	M	H
Rare	L	L	L	L	M

**Table 5: Project Risk Analysis**

Hazard	Original Prob	Original Cons	Original Rating	Controls	Revised Prob	Revised Cons	Revised Rating
Loss of code	Possible	Major	E	Use of GitHub to back up code. Daily Local back up	Unlikely	Minor	L
Loss of report	Possible	Moderate	H	Write a program to automatically back up report daily	Unlikely	Minor	L
Code error resulting in code no longer working	Almost Certain	Minor	H	Use of GitHub to back up code. Daily Local back up	Unlikely	Insignificant	L
Family Concerns	Likely	Moderate	H	Expected family concerns Feb – May, start work in 2022	Possible	Minor	M
Supervisor Delays	Possible	Minor	M	Start work in 2022, regular communication with supervisor	Possible	Minor	M

### 3.3 Resources

The only resource required for this project is a Personal Computer (PC) that meets the requirements for the simulator. The requirements are outlined as (Formula-Student-Driverless-Community n.d.):

- 8 core 2.3Ghz CPU
- 12 GB memory
- 30GB free SSD storage (120GB when building the unreal project from source)
- Recent NVidia card with Vulkan support and 3 GB of memory.

The RAM, GPU and memory were required to be updated on the available PC. Table 6 outlines the completed upgrades.

**Table 6: Proposed PC Upgrades**

	Brand	Model	Cost
RAM	Corsair	CMK32GX4M2A2666C16	\$ 190.00
GPU	Gigabyte	GTX1650	\$ 280.00
Memory	Samsung	MZ-V8P1T0BW	\$ 220.00
<b>Total</b>			<b>\$ 690.00</b>

### 3.4 Timeline

A project Gantt chart was used for initial planning and to track the project through to completion. A screenshot of the timeline during the project is provided in Appendix A3 - Project Gantt Chart.

### 3.5 Project Tasks

To meet the project aims, the project was completed in seven phases:

1. Preparation
2. Familiarisation
3. Develop software platform
4. Develop / source code to perform basic functions
5. Develop advanced perception
6. Combine and optimise codes.
7. Present and handover

A breakdown of tasks for each phase is presented in Table 7.

**Table 7: Breakdown of Project Phases**

Phase 1. Preparation	
1 a)	Literature review
1 b)	Upgrade PC to meet simulator specifications
1 c)	Create Linux partition on PC
1 d)	Download simulation software
Phase 2. Familiarisation	
2 a)	Familiarisation with simulation software
2 b)	Familiarisation with ROS
2 c)	Familiarisation with C++
2 d)	Familiarisation with GitHub
Phase 3. Develop Software Platform	
3 a)	Communicate with FSAE team on existing progress
3 b)	Open a USQ race team GitHub Account
3 c)	Set up GitHub account with clear purpose and instructions
Phase 4. Develop / Source Code to Perform Basic Functions	
4 a)	Develop / source basic traffic cone detection
4 b)	Develop / source basic vehicle control
4 c)	Complete a basic lap of simulation track
Phase 5. Develop Advanced Perception	
5 a)	Develop advanced perception algorithm and code
Phase 6. Combine and Optimise	
6 a)	Develop an overall code combining phase 4 and 5
6 b)	Test and optimise combined code
Phase 7. Present and Handover	
7 a)	Prepare work for handover to USQ FSAE team
7 b)	Present work to USQ FSAE team
7 c)	Final handover to USQ FSAE team



### **3.6 Assumptions and Limitations**

All work completed within this project was done with the intention of being able to utilise the software on a real-world FSAE vehicle. It is important to note some of the limitations of the simulator and the assumptions used which may present challenges during real-world integration.

Computing performance is a possible limitation of the simulator development. All computing was done on the authors PC which may outperform or underperform available computational hardware on the vehicle. Testing will be required on the vehicle hardware prior to integration. Performance optimisation including methods such as parallel processing may be required to ensure hardware is compatible with autonomous software.

Sensor noise and imperfect sensor readings is another important consideration. The simulator generates a noise for GPS, GSS and IMU sensors however this may not represent real-world sensor noise. Imperfect real-world sensor readings such as variations in ground height due to grass or other imperfections may impact autonomous software functionality. Testing and parameter adjusting will be required to overcome this.

Finally, the simulator vehicles dynamic model may not accurately represent the dynamics of the UniSQ race teams' vehicle. Testing and comparison between the simulator dynamics and the real-world vehicle dynamics will be required during integration.



## 4 Autonomous Software Development

The detailed methodology, results and discussion will be organised under the heading of "Autonomous Software Development". Each major step of the software development process has been divided into separate subheadings. This structure allows for a clear presentation of the methodology used, the obtained results, and the discussion of any necessary adjustments or customisations made to meet the project requirements.

### 4.1 Simulator Setup

#### 4.1.1 Design & Methodology - PC Upgrades

The first phase of the simulator setup was to upgrade the PC to meet the required specifications of the simulator as per Table 6. The GPU, RAM and memory are modular components designed to be easily replaced. The power source was unplugged and each of the components was replaced.

The upgraded components were found to sufficiently run the simulator however, the GPU appeared to overheat after approximately 2 minutes of simulator run time. The PC case was removed and an external 12v fan used for external cooling. Figure 14 Shows the upgraded PC components.

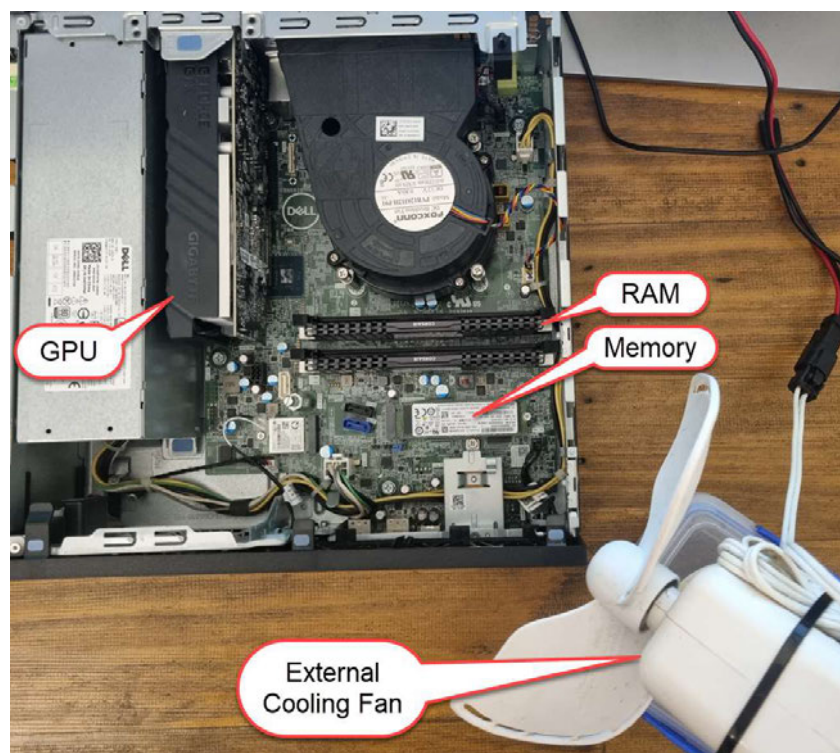


Figure 14: Upgraded PC Components



#### **4.1.2 Design & Methodology - Operating System (Linux Installation)**

A Linux operating system was required to be installed on the PC to run ROS2 and the simulator. It was decided to install the latest fully supported version of Linux Ubuntu 22.04. Initially the PC's original hard drive was partitioned and Linux was installed alongside windows as a dual boot.

The installation process provided in the Ubuntu installation documentation was followed step by step to complete the installation (Ubuntu n.d.).

After several crashes of Linux requiring reboot and a redownload of the simulator it was decided to purchase an additional hard drive and install Linux as the primary operating system on the PC. This required an adjustment to the resources required in Table 6 and the replacement of the modular hard drive.

After installing Ubuntu 22.04 as the primary operating system the PC functioned as expected and no further re-formats were required. It is expected this is a symptom of the PC being used and a dual boot may still be a preferred option for future projects on a different PC.

#### **4.1.3 Design & Methodology - Running the Simulator**

Initially the version of the simulator (V2.1.0) was not compatible with Ubuntu 22.04. This was a result of a remote procedure call server mismatch, and was fixed with the V2.2.0 update completed by the Formula Student Simulator community (Formula-Student-Driverless-Community 2022).

There are several required applications and extensions for the simulator to run on a new Ubuntu 22.04 install. A short cheat sheet has been developed for future FSAE students to reference. Refer Appendix A4 - Running the Simulator – Cheat Sheet for details on running the simulator.

#### **4.1.4 Design & Methodology – Software Storage & Version Control**

At the start of the project a Git repository was created for the ros2 workspace. This will provide backup in the event of local PC issues and allow for future work and collaboration. At the time of writing access to USQGit is not available and the project has been backed up on the authors GitHub account.

Section 5 summarises the git repository at completion of the project. The link for the git repository is available in section 6.2.

#### 4.1.5 Results & Discussion

With the overheating concerns of the GPU a test was established to determine the severity of the issue. The simulator was run with the basic lap autonomous control software (see Section 0) and the temperature of the GPU was recorded using the NVidia Linux terminal command:

`nvidia-smi -q -d temperature`

As per Figure 15 This command provided the GPU temperature, max operating temperature and current temperature.

```

alistair@alistair-OptiPlex-7060:~$ nvidia-smi -q -d temperature

=====NVSMI LOG=====

Timestamp                : Mon Jul 24 09:46:06 202
3
Driver Version           : 515.105.01
CUDA Version             : 11.7

Attached GPUs            : 1
GPU 00000000:04:00.0
  Temperature
    GPU Current Temp      : 36 C
    GPU Shutdown Temp    : 97 C
    GPU Slowdown Temp    : 94 C
    GPU Max Operating Temp : 92 C
    GPU Target Temperature : 83 C
    Memory Current Temp   : N/A
    Memory Max Operating Temp : N/A

```

Figure 15: GPU Parameters

The test was conducted with the external cooling removed and the PC case closed. The temperature of the GPU was taken at one-minute intervals. The results from Figure 16 show that the GPU is operating within its ranges and no external cooling is required.

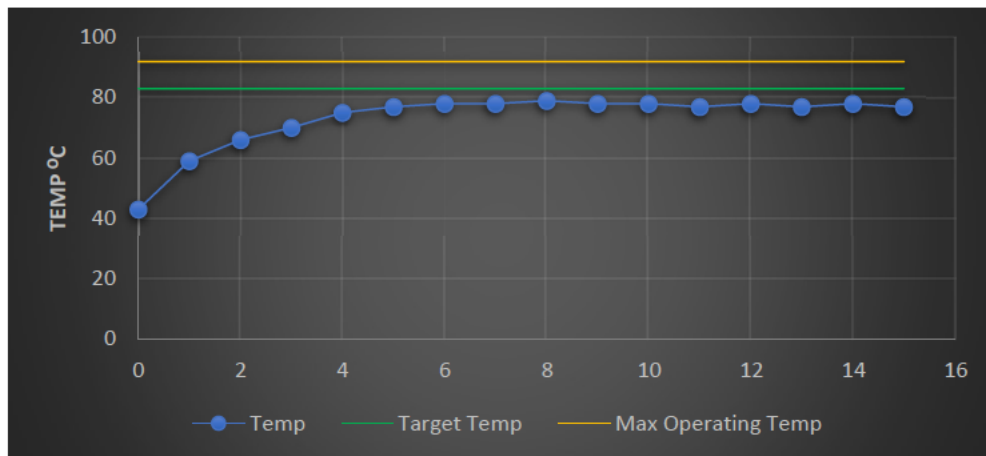
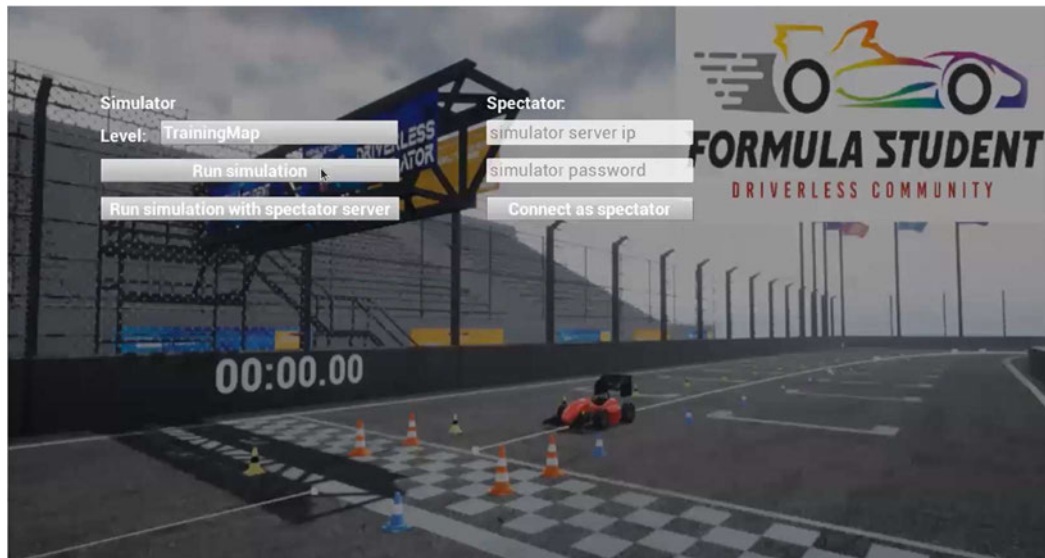


Figure 16: Recorded GPU Temperatures

#### 4.1.6 Conclusion

The result of the simulator set up was eventually successful, with the simulator running on the upgraded PC. Figure 17 shows a screenshot of the simulator running successfully.



**Figure 17: Simulator Running**

Upgrading the PC and successfully running the preferred simulator satisfied Project Aim 1:

Set up and Test Simulation Environment, by meeting the three objectives:

1. Research available simulation software and determine if existing simulators exist.
2. Determine preferred simulator from existing/new solutions.
3. Develop/Install and trial simulation environment.

Creation of Git repository also partly satisfied Project Aim 3: Establish UniSQ FSAE Autonomous Development Platform, by meeting objective 1. Establish a version control and collaborative repository. Development of Appendix A4 - Running the Simulator – Cheat Sheet is also a contribution to objective 2. Compile all completed objectives, algorithms, and relevant documentation into a handover package.

## 4.2 Complete a Basic Lap of Simulation Track

To be able to design and implement effective autonomous vehicle algorithms the car first must be able to drive a lap of the track. This can either be done manually with keyboard controls or a basic autonomous control algorithm can be developed. It was decided a basic control algorithm will be better suited for repetitive testing and to provide a benchmark lap time for all further design stages. The simulator repository is provided with Python examples that will complete a lap of the track.

### 4.2.1 Current State Analysis - Provided python example (simulator repository)

The provided example used a rudimentary bang-bang control. Simply if there are more traffic cones on the left of the vehicle turn 30% left and if there are more traffic cones on the right of the vehicle turn 30% right. The example attempted to maintain a constant vehicle speed, simply applying the throttle if below this speed. In the python example a constant speed of 4 m/s was maintained.

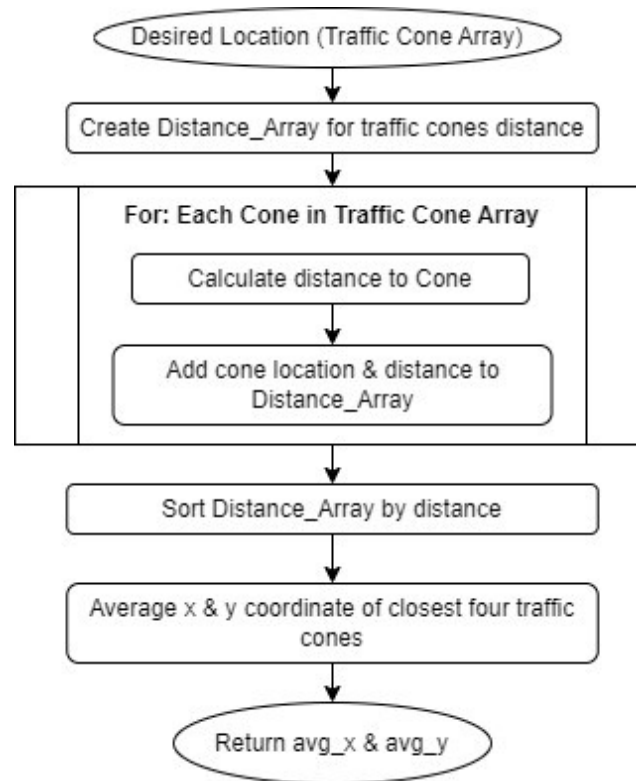
As it was decided to use C++ and ROS2 for all software design tasks the first task was to replicate the supplied python examples in C++. While this effectively completed a lap of the track it was slow and erratic method of control that would provide challenges for further autonomous development.

To improve performance of the benchmark lap while maintaining basic operation it was decided to implement a basic path planning algorithm and proportion steering control. The vehicle speed control remained as per the python example.

### 4.2.2 Design & Methodology – Basic Path Planning

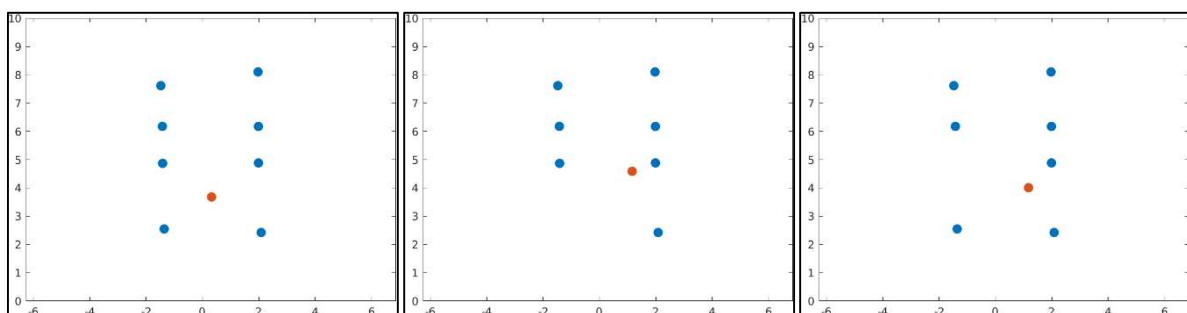
A typical path planning algorithm would output a series of x & y coordinates detailing the entire race path. From this race path a desired location for the vehicle can be extracted, this would either be a point closest to the vehicle or a set distance in front of the vehicle.

Development of a race path is only possible with vehicle localisation and track mapping which had not yet been completed. To overcome this a simplified path planning algorithm was developed with a desired location calculated by averaging the closest four traffic cones. The flowchart in Figure 18 outlines the path planning process and the orange dots in Figure 19 visualise the desired location. Noting this simplification does not allow for throttle and braking control development as a race path with velocity profile would.



**Figure 18: Basic Path Planning Flowchart**

Two common errors were found with this path planning method, at times there are less than four visible traffic cones and the traffic cones not evenly being perceived on the LHS and RHS of the track. Figure 19 show the desired location working correctly (left) and the errors where the cones are not evenly placed on LHS and RHS (centre and right).



**Figure 19: Desired Location Errors**

To overcome this simple error checking was implemented. If less than four traffic cones were detected the desired location was not updated. The principle that the desired location should be in the middle of the track and hence not within a certain proximity of a traffic cone was utilised to remove remaining errors. Figure 20 shows the error radius around the traffic cones with the desired location in the correct position (left) and where the cones are not evenly placed on LHS and RHS (centre and right). If the desired location was within one metre of a traffic cone it was not updated.

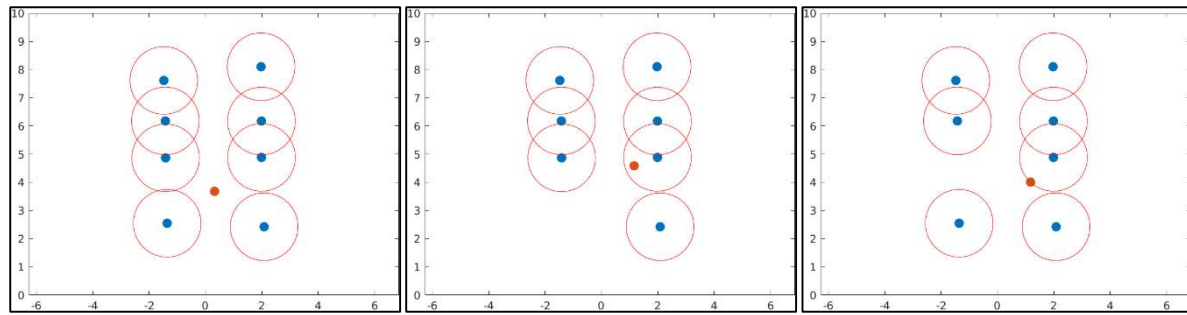


Figure 20: Desired Location Error Radius

#### 4.2.3 Design & Methodology – Basic Proportional Steering

With a desired location available a proportion steering control was implemented based off the angle between the vehicle and the desired location as per section 2.6.. Simply the steering output was proportional to the angle to the desired location with a gain value of  $K_p$ .  $K_p$  was roughly tuned by trial and error to  $K_p = 1.5$ .

It would have been possible to further optimise this parameter or introduce methods such as variability with vehicle speed, however, an improved steering algorithm such as Pure Pursuit or Stanley controller are expected as future work once a race path is available.

The vehicle speed control remained the same as the supplied python example, with a maximum constant speed maintained. The maximum speed was optimised by trial and error as per the results.

#### 4.2.4 Results & Discussion

The most important result of this section is qualitative question, was basic self-driving demonstrated? A completion of multiple laps of the race track has demonstrated basic self-driving and this section was a success.

To further analyse the results the lap time was used as a quantifiable criterion, considering traffic cone (2 second) and off course (10 second) penalties as per the formula student rules (FSG 2023). This was a secondary criterion for further analysis and hence, significant time was not spent optimising steering and throttle gain or other parameters. More advanced algorithms will make these parameters and the associated algorithms redundant in relation to overall autonomous development.

The average lap time of the python example provided with the simulator repository is 160.37 s as per Table 8.

**Table 8: Simulator Example Lap Times**

Lap	Lap Time (s)	Cones Hit	Off course	Total
1	155.41	0	0	155.41
2	157.13	4	0	165.13
3	156.56	2	0	160.56
Average				160.37

The average lap time after the path planning and steering control improvements, with the same 4 m/s max speed as the provided example was 177.15 as per Table 9.

**Table 9: Basic Lap Algorithm Lap Times, max speed = 4 m/s**

Lap	Lap Time (s)	Cones Hit	Off course	Total
1	169.07	1	0	171.07
2	175.43	1	0	177.43
3	174.95	4	0	182.95
Average				177.15

To optimise the maximum speed, the speed was increased gradually until the vehicle would no longer stay on course. The average lap time after the path planning and steering control improvements, after optimising the max speed to 6.5 m/s is 113.17 as per Table 10.

**Table 10: Basic Lap Algorithm Lap Times, max speed = 6.5 m/s**

Lap	Lap Time (s)	Cones Hit	Off course	Total
1	110.26	2	0	114.26
2	108.78	3	0	114.78
3	110.46	0	0	110.46
Average				113.17

It is important to note that the python example provided with the simulator performs far better than the C++ and ros2 example despite being a replicated control algorithm. It was also noticed the frame updated rate of the simulator reduced once the Ros bridge was launched. It is hypothesised the AirSim Ros wrapper effects the frame rate of the simulation. Computing power may affect this issue and produce varying lap times for future work. However, testing will have to be performed to confirm this.



#### 4.2.5 Conclusion

The Development of the basic lap algorithm was a success and will allow further development of the autonomous vehicle control. The benchmark lap time was improved from 160.37 s from the provided example to 113.17 s.

Upgrading the PC and successfully running the preferred simulator satisfied Project Aim 2: Demonstrate Basic Self-Driving, by meeting the three objectives:

1. Research, develop and test a basic perception algorithm.
2. Research, develop and test a basic Motion Estimation and Mapping algorithm.
3. Research, develop and test a basic vehicle control algorithm.

There are two Ros2 packages created for this section of the design:

1. `autonomous_example` – the C++/Ros2 replication of the provided example.  
The `autonomous_example` can be run without a perceived traffic cone graph with the **`autonomous_example.launch.py`** launch file or with a perceived traffic cone graph with the **`graph_autonomous_example.launch.py`** launch file.
2. `basic_lap` – this includes the improved path planning algorithm and the proportional steering control.  
the `basic_lap` can be run without a perceived traffic cone graph with the **`basic_lap.launch.py`** launch file or with a perceived traffic cone graph with the **`graph_basic_lap.launch.py`** launch file.



## 4.3 Perception (LiDAR)

### 4.3.1 Current State Analysis – Perception Algorithm

The perception algorithm supplied with the simulator repository is a very basic LiDAR based algorithm. The LiDAR is required to be programmed with a single laser with a vertical field of view of  $0^\circ$ , 500 points per scan, 10 rotations per second and 7m range. This does not represent a real-world LiDAR and has inherent problems with improved algorithm development.

The 7m range of the supplied perception algorithm is not suitable to autonomous race vehicles operating at speed. Formula student vehicles are capable of speeds up to 30m/s this would result in a perception forecast of only 0.23s. The biggest problem with the supplied perception algorithm is the single laser with a  $0^\circ$  field of view. During cornering the centrifugal force results in a body roll, this results in the laser picking up sections of the ground on the down angle side of the vehicle. The basic algorithm then groups the ground points and assumes them as traffic cones. Figure 21 shows a screenshot of this error, as the vehicle rolls towards the outside of the corner the ground is picked up as a row of traffic cones.

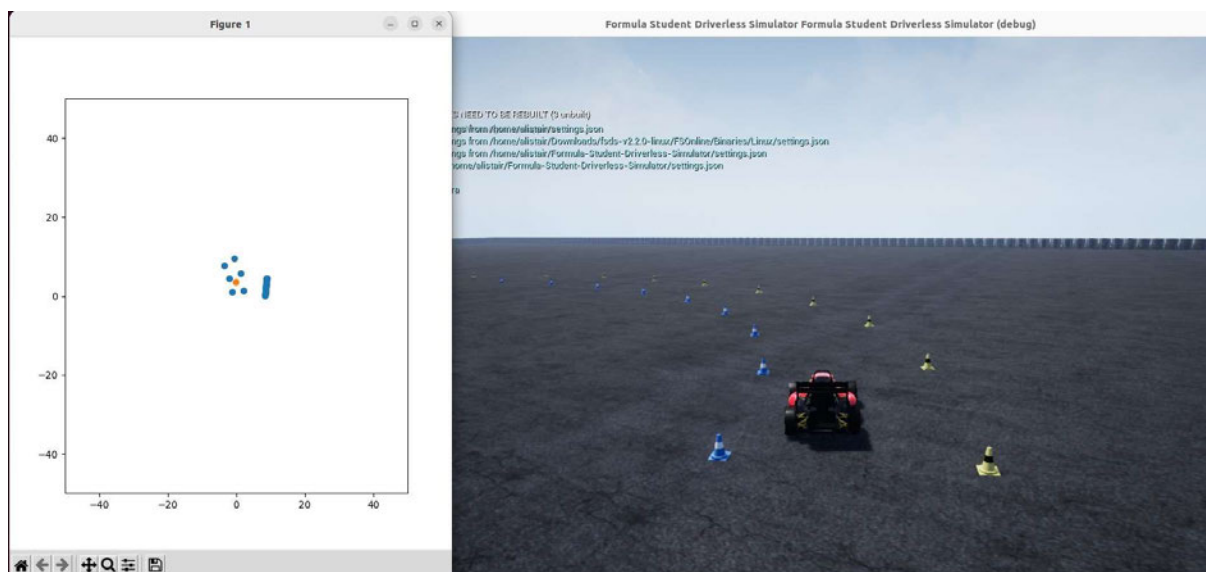


Figure 21: Simulator Example Perception Error

Due to this it was required to improve the perception algorithm before developing mapping and path planning. The focus of the perception improvements was to align the simulation to the available UniSQ FSAE teams VLP-16.

The datasheet for the VLP-16 was referenced and the simulator LiDAR properties were modified to match. VLP-16 set up and properties are further detailed in section 4.3.2.

### 4.3.2 Design & Methodology - VLP-16 LiDAR Setup

The simulator parameters were required to be modified to best represent the available VLP-16 LiDAR. A literature review has been completed on the VLP-16 in section 2.4.1. This set up is based on the VLP-16 datasheet (Velodyne 2019). The simulator LiDAR options are as per Figure 22.

```
"Lidar1": {
  "SensorType": 6,
  "Enabled": true,
  "X": 0, "Y": 0, "Z": -1,
  "Roll": 0, "Pitch": 0, "Yaw": 0,
  "NumberOfLasers": 7,
  "PointsPerScan": 2000,
  "RotationsPerSecond": 20,
  "VerticalFOVUpper": 0,
  "VerticalFOVLower": -25,
  "HorizontalFOVStart": 0,
  "HorizontalFOVEnd": 90,
  "DrawDebugPoints": false
}
```

Figure 22: Simulator LiDAR Options (Formula-Student-Driverless-Community n.d.)

- X, Y & Z is the position of the LiDAR relative to the vehicle centre. This was chosen as 1.2, 0, 0.2 respectively.
- Roll, Pitch & Yaw are the rotation of the LiDAR relative to the vehicle and are all set as 0.
- Number of lasers is determined by the LiDAR construction, for the VLP-16 this is 16.
- Points per scan is dependent on the VLP-16 frequency settings.
- Rotations per second is dependent on the VLP-16 frequency settings.
- Vertical FOV Upper & Lower are determined by LiDAR construction, for the VLP-16 this is 15° to -15°
- Horizontal FOV Start and End are set from -90° to 90° to view 180° in front of the vehicle.

The VLP-16 frequency settings can be selected from 300, 600, 900 & 1200 RPM. There are several factors to consider when determining LiDAR frequency. The first is the perception update rate, that is how often or more importantly how far the car travels between perception updates. Table 11 shows the distance travelled for each frequency setting at different vehicle speeds.

Table 11: Distance Travelled (m) at Different LiDAR Frequency and Vehicle Speeds

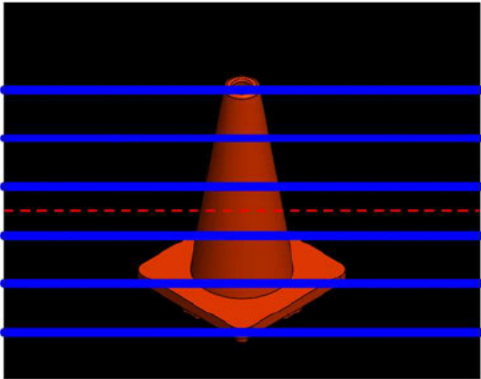
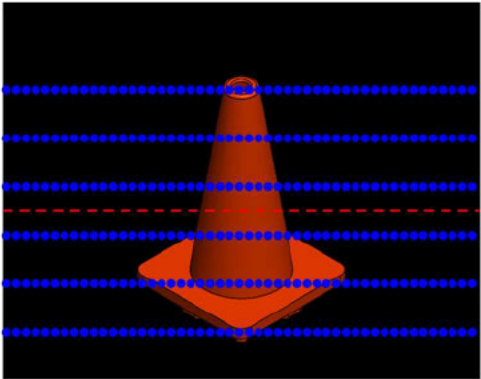
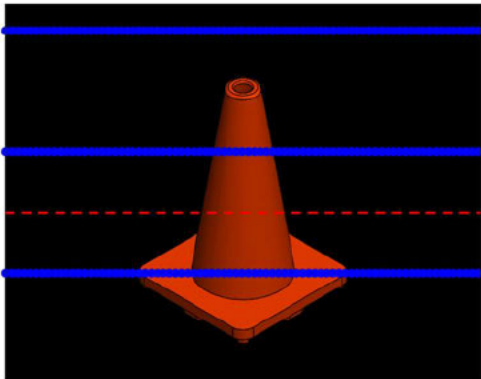
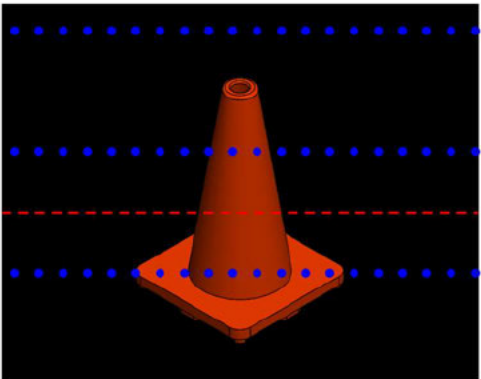
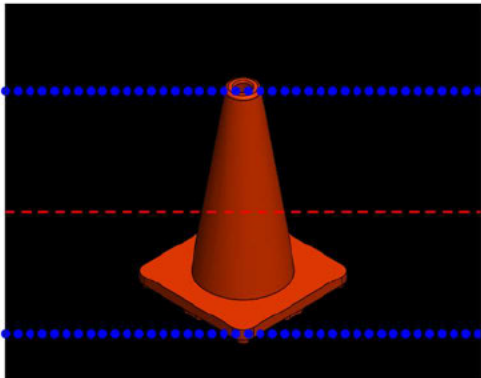
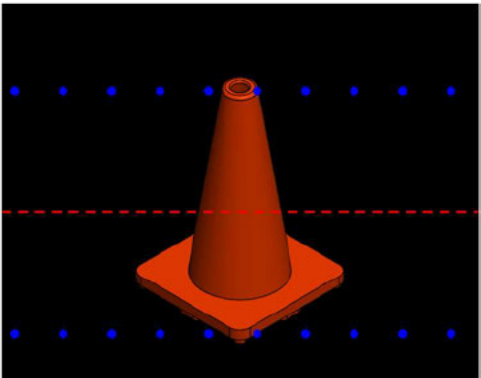
Hz	5	10	15	20
10m/s	2	1	0.67	0.5
20m/s	4	2	1.33	1
30m/s	6	3	2	1.5

Another major consideration is the horizontal resolution. Higher resolution is desired as the vehicle is trying to perceive objects at greater distances. Table 13 shows a visualisation of the horizontal resolution at maximum and minimum frequency settings. Table 12 shows the number of points, the angular resolution, and the horizontal resolution at each frequency setting.

**Table 12: VLP-16 LiDAR Resolution and Points**

RPM	300	600	900	1200
Hz	5	10	15	20
Angular Resolution (deg)	0.1	0.2	0.3	0.4
Horizontal Resolution @ 10 m (mm)	17.5	35	52.4	69.8
Points/Channel	1800	900	600	450
Points/Scan	28800	14400	9600	7200

**Table 13: Visualisation of Horizontal Resolution at Min and Max Frequency Settings**

	Angular Resolution = 0.1	Angular Resolution = 0.4
2m		
5m		
10m		

Considering the perception update rate and horizontal resolution a LiDAR frequency of 10 Hz was chosen. The final settings for the simulator LiDAR are as per Figure 23.

```
"Lidar2": {
  "SensorType": 6,
  "Enabled": true,
  "X": 1.20, "Y": 0, "Z": 0.2,
  "Roll": 0, "Pitch": 0, "Yaw" : 0,
  "NumberOfLasers": 16,
  "PointsPerScan": 14400,
  "VerticalFOVUpper": 15,
  "VerticalFOVLower": -15,
  "HorizontalFOVStart": -90,
  "HorizontalFOVEnd": 90,
  "RotationsPerSecond": 10,
  "DrawDebugPoints": false
}
```

Figure 23: Improved Perception LiDAR settings

#### 4.3.3 Design & Methodology – Perception Algorithm

With the simulator LiDAR replicating the VLP-16, a more advanced LiDAR perception algorithm was required. A bachelor thesis by Daniel Storc at the Czech Technical University in Prague ‘*Detection of Traffic Cones from Lidar Point Clouds*’, was utilised as a guide for the development of the perception algorithm (Storc 2022).

First a main code structure was developed with four sub functions as per the flowchart in Figure 24.

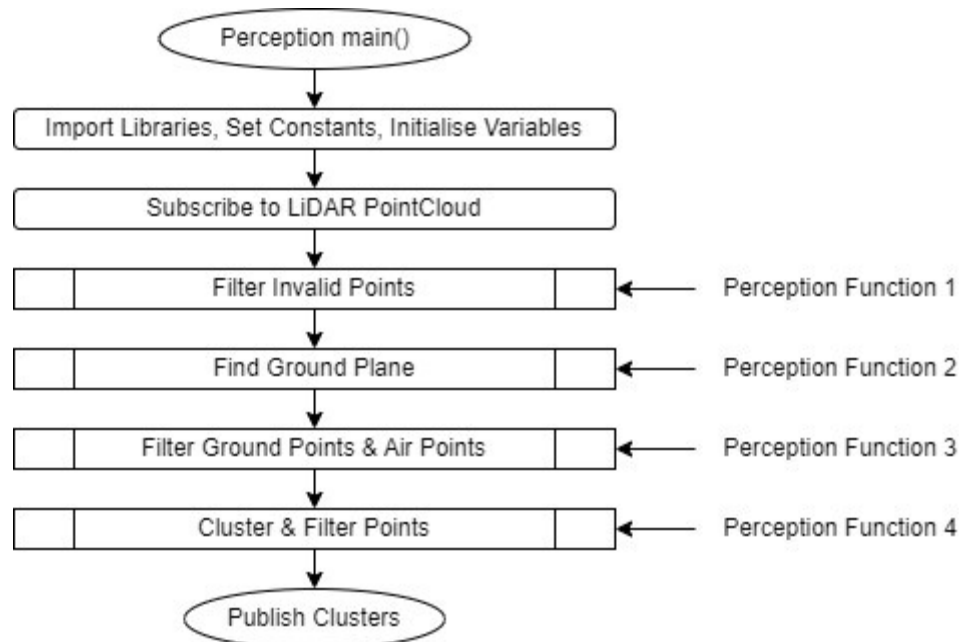


Figure 24: Perception Flowchart - Main Code

### Perception Function 1: Filter Invalid Points

The primary purpose of Perception Function 1 is to remove all the invalid LiDAR points, that is points that did not reflect off an object within the detection range and hence return a zero value. As per the flowchart in Figure 25 a for loop is set up for each of the points within the point cloud, the x, y & z coordinates are extracted from the point cloud, and if the x, y & z values do not equal zero the points are added to the points array.

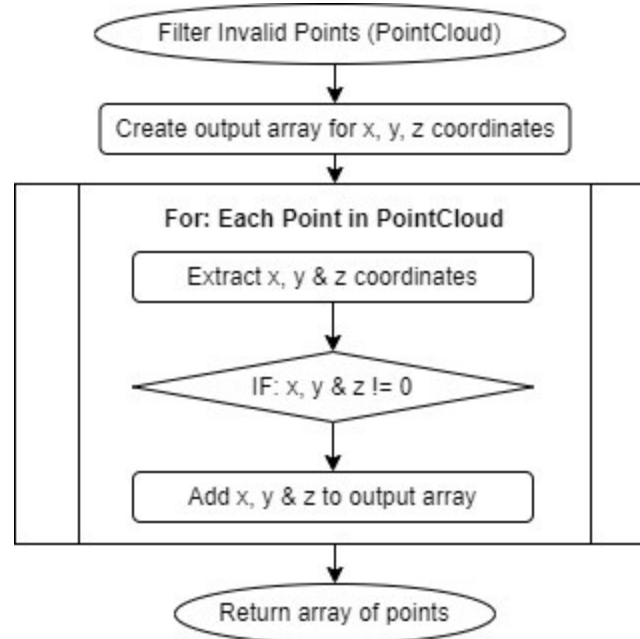


Figure 25: Perception Flowchart - Filter Invalid Points

### Perception Function 2: Ground Plane Equations

A ground plane, in the form of  $Ax + By + Cz + D = 0$ , can be fit to three points  $P1 = [x1, y1, z1]$ ,  $P2 = [x2, y2, z2]$ ,  $P3 = [x3, y3, z3]$  by first calculating two vectors ( $u, v$ ) as per Equation 4.

#### Equation 4: Calculating Two Vectors for Ground Plane Detection

$$u = [x2 - x1, y2 - y1, z2 - z1]$$

$$v = [x3 - x1, y3 - y1, z3 - z1]$$

The cross product of these two vectors then provides a vector ( $n$ ) normal to the plane as per Equation 5.

#### Equation 5: Normal Vector Calculation for Ground Plane Detection

$$n(x) = u(y) * v(z) - u(z) * v(y)$$

$$n(y) = u(z) * v(x) - u(x) * v(z)$$

$$n(z) = u(x) * v(y) - u(y) * v(x)$$

Parameters A, B & C of the ground plane can then be calculated by normalising the normal vector. The normal vector is normalised by dividing the vector by its magnitude as per Equation 6.

**Equation 6: Parameter A, B & C for Ground Plane Detection**

$$mag = \sqrt{n(x)^2 + n(y)^2 + n(z)^2}$$

$$A = \frac{n(x)}{mag}; \quad B = \frac{n(y)}{mag}; \quad C = \frac{n(z)}{mag}$$

Parameter D of the ground plane can then be calculated as the distance from the plane to the origin (D) as per Equation 7.

**Equation 7: Parameter D for Ground Plane Detection**

$$D = -(A * x1 + B * y1 + C * z1)$$

### **Perception Function 2: Find Ground Plane**

The primary purpose of Perception Function 2 is to fit a ground plane to the valid points returned from Perception Function 1. This is the most complex and computationally intensive section of the perception algorithm. The intent of the ground plane function is to detect a ground plane by sampling points of the ground plane, not by fitting a plane to all of the valid points. As a result of this typical methods of fitting ground planes such as the method of least squares are not appropriate. Instead a random sample consensus algorithm was used, which is a method to estimate a model using data that contains outliers (Storc 2022).

It is safe to presume most of the valid points are on the ground plane and a small portion of points will be traffic cones, walls, and other objects. The ground plane function first takes a random sample of 300 points from the valid points. From this sample a random sample of three points is taken and a ground plane is calculated using the equations in Perception Function 2: Ground Plane Equations above.

Typical random sampling consensus algorithms use the number of data points that are inliers as criteria for model suitability. A slight variation has been implemented with the sum of errors between the remaining 297 points and the ground plane being used as the criteria. The error between the ground plane and the remaining points is calculated using

**Equation 8: Calculating Distance to Plane for Ground Plane Detection**

$$e_{cum} = \sum_0^i \frac{A * x(i) + B * y(i) + C * z(i) + D}{\sqrt{A^2 + B^2 + C^2}}$$

If the cumulative error ( $e_{cum}$ ) is less than a determined error limit the ground plane is confirmed. If  $e_{cum}$  is greater than  $e_{lim}$  the three plane points are re-sampled. This is repeated up to an iteration limit, if the iteration limit is reached without  $e_{cum} < e_{lim}$  the iteration with the smallest cumulative error is used.

Figure 26 below shows a flow chart of Perception Function 2: Find Ground Plane.

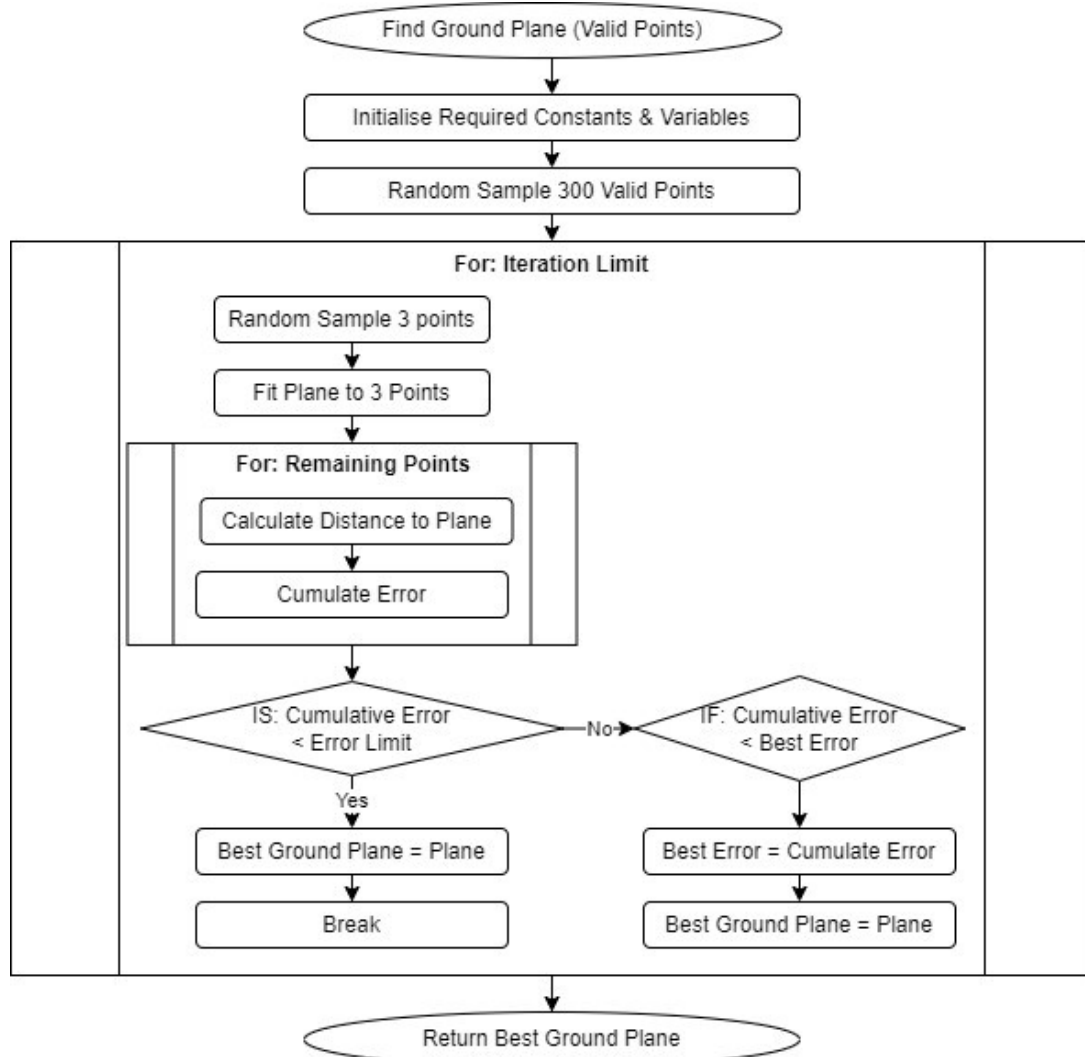


Figure 26: Perception Flowchart - Find Ground Plane

### Perception Function 2: Determination of error limit $e_{lim}$

A critical parameter of the perception algorithm is the error limit  $e_{lim}$ . As per above  $e_{lim}$  is used to confirm the ground plane compared to the cumulative error  $e_{cum}$ .

$e_{cum}$  will vary depending on the number of traffic cones in perception and the height of the perceived cones. It was assumed there would be a step change in  $e_{cum}$  if the sampled point was not on the ground plane, as all 300 sample points will be a considerable distance away from the plane. This step change would be the ideal value to set  $e_{lim}$ .

The algorithm was run over 3000 times while the vehicle moved around the track, and the error was recorded. The graph in Figure 27 shows the expected step change at a cumulative error of approximately 4m, where the remaining outliers are assumed to be incorrect sample points. Due to this an  $e_{lim} = 5m$  was determined for simulation, however this will likely need to be re-visited for integration into a real-world vehicle due to variables such as uneven ground surface not present on the simulator.

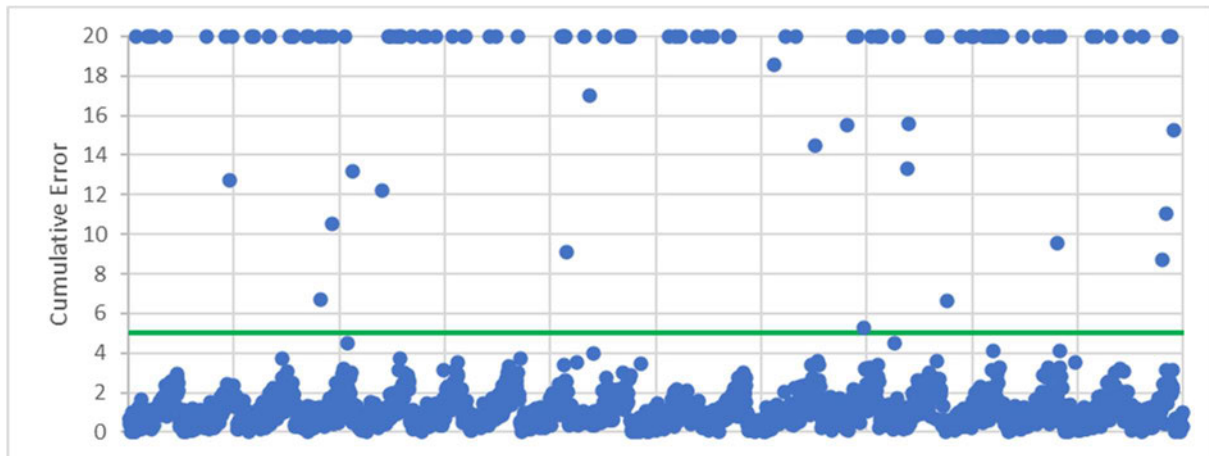


Figure 27: Cumulative Error Scatter Plot

Results in section 4.3.4 found on average a 98.2% chance a selected point is a ground point. It can then be calculated the probability of one of the three plane points landing on a traffic cone is 5.3%. In this test 5.5% of points are above  $e_{cum}$  of 5 which further verifies  $e_{lim} = 5m$ .

### Perception Function 3: Filter Ground Points and Air Points

The primary purpose of Perception Function 3 is to remove points that are close enough to the ground plane to be considered ground points ( $>$ Lower Limit) and too far away from the ground to be a traffic cone ( $>$ Upper Limit). Given the largest traffic cone has a height of 450mm an Upper Limit of 0.5m was implemented. A Lower Limit of 0.02m was found suitable for the simulator, however this may need to be adjust with a real-world vehicle.

As per the flowchart in Figure 28 the Filter Ground and Air Points function simply iterated through each point, measured the distance to the ground plane using Equation 8 from Perception Function 2.



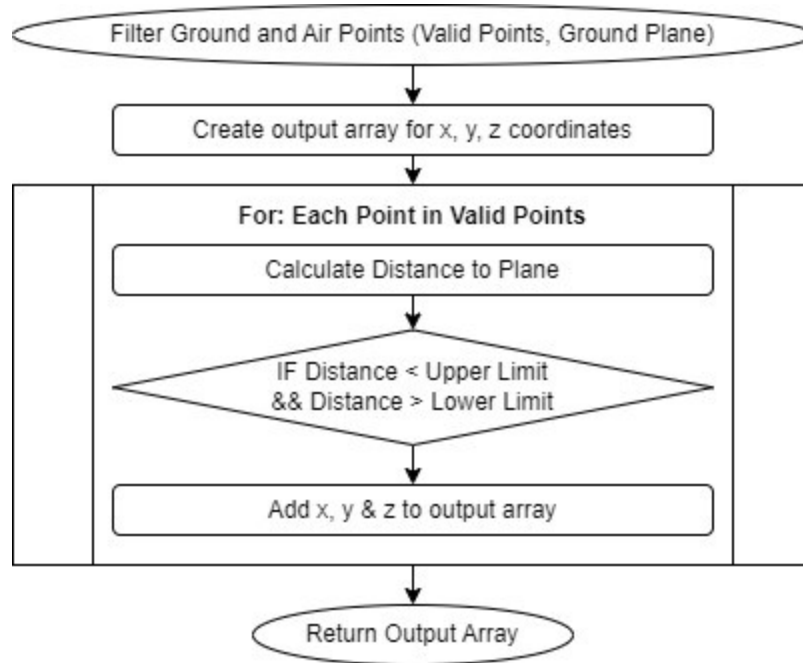


Figure 28: Perception Flowchart - Filter Ground & Air Points

#### Perception Function 4: Cluster and Filter Points

The primary purpose of Perception Function 4 is to cluster the remaining valid points, filter the clusters based on its characteristics and publish cone location based on average cluster locations.

To cluster the point the distance between each point was calculated and if within `cluster_threshold` the points were clustered. The Dimension of the largest traffic cone were utilised to determine a `cluster_threshold` of 0.45 m.

The clusters were then filtered on two criteria; if two clusters are too close to be a traffic cone and if a single cluster is too large to be a traffic cone. If two clusters were found to be within 0.5m of each other they were determined too close to be traffic cones and were not added to the final Cones message. Similarly, if a single clustered varied in dimension by more than 0.5m it was determined to be too large to be a traffic cone and not added to the final cones message.

For the remaining traffic cones each point within the cluster was averaged to provide an traffic cone location. It is important to note that the LiDAR only perceives one side of the traffic cone and as such there will be a tolerance of up to 0.2m depending on which side of the cone is viewed. Figure 29 shows a flowchart of Perception Function 4 Cluster and Filter.

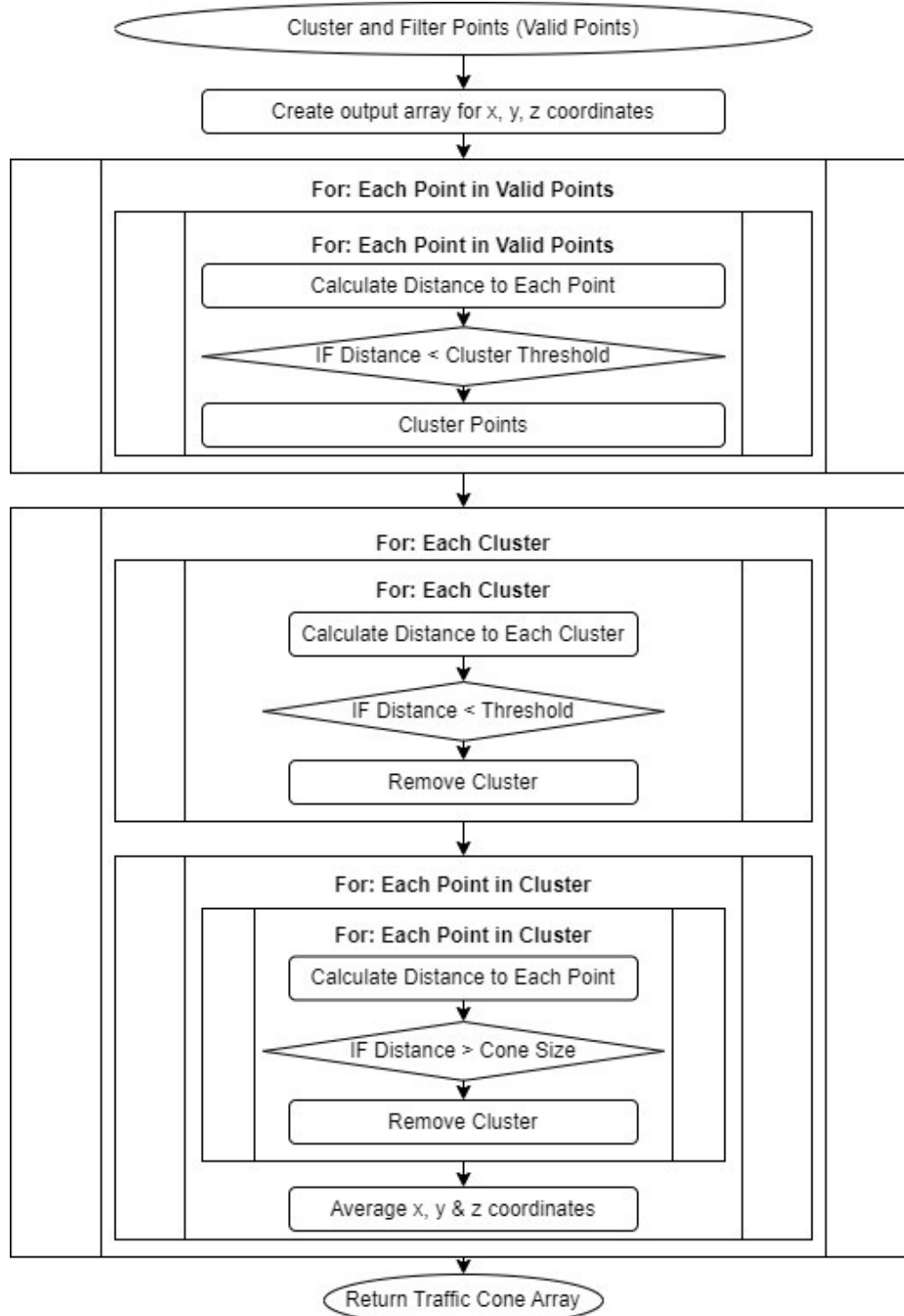


Figure 29: Perception Flowchart - Cluster &amp; Filter

#### 4.3.4 Perception Results & Discussion – Development

A LiDAR frequency of 10 Hz was chosen in section 4.3.2, it would be expected that the perception algorithm needs to run at a significantly less time than 100 ms for success.

The perception algorithm was run over 2000 times while the vehicle drove around the track, and the duration was timed. The box and whisker plots in Figure 30 show that the perception algorithm run for between 1 ms and 14 ms with few outliers. No further optimisation needs to be completed on the perception algorithm at this stage.

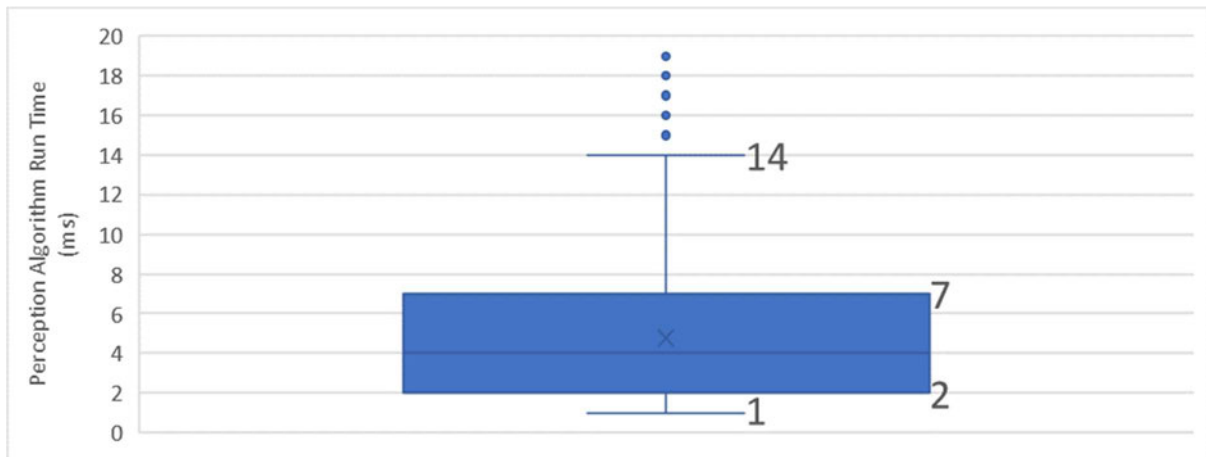


Figure 30: Perception Algorithm Run Time Tests

Testing was completed on the number of LiDAR points at each step of the algorithm. On average it was found of the 14400 LiDAR points 50% (7200) were found to be valid, 0.9% (135) were remaining after ground plane removal, and an average of 14 traffic cones were perceived. The box and whisker plots in Figure 31 show the results of this testing.

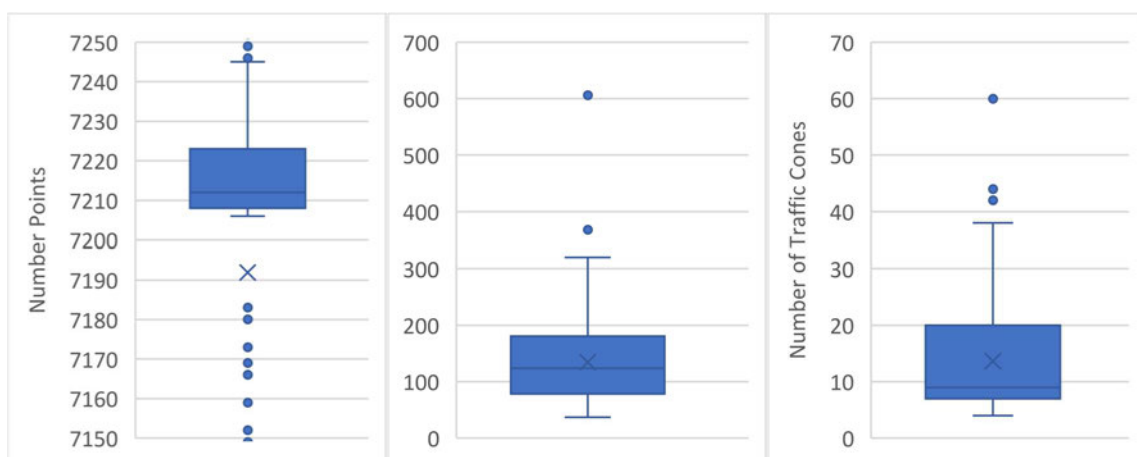
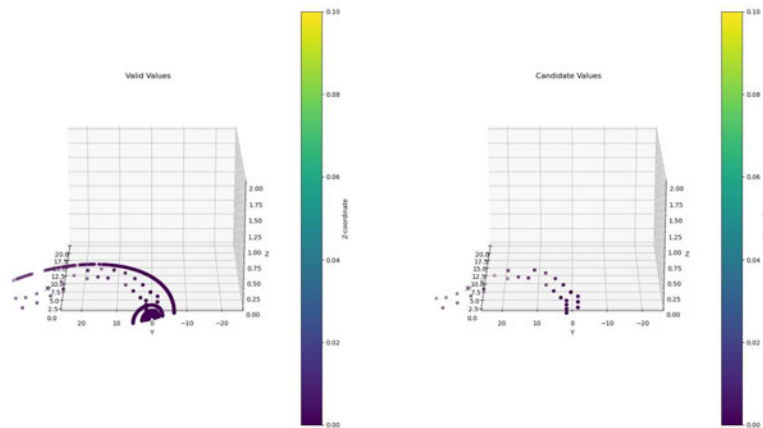


Figure 31: LiDAR Points Results

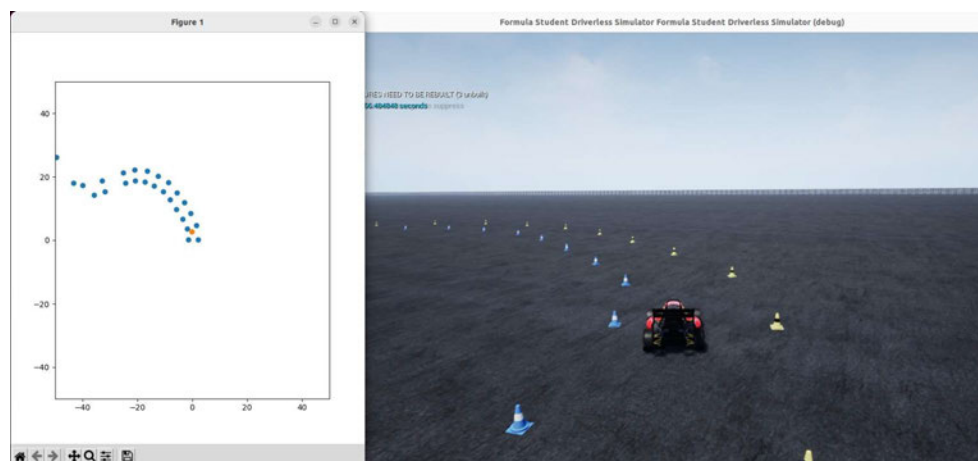
A point\_graph python ROS2 node was created to visualise the point cloud data and ground plane removal. Figure 32 Shows the 3D representation of the LiDAR points before and after ground plane removal.



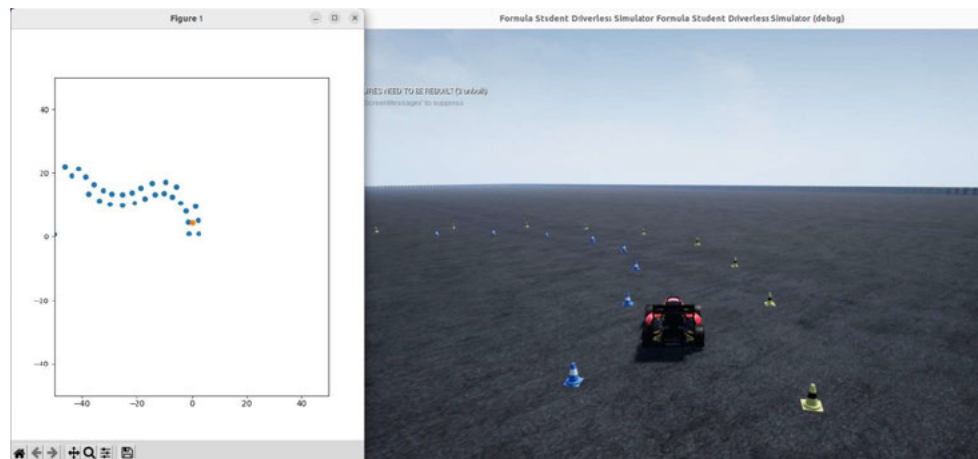
**Figure 32: Visualisation of Ground Plane Removal**

#### 4.3.5 Perception Results & Discussion – Performance

The range of the Perception has increased from 7m to greater then 20m, the accuracy has improved dramatically with no noticed false positives. Figure 33 and Figure 34 show the function of the updated perception algorithm.



**Figure 33: Improved Perception Algorithm Screenshot**



**Figure 34: Improved Perception Algorithm Screenshot**

Although the purpose of this section of the design was to improve the perception for further autonomous development, significant lap time improvements were made. As per Table 14 an approximate 5 s lap time reduction was made from the smoother vehicle operation associated with improved perception before increasing maximum speed.

**Table 14: Improved Perception Algorithm Lap Times, Max Speed = 6.5 m/s**

Lap	Lap Time (s)	Cones Hit	Off course	Total
1	112.91	0	0	112.91
2	108.02	0	0	108.02
3	103.97	0	0	103.97
Average				108.30

To optimise the maximum speed, the speed was increased gradually until the vehicle would no longer stay on course. The average lap time after the perception improvements, after optimising the max speed to 9 m/s is 82.31 s as per Table 15.

**Table 15: Improved Perception Algorithm Lap Times, Max Speed = 9 m/s**

Lap	Lap Time (s)	Cones Hit	Off course	Total
1	81.48	0	0	81.48
2	82.77	0	0	82.77
3	82.69	0	0	82.69
Average				82.31

#### 4.3.6 Conclusion

With Project Aim 2: Demonstrate Basic Self-Driving already satisfied, the development of an effective and robust perception algorithm went above the aims of this project. This will accelerate the development of the UniSQ teams' autonomous software.

Aligning the perception algorithm with the VLP-16 LiDAR available to the UniSQ team will hopefully improve integration of the autonomous software into the real-world vehicle.

The LiDAR based perception exceeded expectations within a simulated environment, allowing development of further autonomous driving processes. Testing will need to be completed on a real-world vehicle to determine the requirement of future perception improvements. These improvements could include the tuning of filtering parameters such as cluster size and distances discussed in section Perception Function 4: and the ground plane  $e_{lim}$  discussed in Perception Function 2:. The integration of camera perception is also future work to improve the perception algorithm.

The benchmark lap time was improved from 113.17 s from the basic\_lap to 82.31 s.

There was one Ros2 package created for this section of the design:

1. perception – improved LiDAR perception.

The 'perception' package can be run without a perceived traffic cone graph with the perception.launch.py launch file or with a perceived traffic cone graph with the graph\_perception.launch.py launch file.

## 4.4 System Synthesis

The Formula Student Driverless Simulator utilises python integration, ROS integration and ROS2 integration for user control. For all design tasks within this project ROS2 has been used. The simulator includes a ROS2 bridge and default topics for sensors and other data communications. Only two sensor messages were used as part of this design; /lidar/Lidar1 to read the LiDAR sensor, and /gss to read the ground speed sensor.

To bring the different stages of the design together and allow for future development and collaboration a new node and message was created at each practical step in the design. Three C++ ROS2 packages and six nodes have been developed as part of autonomous control. Additionally, a python ROS2 package with three nodes was developed for data visualisation, this assisted with development and fault finding. Table 16 shows a summary of the four packages, their associated nodes, and messages.

**Table 16: ROS2 Workspace Summary**

Package	Node	Input Message	Output Message
autonomous_example	car_controller.cpp	steering_angle throttle_pos	control_command
	cone_id.cpp	lidar/Lidar1	cone_location
	steering_angle.cpp	cone_location	steering_angle
	throttle_pos.cpp	gss	throttle_pos
basic_lap	steering_angle.cpp	cone_location	steering_angle
graphing	cone_graph.py	cone_location	2D plot
	map_graph.py	map	2D plot
	point_graph.py	points	3D plot
perception	cone_id.cpp	lidar/Lidar1	points cone_location

Six ROS2 launch files have been developed to run the required ROS2 nodes at different stages of the development. Table 17 shows a summary of the six launch files, and the associated nodes launched.

Table 17: ROS2 Launch File Summary

Launch File	Package	Node
autonomous_example.launch.py	autonomous_example	car_controller.cpp
		cone_id.cpp
		steering_angle.cpp
		throttle_pos.cpp
graph_autonomous_example.launch.py	autonomous_example	car_controller.cpp
		cone_id.cpp
		steering_angle.cpp
		throttle_pos.cpp
	graphing	cone_graph.py
basic_lap.launch.py	autonomous_example	car_controller.cpp
		cone_id.cpp
		throttle_pos.cpp
	basic_lap	steering_angle.cpp
graph_basic_lap.launch.py	autonomous_example	car_controller.cpp
		cone_id.cpp
		throttle_pos.cpp
	basic_lap	steering_angle.cpp
	graphing	cone_graph.py
perception.launch.py	autonomous_example	car_controller.cpp
		throttle_pos.cpp
	basic_lap	steering_angle.cpp
	perception	cone_id.cpp
graph_perception.launch.py	autonomous_example	car_controller.cpp
		throttle_pos.cpp
	basic_lap	steering_angle.cpp
	perception	cone_id.cpp
	graphing	cone_graph.py





## 5 Contributions – Git Repository

All contributions to the UniSQ race team created within this project have been updated in the git repository that was created as part of the simulator setup in section 4.1.4. Throughout the autonomous software development, the git repository has been maintained and updated with a README file summarising the repository as per below.

The link for the git repository is available in section 6.2.

### **autonomous\_formula\_student**

Created by Alistair Thorogood 2023 for the requirements of ENG4111 & ENG4112 Research Project. This Repo is a ros2 workspace designed to work with the formula student driverless simulator.

This repo currently contains four packages, 9 nodes and 6 launch files for autonomous vehicle control:

- **autonomous\_example** (a c++/ros2 copy of the python example provided with the simulator)
  - autonomous\_example.launch.py
  - graph\_autonomous\_example.launch.py
  - car\_controller.cpp
  - cone\_id.cpp
  - steering\_angle.cpp
  - throttle\_pos.cpp
- **basic\_lap** (an improved version of autonomous\_example with path planning & proportional steering)
  - basic\_lap.launch.py
  - graph\_basic\_lap.launch.py
  - steering\_angle.cpp
- **graphing** (a package for graphing traffic cone locations)
  - cone\_graph.py
  - map\_graph.py
  - point\_graph.py
- **perception** (improved LiDAR perception)
  - perception.launch.py
  - graph\_perception.launch.py
  - cone\_id.cpp

**python\_race\_path** was utilised for mapping the race track outside of ROS and may be utilised for future path planning development. It is not a part of the ROS workspace



## 6 Conclusion

This project set out to achieve three project aims; Set up and Test Simulation Environment, Demonstrate Basic Self-Driving, and Establish UniSQ FSAE Autonomous Development Platform. Each of these aims were completed satisfactorily and additionally a robust LiDAR perception algorithm was developed.

After evaluation of a range of existing Formula Student simulators the Formula Student Driverless simulator was chosen, installed, trialled, and utilised for the development of basic self-driving. This was determined as an appropriate simulator for autonomous development. A cheat sheet was developed to assist future students with installation and operation of the simulator.

Basic self-driving was demonstrated utilising the efficient C++ programming language and ROS2 robotics framework. Several ROS2 nodes were created to perceive traffic cones, determine steering angle, determine throttle position, and control the vehicle. The initial lap time of the basic self-driving software was 160.37s this was improved to 82.31s throughout the development.

A UniSQ FSAE Autonomous Development Platform was created in the form of a git repository. The git repository contains the ROS2 workspace with all nodes for basic self-driving, improved perception and visualisation tools used throughout the development. A README file was developed to assist students with utilisation of the repository for further autonomous development.

Above the aims of this project, an effective and robust LiDAR based perception algorithm was developed based off the Velodyne VLP-16 LiDAR currently available to the UniSQ team. This allowed for integration into a real-world vehicle and improved the simulated perception range from 7m to greater than 20m allowing for future autonomous development in motion estimation and mapping.

### 6.1 Future Work

Substantial work remains to complete a competitive autonomous software within a simulation environment. The scope of developing a real-world autonomous vehicle is too large to detail in this conclusion, it will require a multi-disciplinary team of student engineers to complete.



To complete a competitive autonomous software motion estimation and mapping will need to be developed. This will involve but is not limited to; implementation of SLAM, determination of a race path and determination of a velocity profile for the race path.

Advanced vehicle control will also be required to be developed. Control methods such as Pure Pursuit, Stanley Controller and Model Predictive Control will need to be implemented once motion estimation and mapping has been developed.

## **6.2 Links**

Code Repository: [https://github.com/ThoroMech/autonomous\\_formula\\_student.git](https://github.com/ThoroMech/autonomous_formula_student.git)

Final Lap Video: <https://youtu.be/JEwTjVRfeeU>

## 7 References

AMZ-Driverless n.d., *FSSIM - Formula Student Simulator*, github, <https://github.com/AMZ-Driverless/fssim>>.

Behrendt, T 2017, 'Implementation of a Driving Simulator within a Formula Student Team', Monash University, Monash Motorsport.

Betz, J, Zheng, H, Liniger, A, Rosolia, U, Karle, P, Behl, M, Krovi, V & Mangharam, R 2022, 'Autonomous vehicles on the edge: A survey on autonomous vehicle racing', *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458-88.

Broatch, K 2019, 'LAP TIME SIMULATION OF A FORMULA STUDENT RACING CAR'.

cplusplus n.d., *C Plus Plus Tutorials*, viewed 08 October 2022, <<https://cplusplus.com/doc/>>.

Ding, Y 2020, 'Three Methods of Vehicle Lateral Control: Pure Pursuit, Stanley and MPC'.

Donovan, R 2023, *Beyond Git: The other version control systems developers use*, Stack Overflow, viewed 22 July 2023, <[https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/#:~:text=This%20year%2C%20we%20asked%20what,\(Apache%20Subversion\)%20and%20Mercurial.>](https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/#:~:text=This%20year%2C%20we%20asked%20what,(Apache%20Subversion)%20and%20Mercurial.>)>.

Doyle, DA, Cunningham, G, White, G & Early, J 2019, *Lap time simulation tool for the development of an electric formula student car*, 0148-7191, SAE Technical Paper.

Edinburgh-University n.d., *eufs\_sim - ROS/Gazebo simulation packages for driverless FSAE vehicles*, GitLab, viewed 09 August 2023, <[https://gitlab.com/eufs/eufs\\_sim](https://gitlab.com/eufs/eufs_sim)>.

Formula-Student-Driverless-Community 2022, *Required Process [fsds/ros\_bridge-2] has died!* #353, viewed 24 May 2023, <<https://github.com/FS-Driverless/Formula-Student-Driverless-Simulator/issues/353>>.

Formula-Student-Driverless-Community n.d., *Formula Student Driverless Simulator*, <https://fs-driverless.github.io/Formula-Student-Driverless-Simulator/v2.1.0/>>.

FSG 2022a, *FSG Competition Handbook 2023*, Formula Student Germany.

FSG 2022b, *Results FSG 2022*, Formula Student Germany, viewed 24 September 2022, <<https://www.formulastudent.de/fsg/results/2022/>>.

FSG 2023, *Formula Student Rules 2023*, Formula Student Germany.

FSG n.d., *Formula Student Germany International Design Competition*, viewed 01 May 2023, <<https://www.formulastudent.de/teams/fsd/>>.

Gitea n.d., *USQ Student Git*, <https://139.86.55.220/>>.

Hoffman, C 2014, *Dual Booting Explained: How You Can Have Multiple Operating Systems on Your Computer*, How To Geek, viewed 22 July 2023,

<<https://www.howtogeek.com/187789/dual-booting-explained-how-you-can-have-multiple-operating-systems-on-your-computer/>>.

Kabzan, J, Valls, MI, Reijgwart, VJ, Hendriks, HF, Ehmke, C, Prajapat, M, Bühler, A, Gosala, N, Gupta, M & Sivanesan, R 2020, 'Amz driverless: The full autonomous racing system', *Journal of Field Robotics*, vol. 37, no. 7, pp. 1267-94.

Kritayakirana, K & Gerdes, JC 2012, 'Autonomous vehicle control at the limits of handling', *International Journal of Vehicle Autonomous Systems*, vol. 10, no. 4, pp. 271-96.

Kuruville, T 2022, 'Path Planning for Formula Student Driverless Cars Using Delaunay Triangulation', *Student Lounge*.

Large, NL 2020, 'A comparison of different approaches to solve the SLAM problem on a Formula Student Driverless race car', Karlsruhe Institute of Technology.

MathWorks n.d., *MATLAB Help Center*, viewed 08 October 2022, <<https://au.mathworks.com/help/matlab/getting-started-with-matlab.html>>.

matplotlib n.d., *Matplotlib: Visualization with Python*, viewed 24 May 2023, <<https://matplotlib.org/>>.

Monash-Motorsport 2022, *Monash Motorsport defends its title at FSAE-A 2022*, viewed 01 May 2023, <<https://www.monash.edu/engineering/about/news/articles/2022/monash-motorsport-defends-its-title-at-fsae-a-2022>>.

Nvidia n.d., *NVIDIA PhysX SDK 3.4.0 Documentation - User Guide*, <https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Vehicles.html>>.

Okunsky, M & Nesterova, N 2019, 'Velodyne LIDAR method for sensor data decoding', *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, p. 012018.

Ovenden, S 2019, 'Perception Integration for an Autonomous Vehicle', Monash University, Monash Motorsport.

ROS n.d., *ROS Getting Started*, viewed 08 October 2022, <<https://www.ros.org/blog/getting-started/>>.

SAE 2022a, *Formula SAE Rules 2023*, SAE Australasia.

SAE 2022b, *FSAE-A Autonomous Vehicle Addendum*, SAE Australasia.

Slomoi, A 2018, 'Path Planning and Control in an Autonomous Formula Student Vehicle', Monash University, Monash Motorsport.

Storc, D 2022, 'Detection of Traffic Cones from Lidar Point Clouds', Bachelor Thesis thesis, Czech Technical University in Prague.

TUMFTM, AHLHFC 2021, *global\_racetrajectory\_optimization*, GitHub.



Ubuntu n.d., *Install Ubuntu desktop*, viewed 24 May 2023,  
<<https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview>>.

Velodyne 2019, *VLP-16 User Manual*, Velodyne LiDAR.

Velodyne n.d., *What is lidar?*, Velodyne Lidar, viewed 02 May 2023,  
<<https://velodynelidar.com/what-is-lidar/>>.

W3Schools n.d., viewed 08 October 2022, <<https://www.w3schools.com/cpp/>>.



## 8 Appendices

### Appendix A1 - Project Specification

For: Bachelor of Engineering (Honours)

Title: FSAE Autonomous Control – Motion Estimation and Mapping

Major: Mechatronics

Supervisors: Craig Lobsey

Enrolment: ENG4111 – EXT S1, 2021

ENG4112 – EXT S2, 2021

Project Aim: Develop an autonomous mapping and path planning algorithm for the University of Southern Queensland's formula SAE (Society of Automotive Engineers) team, utilising a simulation environment.

**Programme:** Version 1, February 2023

1. Preparation:
  - a. Literature Review
  - b. Upgrade PC to meet simulator requirements
  - c. Create Linux Partition on PC
  - d. Download Simulation Software
2. Familiarisation
  - a. Familiarisation with Simulation Software
  - b. Familiarisation with ROS
  - c. Familiarisation with C++
  - d. Familiarisation with GitHub
3. Develop / Source Code to perform Basic Functions
  - a. Convert autonomous python examples to C++ with ROS
4. Develop Mapping and Path Planning
  - a. Develop discovery lap mapping algorithm and code
  - b. Develop path optimisation algorithm and code
  - c. Develop localisation algorithm and code
5. Combine and Optimise
  - a. Develop an overall code combining phase 4 & 5
  - b. Test and optimise combined code
6. Develop Software Platform
  - a. Create USQ Race Team USQ Student Git account
  - b. Set up USQ Student Git Account
7. Present and Handover
  - a. Prepare work for handover to USQ FSAE team
  - b. Present work to USQ FSAE team
  - c. Final Handover to USQ FSAE Team

## Thesis Project Plan

The Microsoft project Gantt chart developed in ENG4110 is being used to track progress and update tasks as required. Appendix A3 - Project Gantt Chart below shows the project and proposed future plan.

At this stage drumbeat supervisor communication is not required. Supervisor Communication will be required ad hoc as I progress through the phases, I will attempt to provide two weeks' notice for any assistance required.

*To avoid another landscape page the Gantt chart from the project specification has been removed and the Gantt chart in Appendix A3 - Project Gantt Chart referenced.*

## Resource Requirements

The only resource required for this project is a Personal Computer (PC) that meets the requirements for the simulator. The requirements are outlined as:

- 8 core 2.3Ghz CPU
- 12 GB memory
- 30GB free SSD storage (120GB when building the unreal project from source)
- Recent NVidia card with Vulkan support and 3 GB of memory.

The RAM, GPU and Hard drive have been updated to meet requirements the table below outlines completed upgrades.


	Brand	Model	Cost
RAM	Corsair	CMK32GX4M2A2666C16	\$ 190.89
GPU	Gigabyte	GTX1650	\$ 276.00
SSD Hard Drive	Samsung	1TB 980 Pro MZ-V8P1T0BW	\$ 221.19
Total			\$ 688.08

The project has started with the simulator running on the upgraded PC. The resources have been determined sufficient.



## Appendix A2 - Personal and Property RMP

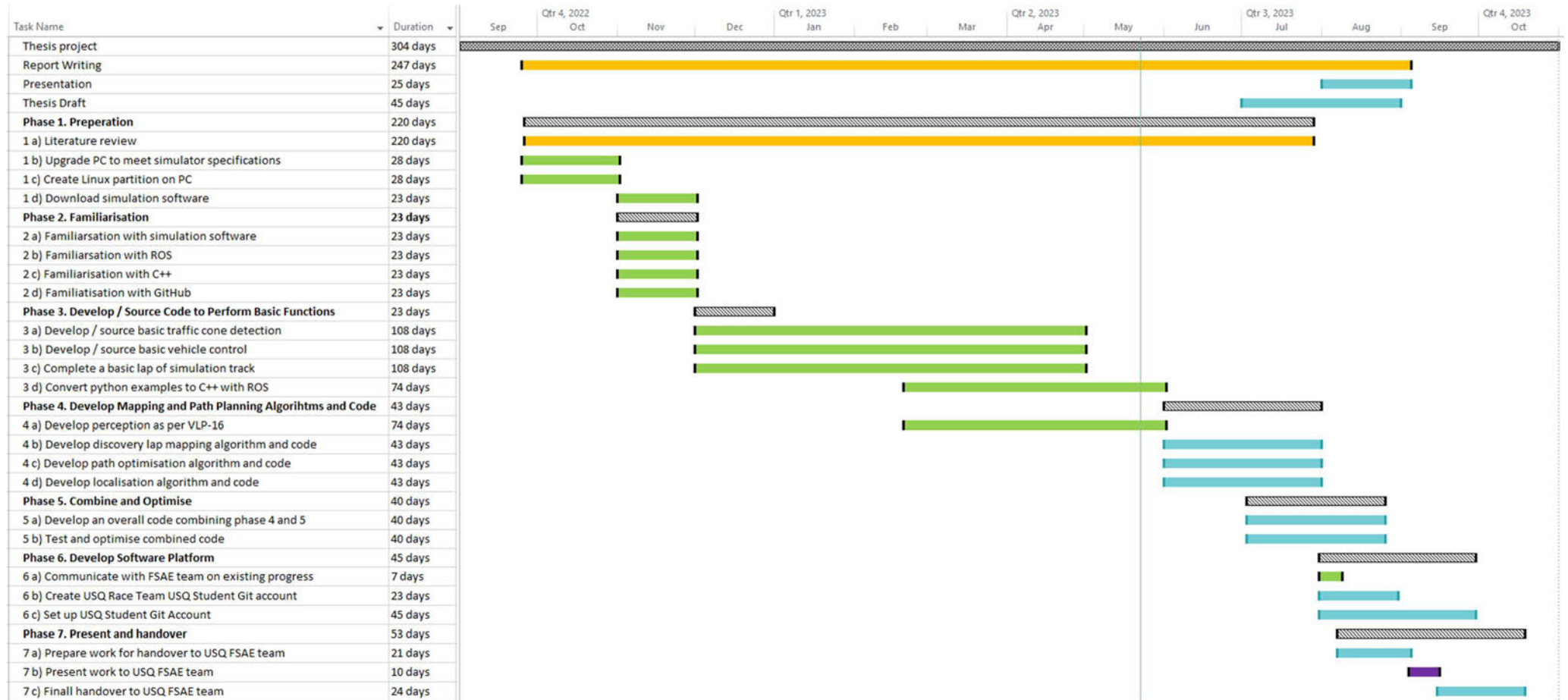
A risk management plan has been completed in UniSQ's Safety Central – SAFETRAK WHS Risk Register. A concise summary of the risk assessment is provided below. For further information review the risk assessment in Safety Central reference number 1620.

NUMBER	RISK DESCRIPTION	TREND	CURRENT	RESIDUAL
1620	ENG4111/2, A.Thorogood, FSEA Autonomous Control		Medium	Low
DOCUMENTS REFERENCED				
ENG4111/2 Research Project FSAE Autonomous Control - Motion Estimation and Mapping				
RISK OWNER	RISK IDENTIFIED ON	LAST REVIEWED ON	NEXT SCHEDULED REVIEW	
Alistair Thorogood	10/05/2023	12/05/2023	12/11/2023	
RISK FACTOR(S)	EXISTING CONTROL(S)	PROPOSED CONTROL(S)	OWNER	DUE DATE
Upgrade of PC components: Using unpowered hand tools (e.g. screwdriver) and exposure to low voltage electricity.	<b>Control:</b> PC components are modular and safe for user access and modification (ELV only - 12V)	<b>Control:</b> Unplug PC from mains power before undertaking any modifications. Do not open the PC power supply enclosure.		01/06/2023



Upgrade of PC Components: Incorrect installation/setup resulting in damage to PC (personal property)	<b>Control:</b> PC components are modular and designed for required component replacement.	<b>Control:</b> Follow installation instructions and documentation and seek support if needed or uncertain.	01/06/2023
Thesis project, software development and report writing Extended periods of time sitting Extend periods of time looking at screen	<b>Control:</b> Ergonomic office chair currently in use.	<b>Control:</b> Follow good ergonomic practice and work area setup for study use.	01/09/2023
Utilisation of simulation in practice This is not happening in this project, as no car is available.	<b>Control:</b> Further risk assessment will be required before utilisation of the autonomous control software in a real-world environment. At this stage there are too many unknowns to develop reasonable controls. For this project this risk is non-existent.	<b>No Control:</b>	

## Appendix A3 - Project Gantt Chart



## Appendix A4 - Running the Simulator – Cheat Sheet

### 1. Install required applications and extensions:

- Install terminator (optional)
  - `sudo apt-get install terminator`
  - Assists by being able to run multiple Linux terminals
- Install git
  - `sudo apt-get install git`
- Install ROS2 humble
  - Follow ROS2 humble documentation for install  
<https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html>
- Install visual studio (optional)
  - `sudo snap install code --classic`
  - Assists in being able to write ROS2 nodes and other code.
  - Open visual studio code using: `code .`
  - Search for and install ROS2 extension for visual studio code.

### 2. Install Required Extensions

- `sudo apt-get install ros-humble-turtle-tf2-py ros-humble-tf2-tools ros-humble-tf-transformations`
- `sudo apt install python3-pip`
- `sudo apt install python3-colcon-common-extensions`

### 3. Add source lines to .bashrc

- `source /opt/ros/humble/setup.bash`
- `source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash`

### 4. Clone simulator repository to home

- `cd && git clone https://github.com/FS-Driverless/Formula-Student-Driverless-Simulator.git --recurse-submodules`

### 5. Setup AirSim

- `~/Formula-Student-Driverless-Simulator/AirSim/setup.sh`
- `~/Formula-Student-Driverless-Simulator/AirSim/build.sh`

### 6. Download the simulator binary

- Download ZIP (fsds-v2.2.0-linux.zip) from <https://github.com/FS-Driverless/Formula-Student-Driverless-Simulator/releases>
- Extrac the ZIP to downloads folder

### 7. Build ROS2 environment:

- `cd ~/Formula-Student-Driverless-Simulator/ros2 && colcon build`
- Add source to bashrc
  - `source ~/Formula-Student-Driverless-Simulator/ros2/install/setup.bash`

### 8. Run simulator

- `~/Downloads/fsds-v2.2.0-linux/FSDS.sh`
- F11 to remove full screen if required

### 9. Launch ROS2

- `cd ~/Formula-Student-Driverless-Simulator/ros2`
- `ros2 launch fsds_ros2_bridge fsds_ros2_bridge.launch.py`

### 10. Launch Autonomous Software