FORMULA STUDENT:
Development of Autonomous Drivers

ELEC5030M TEAM PROJECT
Author(s) Name: James Heavey
SID: 201198933
Supervisor(s) Name: K. Kubiak
Examiner Name: N. Gilkeson
Date of Submission: 28/4/2021
Formula Student:
Virtual Testing Environment
for Autonomous Vehicles

SCHOOL OF MECHANICAL ENGINEERING UNIVERSITY OF LEEDS

MECH5030M TEAM PROJECT 60 credits

TITLE OF PROJECT

FORMULA STUDENT:
Development of Autonomous Drivers

PRESENTED BY

James J. Heavey

OBJECTIVES OF PROJECT

The aim of this individual project is to develop a variety of autonomous drivers with varying competency and sensor configurations in order to validate the functionality of the developed simulator and vehicle evaluator

IF THE PROJECT IS INDUSTRIALLY LINKED TICK THIS BOX AND PROVIDE DETAILS BELOW ☐

COMPANY NAME AND ADDRESS:

INDUSTRIAL MENTOR:

THIS PROJECT REPORT PRESENTS OUR OWN WORK AND DOES NOT CONTAIN ANY UNACKNOWLEDGED WORK FROM ANY OTHER SOURCES.

SIGNED: JAMES HEAVEY DATE: 28/4/2021

J.J.H

Table of Contents

Abstract

This paper describes the design and implementation of a modular autonomous driving algorithm, created to accompany the autonomous vehicle virtual simulated environment. In total, 72 algorithm combinations can be implemented providing an abundance of simulation and vehicle evaluation test data. The resulting modular driver provides a good starting template for development by the future University of Leeds FS-AI team with several immediate improvements suggested as future work.

Acknowledgments

I would like to especially thank my supervisor, Dr. Krzysztof Kubiak, for providing guidance and feedback throughout this project, without whom, this project would not have been possible. I would also like to thank my fellow team members Titusz Ban, Matthew Currie and Daniel Marshall whose work in the development of the simulator as part of the greater team project was instrumental in the success and development of this individual project.

List of Figures

Chapter 1: Introduction

1.1 Introduction

The field of autonomous vehicle technology is developing rapidly, with growing interest from both academia and industry. Competent AI for autonomous vehicles offers several advantages over the human-controlled norm, including: reduced accident frequency, increased transport efficiency and reduction in GHG emissions [1], [2], [3]. Despite the advantages, autonomous vehicles face many barriers to commercial market entry. Commercial autonomous systems are required to monitor a large range of sensor stimuli, create an accurate internal representation of the environment and generate appropriate vehicle control signals that will ensure the safety of both the passenger and pedestrians [4]. Developing a competent system to interact with exceptionally unpredictable and non-uniform environments (as would be seen in cities) is complex, requiring a vast amount of funding and research.

Autonomous racing series such as DARPA, Roborace and Formula Student AI have emerged to spur innovation and interest in these challenges among engineers [5], [6], [7]. This project focuses on the Formula Student AI competition, an education engineering challenge issued by the Institute of Mechanical Engineers encouraging young engineers to develop an autonomous system capable of navigating a track with certain pre-set properties (reducing the challenge from that expected in commercial settings). The primary aim of this competition is to allow young engineers to develop the skills required to enter the industry and bolster innovation.

The team project which encompasses this individual project aims to facilitate the creation of a new University of Leeds Formula Student AI team by designing a Virtual Testing environment. This will allow future teams to design, test and validate autonomous systems on tracks and in settings that reliably simulate the properties of a real-world competition track. Testing and validation of autonomous systems is a major component of the design cycle. It is critical that the systems not only work as intended, but are also safe and robust before implementation in real-world hardware [8], [9]. A virtual testing environment greatly reduces the risks and costs of software testing, while also increasing efficiency and allowing increased feedback and evaluation [10].

1.2 Aims

1.2.1 Team Project Aims

The aim of the team project is “to design and create a robust SIL testing environment for the Formula Student Autonomous Vehicle competition (FS-AI) that can be used by the future University of Leeds FS-AI team to develop, test and optimise driving algorithms. The environment must simulate all the properties and rules set in the competition. “ [11].

1.2.2 Individual Project Aims

The aim of this individual project is to develop a variety of autonomous drivers with varying competency and sensor configurations in order to validate the functionality of the developed simulator and vehicle evaluator. To this end, the driver development will focus on modularity and the ability to utilise a range of sensor inputs and control algorithms to achieve a variety of output combinations. This strays from the typical goal of autonomous AI development, where the aim is to achieve a single sensor configuration and driving algorithm that is highly optimised for racing purposes. The modular algorithm developed over the course of this project aims to serve as a template for development by the future University of Leeds FS-AI team.

1.3 Objectives

In order to complete these objectives, the developed system is required to be able to:

- Utilise a range of sensors and sensor configurations provided by the simulator.
- Determine both the vehicle position and the position of track landmarks.
- Build an accurate internal world map.
- Determine a suitable path trajectory.
- Calculate actuator outputs to trace the calculated path trajectory.
- Allow multiple algorithm combinations (modularity through standardised data transmission)

1.4 Report Layout

Following the introduction and a thorough analysis of relevant literature, this report details the architecture and concept of the designed driver system, followed by an in-depth evaluation of individual module processes. The results are analysed and justified throughout. The report concludes with recommendation for future work to be conducted prior to FS-AI competition entry.

Chapter 2: Literature Review

2.1 Introduction

The development of autonomous driving algorithms is a field thoroughly explored in literature. Given the complexity of AI and numerous mathematical processes required to design a robust system, it is imperative to consult well documented methods and results to inform decisions about the system design and operation.

2.2 Overview of Formula Student Autonomous Racing

Formula Student is a longstanding annual competition, hosted by the Institute of Mechanical Engineers, challenging university student teams to design and race their own vehicle systems. In 2019, the first FS-AI competition was held, a sub-branch of the original competition, introducing a driverless racing series that requires students to explore a variety of AI fields [12]. Two competition classes are presented: Automated Driving System (ADS) and Dynamic Driving Task (DDT). The ADS class requires teams to design a full autonomous vehicle platform including: the hardware and circuitry design, assembly and software implementation. The DDT class offers a standard vehicle platform, with customised sensor configurations as selected by the team, limiting the challenge to software development and optimisation [12]. The end goal for both classes is the completion of a given track in the fastest possible time.

The tracks used for the competition maintain standard properties that allow for simplified perception modules to designed. Each track, unpredictable in layout, uses specific-coloured cones to mark track borders. Blue cones mark the left track border and yellow mark the right, relative to the intended forward direction [13]. The algorithm designed must be able to account for a range of sensor configurations, as defined across both the ADS and DDT classes. The standardised coloured cone track borders are imperative for the design of the perception module, as key landmarks can be identified, allowing data extraction and enrichment to be performed.

2.3 Camera Object Detection

Computer Vision is an interdisciplinary field with literature dating back to the 1970's [14]. Computer Vision is an umbrella term to address the problem of data extraction from a camera image with the goal of creating an AI capable of understanding key image elements akin to the human eye and brain. The camera, like the eye, provides the foundation of how autonomous systems perceive and interact with an environment.

As a highly nuanced and intensely studied field, there are numerous pre-existing resources and algorithmic tools that can be utilised to extract image data. One of the most powerful publicly available tools is the opensource Computer Vision library (OpenCV). Developed by Intel in 1999, it is now available in a multitude of programming languages such as Python and C++. OpenCV provides a toolkit of morphological and data conversion operations that form the foundation of modern engineer’s interaction with camera sensing systems [15]. Operations such as colour masking will be fundamental to the cone detection task of the driver AI in this project.

Recent developments in the field of Computer Vision have seen the integration of convolutional neural nets (CNNs), significantly increasing camera object detection accuracy through machine learning [16]. Where the camera emulates the visual function of the human eye, a CNN is designed to emulate the learning properties of the human brain. In principle, a CNN is a set of connected perceptron nodes, each with an individual bias and weight property. These properties of each perceptron in the network are modified during a training process whereby known inputs, coupled with their target classifications, are provided to the network. An incorrect input classification results in a full network update, reducing the overall classification error. At high level, numerous layered networks are compiled, creating a highly complex, layered neural network, capable of identifying object images by their shape and colour regardless of the image context [17].

2.4 LiDAR Object Detection

Light detection and ranging (LiDAR) sensors are the second most common exteroceptive sensors utilised by both consumer and racing autonomous vehicles following the camera. A LiDAR sensor operates by scanning the environment with lasers and measuring the time elapsed before the reflected ray returns to the receiver [18]. A typical LiDAR module used in autonomous vehicles is omni-directional and operates with a wavelength of 905 nm allowing clear data visibility across all weather and environmental conditions [19]. As an omnidirectional sensor, the raw data generated is 3D in the form of a coordinate point cloud.

Data extraction techniques using LiDAR sensors often involve point clustering to reduce the huge number of data points generated by LiDAR and extract crucial object positions within an environment [20]. The primary benefit of a LiDAR sensor is its high resolution at long range (up to approximately 100m) and its accuracy in depth measurement, where it significantly outperforms camera-made estimations. However, as a depth-based sensor, depthless environmental information elements are lost, such as colour. This can make it difficult to distinguish objects of similar size and shape. The key role of a LiDAR sensor within autonomous vehicle AI is to supplement and cross validate the data provided by the camera using sensor fusion techniques [21].

2.5 Mapping Algorithms

Internal map building is an integral part of any driving algorithm. However, the mapping method is highly dependent on system design, with little more than high-level guidance discussed in literature. Culley *et al*. describe a probability-based mapping system that uses Kalman-filters to model both vehicle and environmental landmarks [22]. The proposed Kalman filter model offers numerous benefits over a standard weighted map. By modelling the uncertainty of a variable over time, a series of sensor updates can be provided allowing constant update and localisation. A typical Kalman filter operates using mean (μ) and variance (σ^2) to represent the centre of the variable estimation and the certainty respectively. They are widely used in many fields for their versatility, accuracy in estimating error-prone sensor readings and eliminating sensor noise [23].

Giacalone *et al.* discuss the challenges of aggregating and fusing data streams from multiple sensors, forming an accurate and reliable world map [21]. The importance of data standardisation and refinement is noted. While LiDAR and Camera imaging present raw environmental data in vastly different formats, the ideal data extraction from each is the same. Special consideration of normalised data formatting is crucial for planning and design of a complex mapping system with multiple data input streams.

A major consideration of all real-world driving algorithms is how to address the error inherent to all sensors, but most crucially, those tasked with odometry and localisation of the vehicle position as this forms the foundation of all navigation and internal world building. A typical autonomous vehicle will employ several proprioceptive sensors such as wheel encoders, compasses, IMUs [22]. Wheel encoders provide an estimate of the wheel speed, vehicle position and heading error, the IMU measures vehicle acceleration and rotation and compasses measure the vehicle heading. Addition of exteroceptive sensors such as GPS or even camera and LiDAR introduces the ability to cross validate internal sensor measurements with external Landmark tracking. Such algorithms are named simultaneous localisation and mapping (SLAM) algorithms and are commonplace throughout high level autonomous racing systems [22].

2.6 Kinematic Vehicle Modelling

When designing a vehicle control system, it is important to consider the kinematics of the vehicle. The simulated vehicle can be modelled as a 2 degree of freedom body within the 2D cartesian plane of the world frame with position (x, y), yaw orientation (θ) [24]. The two front wheels and the two back wheels are collapsed into singular wheels, where the forward wheel is responsible for steering and the rear is responsible for acceleration. The resulting model is known as the “bicycle model”, used throughout literature for its simple yet highly useful representation of vehicle properties [25]. To use this model in vehicle kinematics analysis, a reference point must be selected on the vehicle frame, the location of which alters the resultant equations of motion and ultimately the behaviour of the designed controller. As the intention of this project is to design numerous vehicle controllers, 2 reference point models were considered.

2.6.1 Rear-axle Bicycle model

The rear-axle bicycle model is defined by selecting a reference point at the centre of the vehicles rear axle. Figure 1 depicts the rear axle bicycle model, illustrating the vehicles position (x,y), velocity (v), body length (L), heading angle (θ), steering angle (δ), the turn radius (R) and centre of rotation (IC). Using this model, kinematic equations can be defined, describing the change in state of the vehicle with respect to time and inputs acceleration (a) and steering angle (δ) [26].

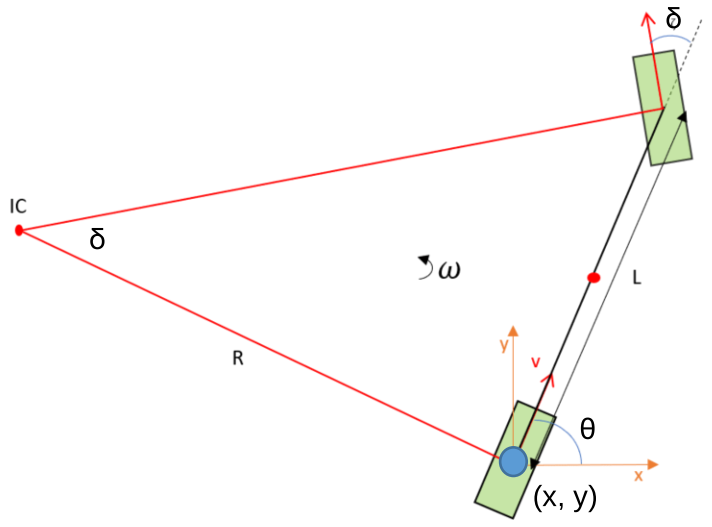

Figure 1: Rear-axle bicycle model [27]

The rate of change of position (ẋ, ẏ) is determined by the horizontal and vertical components of the vehicles instantaneous velocity (v).

$$\dot{x} = v\cos(\theta) \tag{1}$$

$$\dot{y} = v\sin(\theta) \tag{2}$$

By observing the instantaneous rotational centre of motion (3) and the right triangle formed between "R" and "L" (4), the angular acceleration ($\dot{\theta}$) can be calculated by:

$$\dot{\theta} = \frac{v}{R} \tag{3}$$

$$\tan(\delta) = \frac{L}{R} \tag{4}$$

$$\therefore \dot{\theta} = \frac{v}{L}\tan(\delta) \tag{5}$$

Combining the three equations produces the final differential equation of motion representing the change of state of the vehicle with time. The equations of motion are constrained by the physical restrictions of the vehicle where the steering angle is limited by the maximum wheel turn and the velocity and acceleration are limited by the engine, environmental friction and various other factors.

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}v\,cos(\theta)\\ v\,sin(\theta)\\ \frac{v}{L}\,tan(\delta)\end{bmatrix}, \qquad \delta \in [\delta_{min}, \quad \delta_{max}], \qquad a \in [a_{min}, \quad a_{max}] \tag{6}$$

It can be seen that the future pose of the vehicle (x, y, θ), is dependent only on the steering angle (δ) and the acceleration vector (a). These two factors makeup the vehicle control signal and are used to determine the motor and steering wheel actuator outputs, and hence control the vehicle.

As there are two variables defining the differential equation of motion, one controller must be defined for each parameter. The controller defining the acceleration control signal is known as a longitudinal controller (in the direction of motion) and the controller defining the steering angle control signal is known as a lateral control (normal to the direction of motion). The design of such controllers is explored in section 2.7.

2.6.2 Front-axle Bicycle model

The front-axle bicycle model shares all the parameters previously defined for the rear-axle model; however, the resultant differential equations of motion differ. The rate of change of position (ẋ, ẏ) is now also dependent on the steering angle (δ).

$$\dot{x} = v\,cos(\theta + \delta) \tag{7}$$

$$\dot{y} = v\sin(\theta + \delta) \tag{8}$$

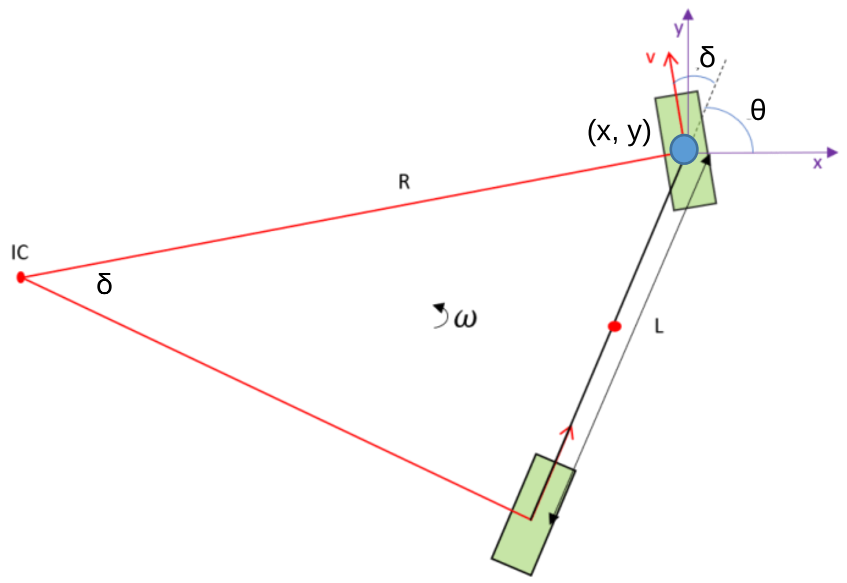

Figure 2: Front-axle bicycle model [27]

The same equations can be utilised to calculate the angular acceleration; however, the radius of rotation is now the hypotenuse.

$$\dot{\theta} = \frac{v}{R} \tag{9}$$

$$\sin(\delta) = \frac{L}{R} \tag{10}$$

$$\therefore \dot{\theta} = \frac{v}{L}\sin(\delta) \tag{11}$$

The final kinematic equations of motion for the front-axle bicycle model, obeying the same physical constraints outlined previously is:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\,cos(\theta+\delta) \\ v\,sin(\theta+\delta) \\ \frac{v}{L}sin(\delta) \end{bmatrix}, \delta \in [\delta_{min}, \quad \delta_{max}], a \in [a_{min}, \quad a_{max}] \tag{12}$$

2.7 Control Algorithms

The final stage of any driving AI algorithm is the determination of actuator outputs. Two types of vehicle control systems are discussed in literature: geometric and model-predictive controllers [25]. Geometric controllers are generic controllers that only utilise the vehicle kinematics and internal map to generate control signals. This controller type will occupy the focus of this study. Superior Model Predictive controllers (MPCs) may be developed, optimising the performance by considering specific vehicle properties (e.g. tire forces) and constructing a predictive cost function to estimate future control response. These controllers are vehicle specific and typically require extensive testing and parameter optimisation. The project purpose requires controller compatibility with a variety of vehicle meshes. Therefore, MPC design is not within the scope of this project, but should be noted for future development prior to competition entry.

A separate controller is required to define the appropriate response of each input in the vehicle kinematic model (as defined in section 2.6). The longitudinal controller defines the acceleration response to measured error in the actual versus target velocity. The lateral controller defines the steering angle response to a measured lateral error. The lateral error measurement is controller-based and can be defined as: the signed angle from the vehicle to the reference path; the signed angle difference between the vehicle and path forwards; or the cross-track error (signed lateral distance between the vehicle position and the nearest track coordinate) [27], [24].

Samak *et al.* performed an extensive review of various longitudinal and lateral controllers [25]. Candidates for the longitudinal controller consisted of combinations of PID terms including: P, PI, PD and PID. Given the that the desired system response to a velocity error is a quick rise time with minimal overshoot, a properly tuned PID outperformed its counterparts although it was noted that PI is sometimes preferable, especially where an overdamped system response is desired.
While a PID controller could also be defined for the lateral controller [22], its performance at high speeds and at sharp corners is limited [25]. Geometric controllers offer a more accurate steering control alternative by considering the geometry of the reference path and kinematics of the vehicle state. The Pure-pursuit controller operates using a lookahead reference point selected from the path trajectory. By examining the vehicle pose relative to the lookahead point, an appropriate steering angle is generated for the calculated circular turn radius [26]. This controller is favoured for its high degree of accuracy and ability to meet all track way points provided it is coupled with a competent longitudinal controller. An alternative lateral controller was developed by Stanford University’s 2005 DARPA Grand Challenge team, aptly named the Stanley controller [28], [29]. This controller accounts for a heading and cross-track error term independently, allowing the controller to generate a steer command that ensures the vehicle returns to the path in the same time, independent of the current velocity [28]. This property was particularly useful in the rough terrain of the DARPA challenge, contributing to their overall win of the event. In recent years, this controller has become a popular alternative to pure-pursuit in autonomous settings, despite the slight decrease in ride stability.

Chapter 3: Development of Driver Concept

3.1 System Architecture

The designed algorithms follow a typical linear autonomous vehicle AI driver structure, splitting core processes into four separate modules: Perception, Mapping, Path-planning and Vehicle Control [21]. These modules were designed as “black boxes”, allowing individual module processes to have several options, providing a wide variety of test data for the simulator.

The data flow throughout the modules is linear, as illustrated in Figure 3. The driver system was designed so that it only interfaces with the simulator through the simulated sensor and actuator Gazebo to ROS interface. This interface can be replaced by a hardware to ROS interface in the future, allowing the driver algorithms to be easily deployed in hardware vehicles.

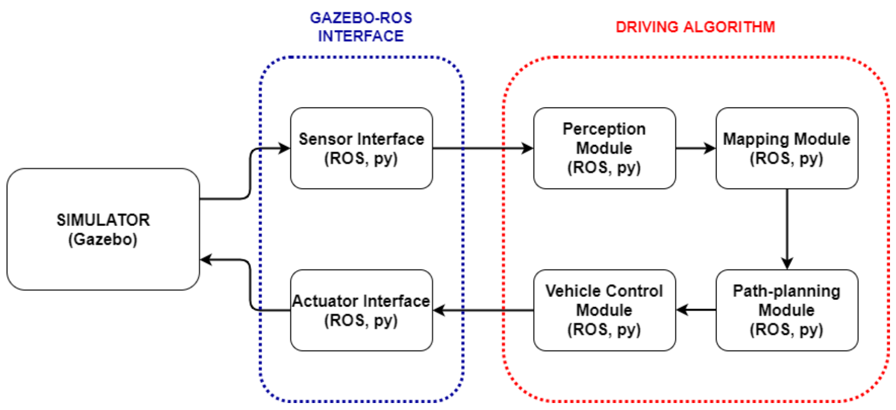
GAZEBO-ROS INTERFACE
DRIVING ALGORITHM
SIMULATOR (Gazebo)
Sensor Interface (ROS, py)
Actuator Interface (ROS, py)
Perception Module (ROS, py)
Mapping Module (ROS, py)
Path-planning Module (ROS, py)
Vehicle Control Module (ROS, py)

Figure 3: Overview of driver algorithm architecture in relation to simulator.

The perception module is responsible for data extraction and enrichment of the raw sensor data streams, providing key information properties to the mapping and planning modules. This module is currently able to extract and standardise cone data from the three available camera sensors and LiDAR, using the simulated odometry generated by Gazebo.

The mapping module is responsible for collating the information generated by the perception module and fusing the various sensor data streams into a single map data structure that accurately describes the key features of the vehicle’s environment. The minimum sensor requirement for this module is simulated odometry and 1 camera sensor. It is also able to utilise LiDAR and additional camera data streams if available.

The planning module is responsible for generating a trajectory for the vehicle to traverse based on the constructed map. Two planning options were implemented, both generating path trajectories that aim to trace the centre-line of the track.

Finally, the control module is responsible for the generation of the two vehicle actuator commands: steering wheel angle and acceleration. Three steer controllers were implemented coupled with a single acceleration controller.

In total 72 different driver module combinations can be created using the various Perception, Planning and Control modules developed in this project, providing sufficient data for the simulator to compare driver efficiency and competencies.

3.2 Project Technologies

Gazebo was chosen to host the simulation, with communication between vehicle systems handled using Robotic Operating System (ROS). ROS is a communication framework used widely in robotics applications for its intuitive structural style. Each of the modules described in this report is defined as a ROS node or processing unit. Topics are used to define information channels between nodes allowing easy transfer of, sensor, actuator and intermediary data between the internal nodes of the driver and the external simulator.

Python was selected as the processing language for its simplicity and compatibility with useful tool libraries such as OpenCV (used to process camera sensor data), numerical Python (NumPy), scientific Python (SciPy), NetworkX and neural network libraries such as TensorFlow and PyTorch. NumPy and SciPy significantly improved productivity and allow mathematical and data manipulation operations to be performed with ease. OpenCV was instrumental in the design of the camera related Perception module processes, and the availability of high-quality neural network libraries such as TensorFlow will allow significant improvements to made in the future.

Chapter 4: Perception Module

4.1 Module Overview

The goal of the perception module is to enrich the raw data streams of all the external sensors made available by the simulator. The sensors available for use in the simulated environment include three 2D ROS camera sensor plugins and a "Faux LiDAR" created by the team to simulate data outputs of a real LiDAR sensor. Odometry sensors such as wheel encoders, compasses, and accelerometers are not included in the simulator. Instead, Gazebo generates pose and location data of the vehicle mesh directly, removing the requirement to calculate this data from raw sensor outputs.

This module must condense the raw sensor data into key properties of the cone, namely, location, an existence probability and colour (if applicable). The cone must be represented in a standard form across all sensor types to allow simple sensor fusion and mapping operations to be implemented

4.2 Camera Sensor(s)

4.2.1 Simulated Sensors

The camera sensors included in the vehicle models of the virtual simulated environment are standard 2D RGB Gazebo camera plugins with ROS compatibility. Three cameras in total were included in each vehicle mesh, each with a position (x, y, z) and an orientation (roll, pitch, yaw) that defines their own frame of reference. The cameras record a standard square image with a resolution of 800x800 pixels and an FOV of 1.7 radians with a maximum update rate of 30 Hz. An example frame of the raw simulated camera video feed can be observed in Figure 4.

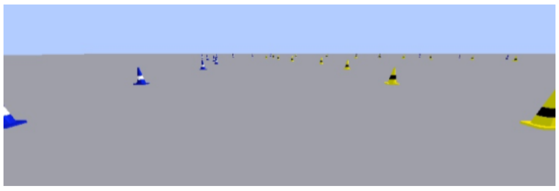
Figure 4: Image frame from simulated camera video feed (cropped)

4.2.2 OpenCV Cone Detection

The opensource Computer Vision library (OpenCV) provides a library of tools to perform image manipulation operations. This library can be used to identify the coloured cones in each camera image and return a single pixel coordinate indicating their location, a weight corresponding to the certainty and a colour to categorise the track walls.

The simulated cameras display image frames as RGB images. Before data extraction operations can be conducted, the image must be converted to a HSV colour base. RGB colour definition contains three colour components (red, green and blue), HSV separates the colour component by using a single colour value (hue) and two intensity components: saturation and value. The HSV definition allows colour ranges to be easily defined over a range of intensities, making the system robust against lighting changes and environment shadows. RGB to HSV conversion is implemented using the OpenCV function "RGB2HSV", where the hue is measured in degrees between 0-179° and saturation and value are 8-bit integers measured between 0-255.
A masking operation can be used to identify image pixels that fall within a given HSV colour range. The resulting mask is a binary image array with the same dimensions as the original image in pixels. A "1" in the mask array corresponds to the original image pixel colour value falling within the selected HSV colour range. The HSV colour ranges used to mask the primary cone colours blue and yellow are displayed in Figure 5.

BLUE RANGE	**[100,100,100]**	**[125,255,255]**
YELLOW RANGE	**[25,150,150]**	**[35,255,255]**

Figure 5: HSV Colour ranges used for cone detection

Figure 6 displays blue and yellow cone masks extracted from the raw image (Figure 4). The masking operation can be extended to include the secondary band colours of the cones (white and black), then a bitwise AND operation between the primary and secondary cone masks will allow a full mask of each cone to be defined. However, within the simulated environment, the background colour texture falls within the colour range of the shaded region of the blue cones' white band. This prevents secondary mask addition from being implemented as a solution to defining a full cone mask.

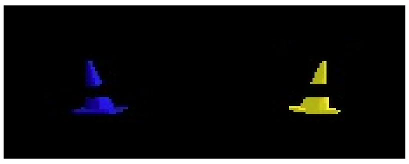

Figure 6: Cone masks extracted from raw camera image

Instead, a morphological approach known as a closing operation was implemented. In Computer Vision, a closing operation is defined as a dilation, followed by an erosion to close small gaps within a colour mask. As discussed previously, a mask is a binary image matrix where “1” represents a pixel of the original image within the given HSV range. A dialtion operation is a 2D convolutional operation requiring a structuring element (kernel) to define the dilation behaviour. A kernel is a second 2D binary matrix of any size. The centre of the kernel matrix is always a “1”, the remaining elements determine how the mask image grows in dilation or shrinks in erosion. To connect the lower and upper cone portions, it best if dilation only takes place in the vertical directions. Therefore a 3x3 kernel “K” was defined to perform the dilation convolution.

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{13}$$

The kernel K is convoluted throughout the original binary mask image, centred on each pixel with a value of “1”, and setting the pixel coordinates directly above and below it to “1” as well. A single iteration of this dilation operation will grow all mask elements vertically by 2 pixels. However the gap between the upper and lower portions can be up to 50-100 pixels depending on its proximity to the camera, therefore the dilation must be performed iteratively to ensure the masks are connected. The result of 50 dilation operations can be seen in Figure 7.

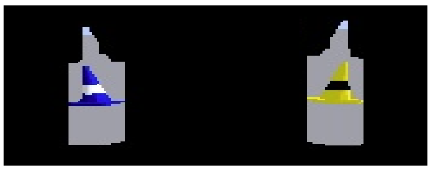

Figure 7: Result of dilation operation on image mask

Once the upper and lower cone masks are connected, an erosion operation, the inverse of a dilation operation, can be used to remove the unwanted pixels accumulated by the dilation. Using the same kernel “K”, the kernel is convoluted through the dilated mask, centreing on all pixels with a value of “1”. Any pixel that is a “1”, that does not have another pixel marked “1” directly above and below, is set to “0”. This erodes the top and bottom edges of the mask, leaving the now connected centre of the cone intact. The result of the full closing operation can be seen in Figure 8.

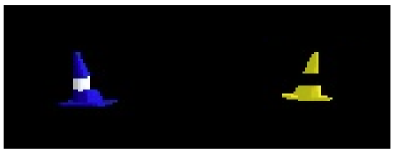

Figure 8: Full cone masks defined after closing operation

With the full cones detected in each image, the coordinates of the pixels at the lowest point of each mask are extracted, providing a single coordinate estimation of the cones position. The area of each mask is scaled between 1-0, providing an estimate for the cones existence probability. The full labelled frame published by the perception module can be seen in Figure 9 with blue cones highlighted green, yellow cones highlighted pink, and the cone estimation pixels marked with red circles.

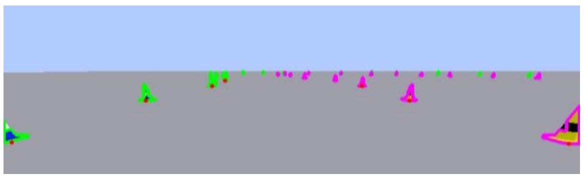

Figure 9: Labelled camera image with cone objects highlighted

4.2.3 Camera Projection

The cones perceived in each frame update of the camera have a pixel coordinate (u, v) and a pixel area used for weighting. Before this cone data can be used by the mapping module, the cones must be transformed from the image frame of reference to the camera, then vehicle and finally world reference frames.

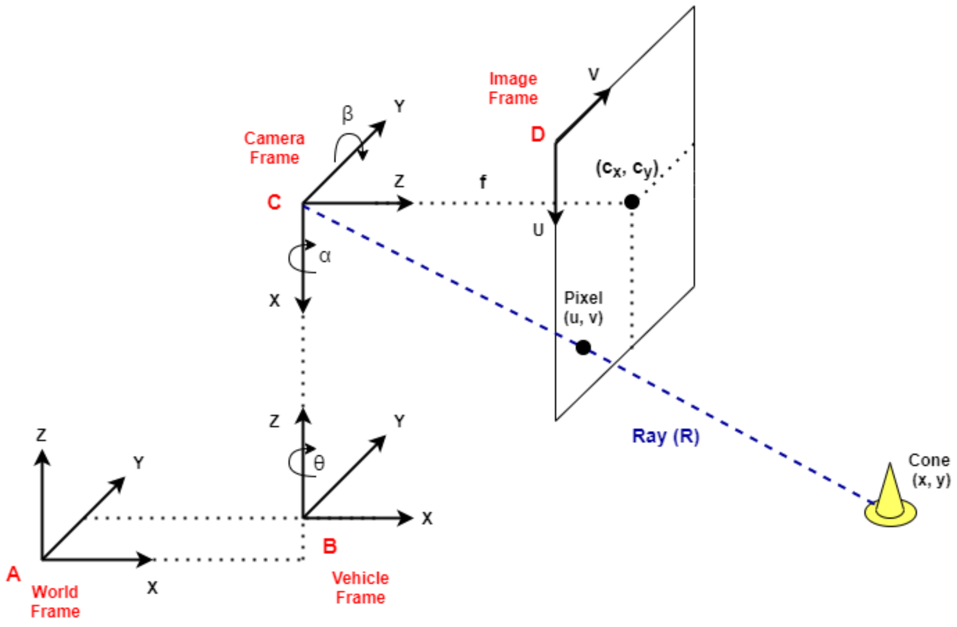

Figure 10: Diagram demonstrating the position and orientations of all reference frames relevant to camera image projection transformations.

Figure 10 depicts the orientation of each of the 4 reference frames involved in the camera pixel to cone coordinate projection transformation. Given the vehicle rests on a 2D ground plane, the transformation between the world (A) and vehicle (B) reference frames involves a translational transformation $(t_x, t_y, 0)$ followed by a rotational yaw transformation about the z-axis, equal to the current heading of the vehicle (θ).

$$\overrightarrow{AB} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

Camera(s) are placed relative to the vehicle with a vertical and horizontal translation (0, t_y, t_z). The convention for a camera reference frame requires the z (optical) axis to align to the image plane normal. Therefore, a 90° rotation about y is required, followed by further rotations about x then y to set the camera yaw (α) and pitch (β) angles. Yaw and pitch angles are predefined in the vehicle hardware, relative to the vehicle.

$$\overrightarrow{BC} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

Finally, the image plane frame of reference defines the 2D virtual reference plane where cone pixels are defined. The relative location and centre of the image plane are defined by the intrinsic focal length (f) and principal point (c_x, c_y) properties of the camera. This plane exists relative to the camera frame of reference in a translation of "f" in the Z axis and "c_x, c_y" in the x and y axes.

$$\overrightarrow{CD} = \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 1 & f \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

In order to transform the detected cone pixel produced by the cone detection algorithm from the image frame of reference to the actual coordinates within the world frame of reference, the transformation matrices defined above are required.

The camera has an internal intrinsic matrix that maps the 3D environment to a 2D image plane. The image plane is aligned normal to the camera z axis, at a focal distance "f" from the camera projection centre. The centre of the image plane is denoted by the principal point (c_x, c_y) the coordinates of which are defined relative to the local plane origin. Using these parameters, the intrinsic camera matrix (P) is defined to map 3D world coordinates (X, Y, Z) to the image plane.

$$P = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{17}$$

Within the camera reference frame, a pixel coordinate (u, v) on the image plane has the direction vector (u-c_x, v-c_y, f) or ((u-c_x)/f, (v-c_y)/f, 1) relative to the centre of projection. This direction vector represents a ray from the camera frame of reference intersecting the cone location in the 3D plane. The ray vector (R) is calculated by multiplying the pixel vector (u, v, 1) by the inverse projection matrix (P^{-1}).

$$R = P^{-1}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{18}$$

$$R = \begin{bmatrix} \frac{1}{f} & 0 & -\frac{c_x}{f} \\ 0 & \frac{1}{f} & -\frac{c_y}{f} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{19}$$

Given that the cone objects are known to lie on 2D ground plane, a simplified camera projection method can be used to determine the cone coordinates, known as back projection. The ray R that intersects both the image pixel and the projected cone position, must be rotated using the camera rotation matrix $\overrightarrow{BC}$ (15) to align with the camera orientation, transforming the ray to the vehicle frame of reference. The rotated ray now intersects the ground plane of the simulation at the approximate position of the cone. This intersection point provides a coordinate for the cone within the vehicle frame of reference. The final transformation required to convert the cone coordinates to the world reference frame is a rotation using the vehicle rotation matrix $\overrightarrow{AB}$ (14).

4.2.4 Sensor Fuzzing

To simulate noise associated with real hardware sensors, the simulator can add a variable quantity of both Gaussian and Salt and Pepper noise to the virtual camera feed. In Figure 11 A & B, the maximum gaussian noise (1×10^{-4}) and the maximum salt and pepper noise (1×10^{-2}) were applied separately and then together in C.

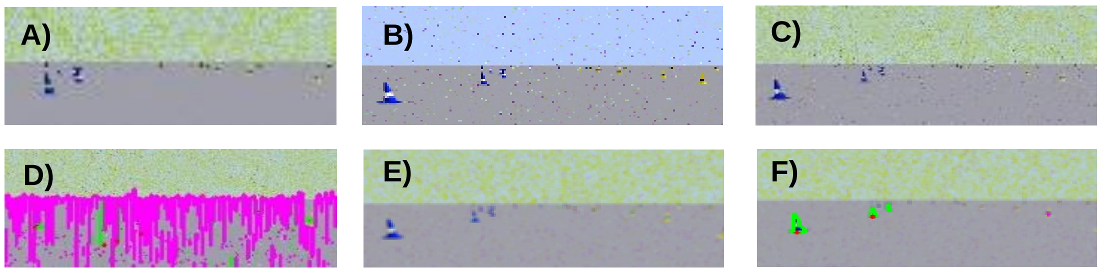

Figure 11: Camera image fuzzing and correction (cropped, magnified)

The labelled frame with both noise effects applied can be seen in Figure 11 D. The pixel noise causes random background pixels to fall within the masking colour ranges, primarily affecting the yellow masking range and preventing cone detection. To address this, a smoothing convolution operation was applied to fuzzed images before cone detection. The smoothing convolution averages the pixel colour values within the local range defined by the structuring kernel, resulting in a reduction of pixel noise. A 5x5 kernel “S” was defined to account for the maximum noise output of the simulator.

$$S = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{20}$$

The result of the smoothing convolution on Figure 11 C is shown in E and the final labelled image masking the smoothed image is shown in F. The smoothing operation reduces the clarity of perceived cones, but significantly increases object detection performance when sensor noise is introduced.

4.2.5 Discussion

An attempt was made to integrate the neural network developed by team 473 as part of the 2020 University of Leeds masters project. However, the neural net was unable to detect cones within the simulation [30]. Instead, an opensource “You-Only-Look-Once” CNN was implemented. It was developed using tensor flow and trained using sample images from the COCO (Common Objects in Context) image database [31].

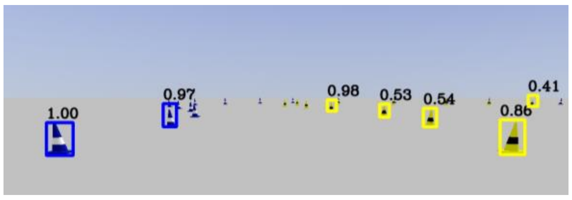

Figure 12: Opensource CNN [31] utilised in simulation

The data output is similar to OpenCV processing but the object detection is more reliable, especially when noise is introduced. The CNN was excluded from the final system due to high computation times which produced a frame rate of approximately 0.5 FPS on the host computer. This was insufficient to produce viable results for simulation testing. However, optimisation of a similar neural network model will result in a more reliable camera detection system.

4.3 LiDAR Sensor

4.3.1 Simulated Sensors

As the simulator was unable to implement an official Gazebo ROS LiDAR plugin, a “Faux Lidar” simulated sensor was created to output similar point cloud information as a real LiDAR sensor. Thus, enabling the driver algorithm to incorporate this data into the perception module.

The Faux LiDAR sensor generates a circle of projection rays at a resolution (simulating laser scans) centring on the vehicle and uses the track map information available to the simulator to return the coordinates where a laser ray intersects a cone circle (2D planar slice of a cone). This generates a 2D point plane that can be used by the driver to estimate cone positions. A real LiDAR sensor generates data in the form of a 3D point cloud, the faux lidar is limited to only a planar slice of this information.

4.3.2 Cone Detection

The coordinate data points generated by the Faux LiDAR are inherently relative to the world frame. Therefore, reference frame transformations are not required. However, the raw coordinates are presented in polar form with a range (r) and an angle from the world frame x-axis (θ). Given that the coordinates are 2D and parallel to the ground plane, conversion to cartesian coordinates does not consider an elevation angle. The final cartesian conversion can be calculated using the following equations:

$$x = r\cos(\theta) \tag{21}$$

$$y = r\sin(\theta) \tag{22}$$

To convert LiDAR ray detections into singular cone locations, the data points are clustered by Euclidean proximity then averaged, returning a final coordinate for the cone location within the world frame.

The designed clustering algorithm splits the original flat coordinate array into several sub arrays where each coordinate within the sub array is within a set proximity threshold from another coordinate.

1. Select a point (locus) from the coordinate plane array
2. Create a sub array cluster
3. Iteratively check the Euclidean distance between the locus and all other points in the plane. Add all points with a Euclidean distance less than the threshold to the sub array cluster.
4. Remove the locus from the array
5. Iterate through the sub array cluster points, repeating steps 3-4 adding other plane coordinates that meet the threshold criteria to the cluster.
6. Repeat steps 1-5 until no points remain in the coordinate plane array
7. Calculate a mean (x, y) coordinate within each cluster

An estimation of the cone's existence probability can be determined by scaling the number of points in each cluster. The number of coordinates within a cluster represents the number of LiDAR rays that have detected an object in this region. Therefore, the size of the cluster is proportional to the cone existence probability. The maximum cluster size was capped at 10 rays, this value was divided by a scale factor of 5 to ensure an input range between 0-2. It is input into a hyperbolic tangential function (tanh), the final output of which is a probability where a cluster size of 10 will generate a probability of 1. The tanh scale was chosen as it provides a more accurate representation of cone existence certainty as the cluster size increases. The difference in certainty between a cluster size of 1 and 2 is large whereas, the difference between 9 and 10 is minimal.

4.3.3 Sensor Fuzzing

Sensor fuzzing was also applied to the Faux LiDAR data, introducing variance to both the angle and the range values of the raw data. Despite this, minimal further action was required to compensate for the increased variance as the averaging property of the cluster algorithm removes the error introduced by noise.

4.3.4 Discussion

While this module is successfully able to extract cone data from the faux LiDAR sensor, introduction of a real LiDAR sensor plugin will greatly increase the raw data quality and display a much higher resolution of the environment. In this case, proximity clustering would result in the loss of valuable data quality. Several more advanced LiDAR processing and sensor fusion techniques are discussed in literature including sensor layering and object tracking algorithms [21]. If accompanied by the development of a CNN, such algorithms would generate much richer perceptions of track environments.

4.4 Results

4.4.1 Module data output

The perception module successfully extracts the key information from available LiDAR and Camera sensors, standardises cone data in the form (position, probability) and categorises cones by colour if applicable. The final output is 3 structures representing all perceived cones in the most recent sensor update: Blue, yellow and unknown cones. These cone collection arrays are published individually by the perception module allowing trivial interpretation and categorisation in the mapping algorithm.

4.4.2 Detection accuracy

The detection accuracy of the sensors was examined and optimised extensively using the simulator’s dashboard and API functionality. Camera detection was tested and optimised iteratively using a variety of lighting conditions, ground textures and fuzzing operations. The resulting image pipeline successfully accounts for all factors present by the simulated environment.

Similarly, performance tests were conducted on the Faux LiDAR. The primary inhibiting factor of a LiDAR sensor is the environmental weather conditions, a parameter currently unavailable in simulation. However, the sensor responded well to both gaussian and salt and pepper noise, allowing accurate data to be recorded.

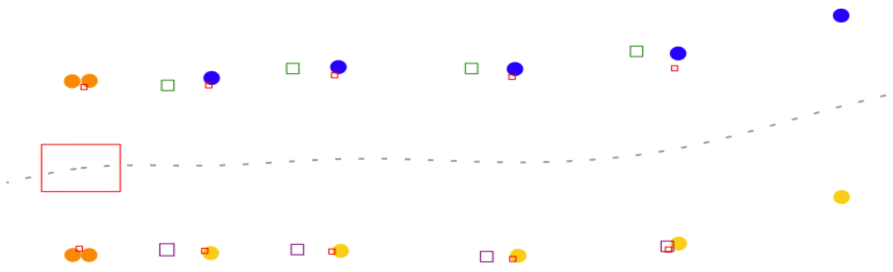

Figure 13: Screenshot of map displayed by the simulator comparing perceived cone locations by the camera and LiDAR to actual cone locations

Figure 12 demonstrates the discrepancy between the perceived cone locations as detected by each sensor and the actual cone locations with no sensor fuzzing. The smaller red squares indicate the cones' estimated positions as detected by LiDAR. Cone locations are estimated with a high degree of accuracy (on average +/- 0.1 m from the cone centre of gravity (CoG)). Green and purple squares indicate the estimated location of blue and yellow cones respectively. The camera consistently underestimates the location of the perceived cone with errors increasing the closer the cone is to the vehicle (on average +/- 0.5 m from the cone CoG). This is predominantly caused by the lowest pixel approximation used to represent each cone in the masking stage. The lowest pixel will always appear before the cones CoG. An immediate improvement to this estimation could be consideration of the cones' dimensions and angle to the camera to better estimate a ray to the cone centre. In the long term, the OpenCV based camera perception should be replaced with a CNN for drastic improvement in detection and estimation accuracy. The inclusion of full LiDAR sensors able to generate a full 3D point cloud will greatly increase the perception possibilities and overall map resolution.

Chapter 5: Mapping and Path-planning Module

5.1 Module Overview

The mapping and planning module is responsible for: subscribing to all available processed sensor data streams published by the Perception module; generating and updating a track map and using this to determine an appropriate vehicle trajectory. The output of this module is a set of coordinates in the world frame, indicating the vehicle trajectory. A single mapping algorithm and two planning algorithms were developed.

5.2 Mapping

5.2.1 Cone Representation

Cone data as perceived by the external sensors describes each cone with a position (x, y) and an existence probability. In order to represent and update the cone map efficiently, each mapped cone is represented by a 2D Kalman filter with a gaussian distribution. The estimation of the variable is given by a gaussian distribution with a mean (μ) and a variance (σ^2) where the mean indicates the centre of the gaussian estimation curve and the variance indicates the width or certainty.

The location of a perceived cone in the x direction is estimated by a Kalman filter with a mean of the cone’s x-coordinate and a variance inversely proportional to the cone’s existence probability. A second Kalman filter is defined for the cone’s y-coordinate, generating a position estimation on a 2D plane with a corresponding location certainty.

Once a 2D Kalman filter has been defined to represent a cone, its properties are continually updated to reflect new sensor measurements. Numerous functions exist to update Kalman filters. For the purposes of cone mapping, a merging operation was selected whereby new cone measurements that fall within a set radius threshold of another cone location will not generate a new Kalman filter and will instead update the coordinate means and variance of the existing cone estimation. Existing cones that migrate towards one another over a series of measurements are also merged.

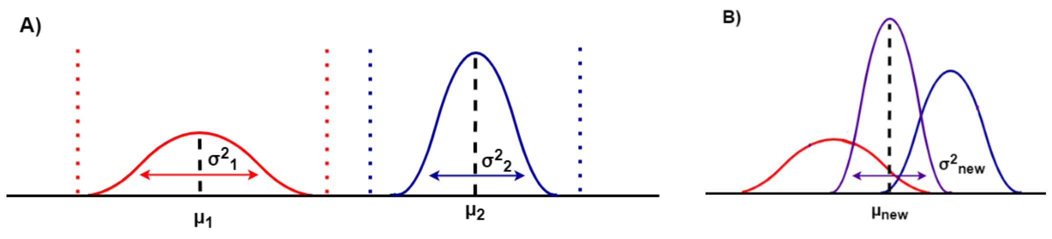

Figure 14: Diagram demonstrating the merge thresholds (A) and the merge operation result (B) using 1D Kalman Filters

New coordinate means of the resulting merged cone are calculated using the formula:

$$\mu_{new} = \frac{\sigma^2{}_1\,\mu_2 \ + \ \sigma^2{}_2\,\mu_1}{\sigma^2{}_1 \ + \ \sigma^2{}_2} \tag{23}$$

Where the position of the cone with the lowest variance and therefore highest certainty is favoured. The variance is updated using the following formula:

$$\sigma^2{}_{new} = \frac{1}{\frac{1}{\sigma^2{}_1} + \frac{1}{\sigma^2{}_2}} \tag{24}$$

The resulting merged variance will always be less than both the original variances, increasing the certainty of a cone within that region.

5.2.2 *Sensor Fusion Mapping*

Cone data is received from the camera sensors and the LiDAR sensor. While both generate cone information in a normalised form, the camera sensor is able to provide a cone colour which is essential in determining the left and right track boundaries. Cones detected by the cameras are mapped in cone collection data structures corresponding to their detected colour. This produces a collection of blue cones and a collection of yellow cones, each with a position vector and a variance. A third collection of cones was defined to map cones of unknown colour as perceived by the LiDAR. With each new sensor measurement update, the unknown cones are iteratively checked against the coloured mapped cones. If an unknown cone is within the merge radius of a coloured cone, the two are merged, keeping the coloured property.

Several pruning operations are performed to reduce the certainty (increase the variance) of mapped cones that satisfy a set of conditions. Mapped cones that were not perceived in the last sensor update but exist either within the camera FOV or the LiDAR detection radius, receive a variance increase proportional to their proximity to the vehicle. Mapped cones that exceed a predefined variance threshold are removed from the map. These pruning algorithms eliminate false sensor updates over a series of correct sensor measurements and validate the accuracy of the internal world map.

5.2.3 *Forgetful mapping*

The mapping algorithm implemented in this module is reliant on the accuracy of the simulated odometry. To better mimic real world sensor data, the simulator adds both heading and position error to the perfect odometry data generated by Gazebo. This prevents the vehicle from correctly updating its position within the world frame. Despite this, the mapping algorithm is able to correctly identify cone positions relative to the vehicle allowing the vehicle to correctly navigate the track on the first lap. The disparity in actual and internal vehicle position becomes apparent at the beginning of the second lap. The coordinate drift of the vehicle prevents the internal map from connecting with the start of the track, causing severe performance issues and often causing unsuccessful completion of the second lap. Figure 15 displays the difference between the actual (red square path) and predicted (blue square path) positions of the vehicle when odometry fuzzing is introduced. It can be seen that despite the positional discrepancy, the vehicle successfully navigates the track as expected.

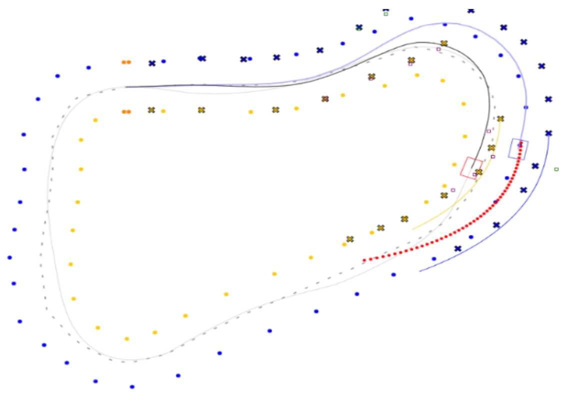

Figure 15: Actual path vs perceived path using imperfect odometry

A simple solution to coordinate drift was devised where the mapping algorithm forgets the location of previous cones, only mapping the track within the vehicle reference frame. This allows the vehicle to correctly navigate laps as required.

5.2.4 *Discussion*

A superior solution to the issue created by imperfect odometry that will allow mapping within the global frame of reference is the implementation of a simultaneous localisation and mapping algorithm (SLAM). A SLAM algorithm utilises sensor data from both the external camera and LiDAR sensors, as well as additional internal vehicle sensors to cross validate and localise the vehicle position, whilst mapping the track in the world reference frame. The design of a SLAM algorithm is imperative to the optimisation of the mapping algorithm prior to competition entry. Numerous efficiency benefits are gained via track learning, enabling significantly faster subsequent lap times as is the goal for any racing algorithm.

5.3 Planning

Two distinct path planning algorithms were designed, both of which attempt to generate a set of coordinates that trace the centreline of the perceived track.

5.3.1 Closest-pair Centreline Generation

The closest-pair path-planning algorithm simply calculates the midpoint between each cone and it closest pair from the opposing colour set. Given that the colours are always on opposite sides of the track centreline, the midpoint between any adjacent pair is guaranteed to lie on the centreline of the track. This trajectory calculation is heavily reliant on the accuracy of sensor measurements and was found to be exceptionally volatile in simulation testing. Furthermore, the lack in quantity of trajectory coordinates was unfavourable for control signal generation in the vehicle control module. This algorithm serves as a baseline for trajectory generation and highlighted numerous areas for improvement via simulation evaluation.

5.3.2 Wall-building Centreline Generation

To address the issues exposed by the closest-pair algorithm, a second path-planning algorithm was developed. The wall-building algorithm aims to approximate the coloured cone coordinate track borders as continuous walls, allowing filtration of anomalous cone data and producing a smoother, continuous. The designed algorithm consists of 4 steps:

1. Select the relevant mapped cones to be included in trajectory calculation.
2. Order the selected cones.
3. Fit a spline through the ordered cone coordinates of each coloured cone collection, approximating the two track walls.
4. Calculate the midpoint between each wall coordinate pair, generating a centreline estimation.

In order to select the mapped cones that are relevant to the future pathing of the vehicle, an inclusion box in the world frame of reference is defined. The inclusion box used is 20m in the vehicle forward direction, 10m in the vehicle right and left, and 4 metres in the vehicle backward direction. Any cone coordinates within the area defined by the inclusion box are selected to be used for trajectory generation.

The cone data received from the mapping module is unordered. As demonstrated in Figure 16, it is essential that the coordinates are ordered correctly prior to spline fitting to ensure that the resultant spline wall accurately represents the geometry of the track. The "best order" is one that starts at the node closest to the vehicle and orders the points sequentially so that the total Euclidean distance travelled from the starting coordinate to the end coordinate is minimised. As such, the ordering problem can be represented as a variation of "travelling salesman" problem in graph theory. Using this representation, the path through all nodes with the smallest cost (shortest total Euclidean distance) will return a reliable approximation of the track geometry. This algorithm was further optimised to exclude anomalies where, given three sequential cones (A, B and C) the middle cone (B) is excluded if: $\overrightarrow{AB} + \overrightarrow{BC} > 2\,\overrightarrow{AC}$.

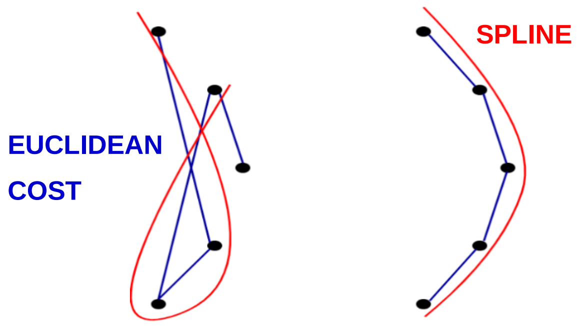

Figure 16: Diagram demonstrating the requirement for cone ordering prior to spline-fitting

Finally, using the correctly ordered set of cone coordinates defined for both the yellow and the blue cone collections, a spline is fit through each data set, approximating a track wall. The spline function chosen for this task was SciPy's Univariate spline function. This function has favourable properties that allow a weighting to be applied to each point and a smoothness factor to be chosen. Each cone was given a weighting of the inverse of its variance, meaning the spline will prioritise the cones with the highest existence probability. As a smooth spline is favoured over one that directly interpolates through all existing coordinates, a high smoothness factor was selected. By calculating the midpoint between the blue and yellow wall spline coordinate pairs, a centreline estimation of the track is generated with arbitrary resolution.

5.3.3 Discussion

During the first lap, while the track is being learned and the future trajectory is uncertain, it is best to maintain a central position with regard to the track to avoid incurring any rule violations or time penalties. However, as the primary goal of the driving algorithm is complete a given track in the fastest time possible, subsequent laps may employ a separate trajectory calculation utilising the full map layout. Many advanced autonomous systems employ a racing trajectory calculation that allows the vehicle to hit the apex of every known corner, thus calculating the feasible shortest path through all track way points. Such an algorithm will be exceptionally beneficial in the FS-AI competition and its development is advised as future work.

5.4 Results

5.4.1 Module Output

After the perceived cones are processed via the mapping and planning algorithms, a set of coordinates indicating the calculated vehicle trajectory is published as the module output. This coordinate set is used by the control module to determine appropriate actuator outputs.

5.4.2 Validation of the mapping and planning module

A mapping algorithm was created that enables the fusion of all available data streams into a single coherent and accurate map of the vehicle environment. Two planning algorithms were created, both generating centreline trajectories that accurately represent the absolute track centreline provided accurate sensor information is recorded. Figure 17 demonstrates the accuracy of the implemented planning algorithms, tested using a single, forward facing camera sensor.

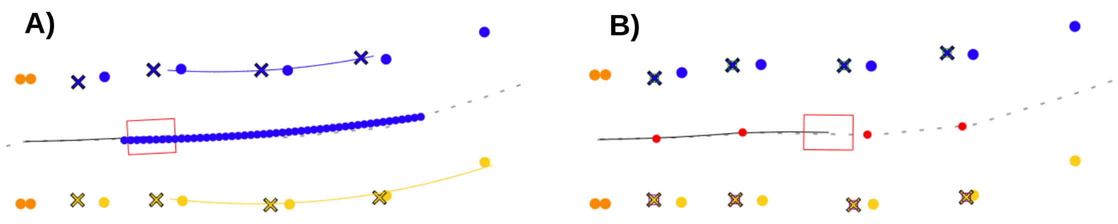

Figure 17: Wall-building (A) and closest-pair (B) centreline calculations relative to the absolute centreline

In simulation testing it was found that the mapping and planning algorithms were limited by the accuracy and reliability of the perception measurements. Precautions were taken to reduce the impact of erroneous cone measurements in the form of pruning algorithms. Aggressive pruning risks the accidental removal of correct cone measurements whereas passive pruning does not remove incorrect cones quick enough, resulting in a faulty cone being temporarily included for path planning. The effects of this were most prominent in the closest pair algorithm. The smoothing, weighting and exclusion of anomalous cones in the wall-building trajectory provided a significant increase to performance and reduced sensor measurement noise.

Chapter 6: Vehicle Control Module

6.1 Module Overview

The vehicle control module subscribes to the path trajectory data published by the mapping and planning module and determines actuator command signals which allow the vehicle to accurately trace the calculated trajectory. The controllers implemented in this module utilise the kinematic bicycle vehicle models defined in Chapter 2 section 2.6. As such, this module is responsible for the generation of an acceleration command for the simulated motor actuator and a steer angle for the simulated steering wheel actuator.

Three Lateral controllers were implemented along with a single Longitudinal controller to provide varied data for simulator validation and testing.

6.2 Longitudinal Controller

6.2.1 PID Controller

The longitudinal controller is responsible for determining the appropriate acceleration (a) control signal that will allow the vehicle to transition from the measured current vehicle velocity ($v_{current}$) to the target velocity (v_{target}). The PID control law expressing this functionality in the continuous time domain can be seen below, where the Error function, traditionally associated with a PID equation is replaced with the velocity error.

$$a = K_P\left(v_{target} - v_{current}\right) + K_I \int_0^t \left(v_{target} - v_{current}\right) dt + K_D \frac{d\left(v_{target} - v_{current}\right)}{dt} \qquad (25)$$

To implement this controller in real-time, the above continuous PID control law was discretised into individual time steps, allowing updates to be calculated in relation to previous timestep updates. In the discrete domain, the integral term is represented as a summation of all previous velocity errors at each timestep and the derivative term is approximated as the difference between the previous and current error measurement. The proportional error remains unchanged and is represented as the signed difference between the target and current velocity scalars.

The optimal controller constants for to achieving the desired steady state with minimal overshoot and the shortest possible rise time were found using the Ziegler-Nichols iterative PID controller tuning algorithm utilising the simulator's API tuning functionality.

6.2.2 Velocity Target using Gaussian Path-curve Weighting

To implement the longitudinal controller described above, an appropriate velocity target must be selected, in keeping with the limitations of the simulated actuators. Racing vehicles must complete laps with the highest average velocity to decrease lap times. It is also vital that control is maintained and no FS-AI rules are violated, inciting time penalties. To this end, the velocity target should encourage acceleration on track straights and decelerate on corners. Determining the highest possible cornering speed while maintaining control and position on the track is of high priority in a racing system.

An algorithm was developed to examine the future path curvature and represent this quantity as a single value. The inverse of this value is then used to scale the target velocity for the PID. The designed algorithm follows the steps laid out below:

1. Translate the path trajectory to the vehicle frame of reference
2. Differentiate the path twice, calculating the rate of change of gradient at each path coordinate.
3. Calculate the Euclidean distance between each path coordinate and a specified point, X metres in front of the vehicle (lookahead point).
4. Apply a gaussian weighting to each path curvature value, using the distance of the point to the specified lookahead point.
5. Sum all weighted curvature values, resulting in a single value representing the curve of the track with higher priority to the region X metres in the direction of the vehicle forward.

The gaussian weighting function shown in equation 26 was selected. The parameter “b” is the position of the centre of the peak and “c” controls the width of the curve. In this application, constants “b” and “c” are both set to the distance between the vehicle and the lookahead point, ensuring the weighting is centred on the lookahead and the weighting reaches 0 at vehicle position. The input “d” is the Euclidean distance between the path point and the vehicle lookahead point.

$$Gaussian(d) = e^{\left(-\frac{(d-b)^2}{2c^2}\right)} \tag{26}$$

For each path coordinate (p) the gaussian weighting output value (Gaussian(d)) is multiplied by the previously calculated curvature at “p” (c). The cumulative summation of this calculation across all path coordinates produces a single value (PC) representing the path curvature in the lookahead region.

$$PC = \sum \left(c \cdot Gaussian(d)\right) \tag{27}$$

The value “PC” can be scaled so that the range of values exists between 0-1, where 0 represents a perfect straight and 1 represents an extreme curve. All path curvatures that exceed a defined maximum curvature value saturate to 1.

The target velocity is calculated with PC and the physical limit of the velocity (v_{max}).

$$v_{target} = v_{max} - PC\ v_{max} \tag{28}$$

6.3 Lateral Controller

The lateral vehicle controller is responsible for determining an appropriate steering angle (δ) that will correct the vehicle heading and allow it to follow the calculated path trajectory. Three lateral control algorithms were implemented to provide a variety for simulation testing. The lateral error measurements utilised in this module are discussed in detail in Chapter 2 section 2.7.

6.3.1 Semi-Quadratic Path Approximation

The semi-quadratic approximation controller is the simplest and least documented in official publications of the 3 lateral controllers designed in this project. It is not often considered for serious racing purposes due to its inability to meet all path waypoints.

The semi-quadratic path approximation, approximates a quadratic function through the path coordinate data set within the vehicle frame of reference. The semi-quadratic equation removes the “x” term (as seen in equation 29). Fitting a semi-quadratic function to the set of path coordinates within the vehicle frame of reference ensures that the vertex is located on the vehicle right vector (y-axis). The “a” coefficient determines the direction and magnitude of the path curvature, providing a basis for a suitable steer command. The “c” coefficient approximates the instantaneous cross-track error and can be used to scale the strength of the steer command.

$$y = ax^2 + c \tag{29}$$

The final steer command is proportional to the distance coefficient and signed square of the curvature coefficient. Coefficients “K_d” and “K_c” were tuned experimentally.

$$\delta = K_d c + K_c(a\ |a|) \tag{30}$$

6.3.2 Pure-Pursuit Geometric Controller

The pure-pursuit controller is among the most popular geometrical lateral control algorithms in literature. The pure-pursuit algorithm allows a significant increase in path-following accuracy and decreases the average cross-track error in comparison to the semi-quadratic controller.

The pure-pursuit algorithm requires selection of a point on the calculated path at a specified lookahead distance (L_d). Using the rear axle bicycle model, the angle (α) can be calculated between the vehicle forward vector and the lookahead path point direction vector.

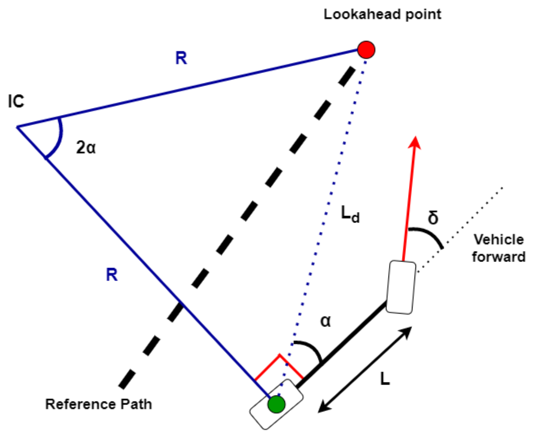

Figure 18: Geometry defining the Pure-pursuit controller

Using the law of sines, an equation can be derived relating the distance to the centre of rotation (R), the angle to the lookahead point (α) and the lookahead distance (L_d):

$$\frac{R}{\sin\left(\frac{\pi}{2} - \alpha\right)} = \frac{L_d}{\sin(2\alpha)} \tag{31}$$

$$\frac{R}{\cos(\alpha)} = \frac{ld}{2\sin(\alpha)\cos(\alpha)} \tag{32}$$

$$\therefore R = \frac{L_d}{2\sin(\alpha)} \tag{33}$$

This relationship can be used to determine the steering angle (δ). It can be seen from the rear-axle bicycle model (Figure 1) that:

$$\delta = \tan^{-1}\left(\frac{L}{R}\right) \tag{34}$$

Therefore, by substitution of equation (33) into (34), it can be shown that:

$$\therefore \delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{L_d}\right) \tag{35}$$

At constant velocity, a smaller lookahead distance will induce a stronger steer control response to cross track error, often causing a degree of oscillation but an increased response time. A distant lookahead will cause a weaker control signal, increasing ride smoothness but delaying response time.

In race settings, varying the lookahead distance proportionally to vehicle velocity enables weaker control signals on straights and more aggressive turns as velocity decreases.

$$\delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{K_v\, v}\right) \tag{36}$$

6.3.3 Stanley Geometric Controller

The Stanley geometric controller is a controller designed by Stanford Universities' 2005 DARPA Grand Challenge autonomous racing team. This controller was designed to minimise two error terms and operate independently of vehicle velocity. The Stanley controller uses the front axle bicycle model.

The Stanley controller steer output law is comprised of two parts: firstly, the heading error is accounted for by addition of the current heading error (ϕ); secondly, the cross-track error is accounted for by addition of the arctan term where "E" is the current signed lateral cross-track error and "v" is the magnitude of the velocity.

$$\delta = \phi + \tan^{-1}\left(\frac{K\,E}{K_s\ +\ v}\right) \tag{37}$$

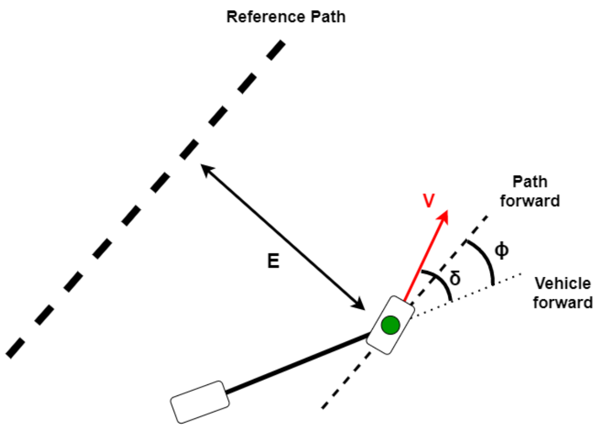

Figure 19: Geometry defining the Stanley controller

The smoothing coefficient “K_S” in the denominator of the arctan term prevents controller value errors in the case of velocity equalling 0. Additionally, a higher “K_S” value reduces the strength of the controller response to cross-track error. As there is no lookahead point associated with the controller, the overall ride is not as smooth as its pure-pursuit counterpart. The “K_S” value reduces the volatility of the controller. However, if it is set too high, it will render the velocity input redundant.

6.4 Results

6.4.1 Final Module Output

The vehicle control module produces two signed outputs: the steering angle target (radians) and the acceleration target (m/s^2). These are published as ROS messages to the Gazebo-ROS actuator interface for further processing by the internal vehicle PID controllers ensuring the target values are exhibited in the simulated vehicle.

6.4.2 Comparison of Controller performance

Extensive testing was completed validating the function and performance of the implemented controllers. All tests were performed using: the forward-facing camera and LiDAR in the perception module, the wall building trajectory mapping and planning module and each of the lateral controllers. Each driver combination was required to complete two laps of 10 of simulator tracks. The average lap time, average RMS cross-track error (CTE) and average incurred FS-AI penalties were recorded across all tracks. The results of the tests are displayed in Table 1. If the driving algorithm causes the vehicle to fully leave the defined track boundary, that track was marked as a failure and the data was excluded from the tabulated averages. Example simulator data collected from a single-track test is displayed in Appendices i, ii and iii.

Table 1: Lateral Controller Simulation Test Results

Lateral Controller	Average Time (s)	Average Penalties (s)	Average RMS CTE (s)	Number of Tracks completed (/10)
Semi-Quadratic	45.57	5.67	1.37	6
Pure-pursuit	42.93	0.67	0.31	10
Stanley	51.44	3.33	0.45	9

The results of the simulation testing show the Pure-pursuit controller to be the best lateral control algorithm across all categories. The lookahead nature of the Pure-pursuit allowed a smoother, more controlled response to reference path fluctuations, which provided numerous benefits. As displayed in the exemplar vehicle path diagrams in Appendices i, ii and iii, the Pure-pursuit algorithm was best able to trace the absolute centreline of the example track resulting in the lowest average RMS cross-track error and the lowest average incurred penalty time. The smoothness of the steer response (and resultantly, path) allow the longitudinal controller to operate as intended. The effect of this is seen in both the G-G diagrams and the consequential average lap times. Fewer lateral fluctuations allow the vehicle to accelerate predominantly in the longitudinal direction and achieve higher overall velocities.

While the Stanley controller was able to trace the centreline with reasonable accuracy, the roughness of the ride resulted in a much lower overall acceleration (Appendix iii, G-G diagram) and an increase in the total distance travelled contributing to the overall slower lap times. The primary cause of ride roughness was identified as unpredictable fluctuations in the perception and consequently mapping modules. As the Stanley bases its control output on the immediate path, fluctuations of the path cause fluctuations of the control signal. The addition of a lookahead significantly reduces the effects of sensor fluctuation but it does not address the root of the issue.

Finally, the semi-quadratic approximation performed well on tracks with gentle curves and corners with large turning radii, achieving similar results to the Pure-pursuit. However, as the approximation considers the full future curvature of the track with equal weighting, generated steer commands are often too strong and turns begin too early, causing the vehicle to turn sharply before the apex of the corner. This often results in cone collisions, incurring penalties. In some cases, vehicle collisions cause the vehicle to tilt and produce erroneous sensor measurements severely, impacting the planning performance. This is the primary cause of the 4 track failures.

Overall, the success of the implemented control algorithms was demonstrated and the vehicle evaluation functions of the simulator were thoroughly tested and validated.

Chapter 7: Conclusion

7.1 Achievements

This project successful completed all the aims and objectives presented in Chapter 1. A modular autonomous driving algorithm was designed and developed alongside the virtual simulator, demonstrating its use in algorithm development and its importance to the future entry of the University of Leeds into the FS-AI competition.

A perception module capable of extracting and standardising key information features from both camera and LiDAR sensors was developed. Functionality was tested and validated in simulation to ensure performance was maintained even with the introduction of noise, varying lighting conditions and background textures.

A mapping and planning module was developed which was capable of utilising sensor fusion techniques, combining a variety of sensor data streams and building an accurate internal representation of the world map. Two trajectory calculation algorithms were designed, both able to calculate an appropriate centreline trajectory and one able to account for anomalous cone positioning and sensor measurement error.

A vehicle control module was implemented with a single longitudinal and three lateral controllers. Each of the lateral controllers produce a distinct control response providing excellent test data for validation of the simulator’s vehicle evaluation functions.

In total 72 driving algorithm combinations were implemented and evaluated using the simulator. This modular driver will provide a template for the development of an advanced, racing optimised driver for entry into the FS-AI competition.

7.2 Future Work

Several areas of future work have been identified for each of the implemented driving modules. Perhaps the biggest performance limiter in both this specific system and autonomous vehicle AI in general is vehicle perception. The implemented perception algorithms are simplistic. The current camera object detection algorithm examines only the cones colour properties. An advanced CNN is able to verify object classification using shape, colour and context significantly increasing detection accuracy even if the target object is partially hidden. Design and implementation of a CNN is typically a complex task involving an extended period of model training and verification. However, a competent CNN is a vital addition to an AI driving algorithm and is utilised by all top autonomous racing systems.

The implementation of a simultaneous localisation and mapping (SLAM) algorithm represents another significant challenge in the generation of a racing optimised autonomous driver. At present, imperfect odometry is addressed via local mapping, where cones outside a certain radius from the vehicle are removed from the map. This prevents the problems caused by coordinate drift and allows the vehicle to successfully navigate a track using the relative position of the cones despite the discrepancy between the predicted and actual vehicle locations. While effective, the efficiency and speed benefits gained by track learning are lost. Instead, object tracking algorithms can be integrated with the current perception module, allowing objects to be tracked between sensor updates and their relative positional change recorded relative to the vehicle. The displacement of environmental landmarks can be used to cross-validate the internal odometry sensor outputs, localising the vehicle position and allowing accurate mapping to resume within the world reference frame.

While the geometric lateral controllers implemented in this project are adequate for simulation testing purposes, they do not account for specific vehicle physics and thus their performance reduces significantly in real-world application. At high speeds, inertial and tire forces are large enough to significantly affect the handling characteristics of the vehicle. These forces must be compensated for by the controller to ensure accurate course is maintained. Model predictive controllers are vehicle controllers that develop a predictive cost function using an advanced model of the full specific vehicle dynamics to estimate the forces experienced by the vehicle at specific speeds and turn angles, providing a compensated control signal. An optimised MPC will greatly increase handling performance in physical deployment and competition settings.

Provided the perception module is improved and the measurement fluctuations are reduced, an alternative planning algorithm could be developed to calculate a racing line trajectory enabling the vehicle to hit the apexes of each track turn. This algorithm will enable the vehicle to plan a minimum distance trajectory around the entire track, optimising the completion times as required for competition.

References

[1] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barries and policy recommendations," *Transportation Research Part A: Policy and Practice,* vol. 77, pp. 167-181, 2015.

[2] A. Millard-Ball, "Pedestrians, Autonomous Vehicles, and Cities," *Journal of Planning Education and Research,* vol. 38, no. 1, pp. 6-12, 2018.

[3] J. Yang and J. F. Coughlin, "In-vehicle technology for self-driving cars: Advantages and challenges for ageing drivers," *International Journal of Automotive Technology,* vol. 15, no. 2, pp. 333-340, 2014.

[4] S. A. Bagloee, M. Tavana, M. Asadi and T. Oliver, "Autonomous vehicles: challenges, opportunities, and future implications for transportation policies," *Journal of Modern Transportation,* vol. 24, p. 284–303, 2016.

[5] Institute of Mechanical Engineers, "Formula Student Artificial Intelligence," [Online]. Available: https://www.imeche.org/events/formula-student/team-information/fs-ai. [Accessed 22 4 2021].

[6] Defense Advanced Research Projects Agency (DARPA), "The Grand Challenge," [Online]. Available: https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles. [Accessed 22 4 2021].

[7] Roborace, "Roborace," [Online]. Available: https://roborace.com/. [Accessed 22 4 2021].

[8] M. Ahamed, G. Tewolde and J. Kwon, "Software-in-the-Loop Modeling and Simulation Framework for Autonomous Vehicles," *IEEE,* 2018.

[9] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," *IEEE,* 2005.

[10] S. Dai, J. Hite, T. Masuda, Y. Kashiba and N. Arechiga, "Control Parameter Optimization for Autonomous Vehicle Software Using Virtual Prototyping," *IEEE,* 2017.

[11] T. T. Ban, M. Currie, J. Heavey and D. Marshall, "Formula Student: Virtual Testing Environment for Autonomous Vehicles," University of Leeds, Leeds, 2021.

[12] "FS-AI - Formula Student Artificial Intelligence," IMeche, [Online]. Available: https://www.imeche.org/events/formula-student/team-information/fs-ai.
[Accessed 13th April 2021].

[13] Formula Student, “Rules,” [Online]. Available: https://www.imeche.org/events/formula-student/team-information/rules. [Accessed 21 04 2021].

[14] R. Szeliski, Computer Vision: Algorithms and Applications, Springer Science & Business Media, 2010.

[15] Intel, “OpenCV Docs,” Intel, [Online]. Available: https://opencv.org/. [Accessed 12 2 2021].

[16] M. Babiker, M. Elawad and A. Ahmed, “Convolutional Neural Network for a Self-Driving Car in a Virtual Environment,” *IEEE,* 2019.

[17] J. Kocic, N. Jovicic and V. Drndarevic, “An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms,” *NCBI,* 2019.

[18] Velodyne, “What is LiDAR?,” Velodyne, [Online]. Available: https://velodynelidar.com/what-is-lidar/. [Accessed 12 2 2021].

[19] J. Wojtanowski, M. Zygmunt and M. Kazsczuk, “Comparison of 905 nm and 1550 nm semiconductor laser rangefinders’ performance deterioration due to adverse environmental conditions,” *Opto-Electronics Review,* 2014.

[20] S. Feraco, A. Bonfitto, N. Amati and A. Tonoli, “A LIDAR-Based Clustering Technique for Obstacles and Lane Boundaries Detection in Assisted and Autonomous Driving,” *American Society of Mechanical Engineers,* 2020.

[21] J. Giacalone, L. Bourgeois and A. Ancora, “Challenges in aggregation of heterogeneous sensors for Autonomous Driving Systems,” *IEEE,* 2019.

[22] J. Culley, S. Garlick, E. G. Esteller and P. Georgiev, “System Design for a Driverless Autonomous Racing Vehicle,” *IEEE,* 2020.

[23] D. G. Perumal, B. Subathra, G. Saravanakumar and S. Sirinivasan, “Extended Kalman Filter Based Path-Planning Algorithm for Autonomous Vehicles with I2V Communication,” in *4th International Conference on Advances in Control and Optimization of Dynamical Systems, ACODS-2016*, Trichirapalli, India, 2016.

[24] W. F. Miliken and D. L. Miliken, Racecar Vehicle Dynamics, SAE International, 1995.

[25] C. Samak, T. Samak and S. Khandhasamy, “Control Strategies for Autonomous Vehicles,” Autonomous Systems Laboratory, SRM Institute of Science and Technology, Chennai, 2020.

[26] R. C. Coulter, “Implementation of the Pure Pursuit Path Tracking Algorithm,” Carnegie Mellon University, Pittsburgh, 1992.

[27] The F1 Clan, “Vehicle Dynamics: The Kinematic Bicycle Model,” 21 9 2020. [Online]. Available: https://thef1clan.com/2020/09/21/vehicle-dynamics-the-kinematic-bicycle-model/. [Accessed 20 3 2021].

[28] G. Hoffman, C. Tomlin, M. Montemerlo and S. Thrun, “Autonomous Automobile Trajectory Tracking for Off-Road Driving:,” Stanford University, Stanford, 2007.

[29] S. Thrun, M. Montemerlo, H. Dahlkamp and D. Stavens, “Stanley: The robot that won the DARPA Grand Challenge,” *Journal of Field Robotics,* no. 23, pp. 661-692, 2006.

[30] J. Hyde, P. Needham, S. Rubbani, C. Williams and J. Yeo, “Autonomous Vehicle: Development of Fully Functional Small-Scale Self Driving Vehicle,” University of Leeds, Leeds, 2020.

[31] F. Diaz, “Cone Detector TF,” 8 7 2019. [Online]. Available: https://github.com/fediazgon/cone-detector-tf. [Accessed 12 1 2021].

Appendices

i. Simulation Results of Semi-Quadratic Controller

The semi-quadratic controller was tested using the front facing camera, LiDAR and wall building algorithms. This sensor combination was tested on a variety of maps, with 2 laps completed on each. Example data collected from the simulation of one of these test maps is displayed below.

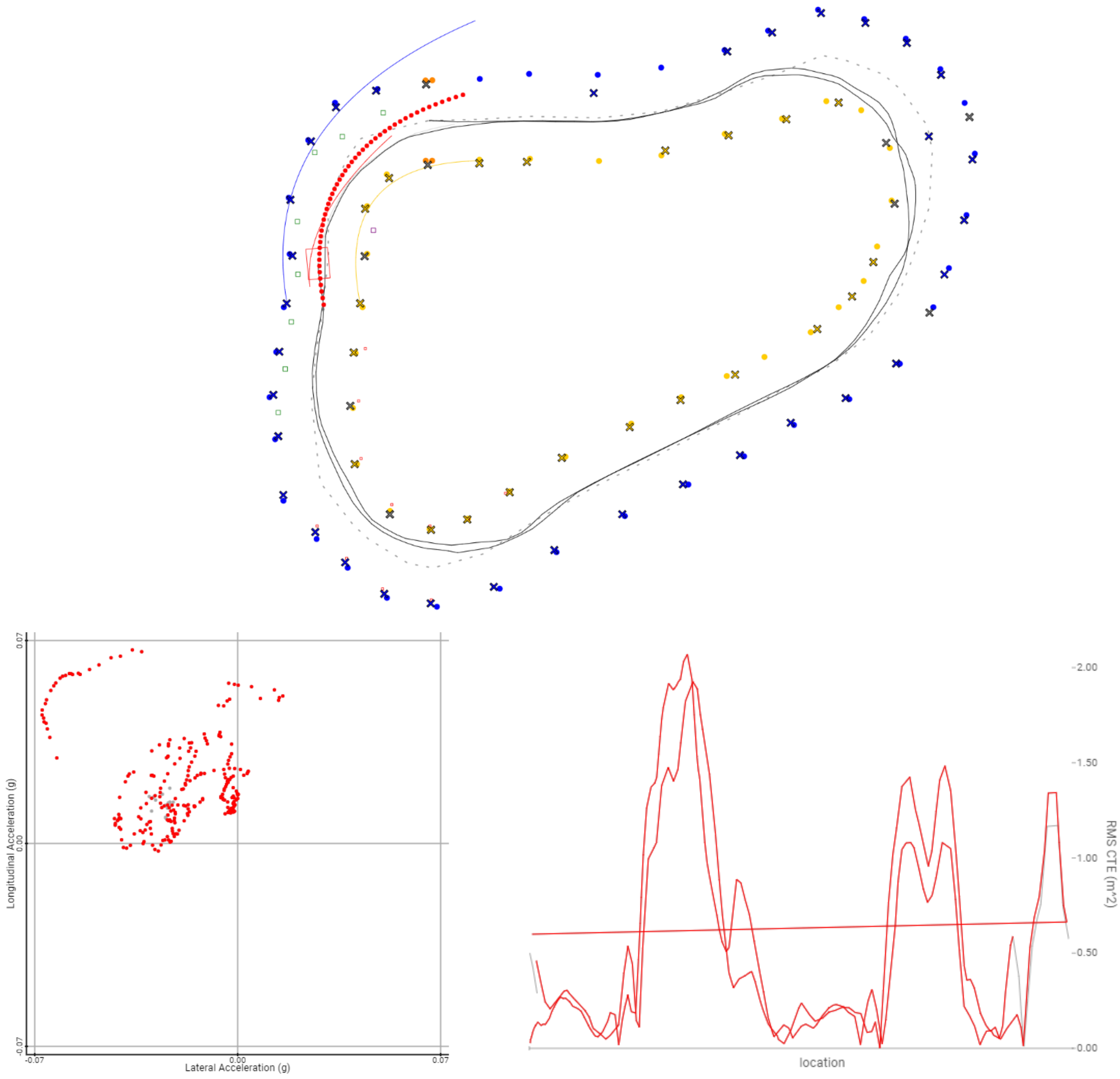

Figure 20: Semi-quadratic simulation results. Vehicle path relative to centreline (TOP), G-G diagram (LEFT) and RMS chart with respect to time (RIGHT)

ii. Simulation Results of Pure-pursuit Controller

The Pure-pursuit controller was tested using the front facing camera, LiDAR and wall building algorithms. This sensor combination was tested on a variety of maps, with 2 laps completed on each. Example data collected from the simulation of one of these test maps is displayed below.

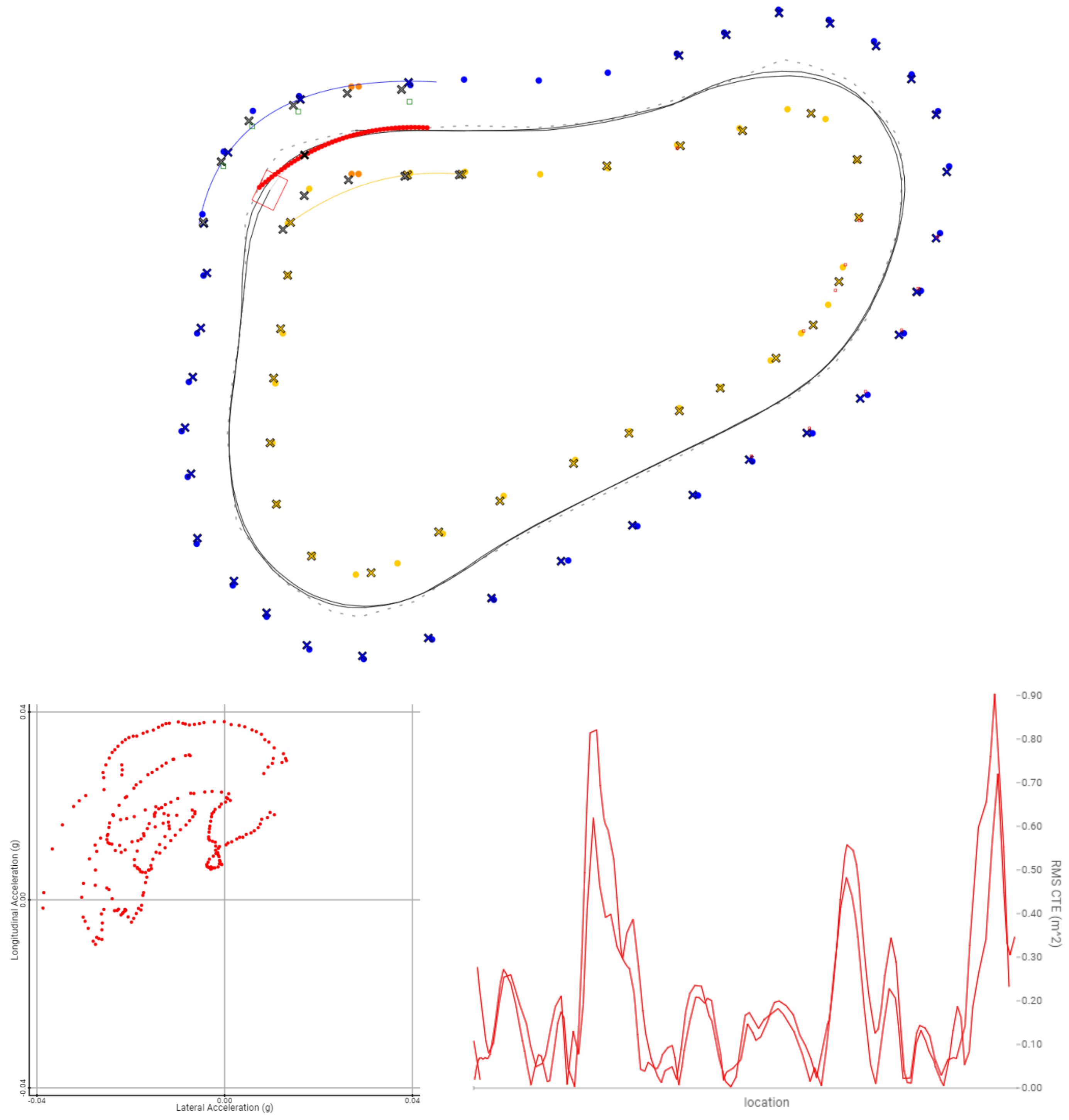

Figure 21: Pure-pursuit simulation results. Vehicle path relative to centreline (TOP), G-G diagram (LEFT) and RMS chart with respect to time (RIGHT)

iii. Simulation Results of Stanley Controller

The Stanley controller was tested using the front facing camera, LiDAR and wall building algorithms. This sensor combination was tested on a variety of maps, with 2 laps completed on each. Example data collected from the simulation of one of these test maps is displayed below.

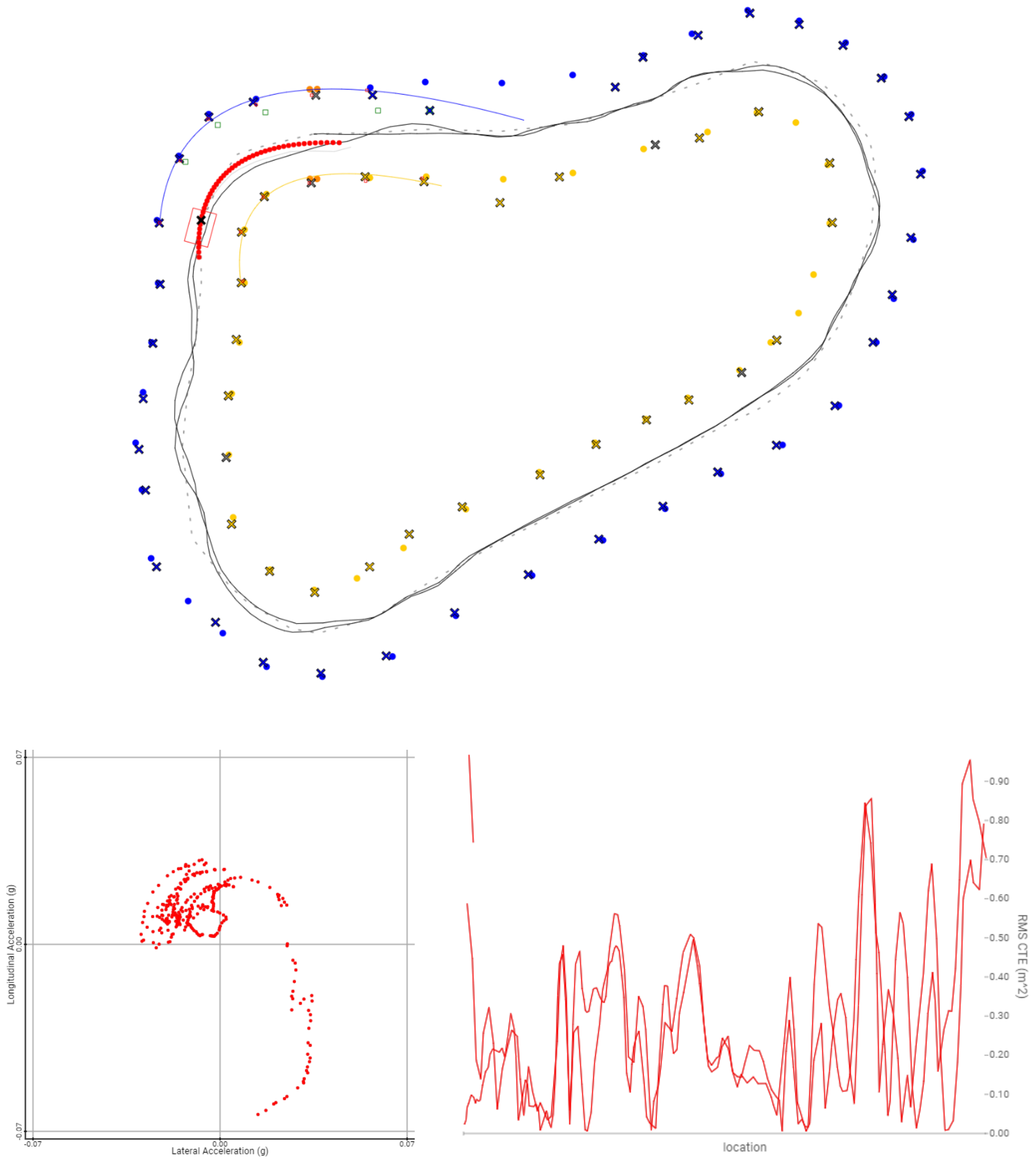

Figure 22: Stanley simulation results. Vehicle path relative to centreline (TOP), G-G diagram (LEFT) and RMS chart with respect to time (RIGHT)