# Software Architecture and Hardware-in-the-loop Simulation for an Autonomous Formula SAE Vehicle

Craig Brogle
21313578
The University of Western Australia
School of Electrical, Electronic and Computer Engineering
Supervisor: Professor Dr. Thomas Braunl

Submitted: 22nd October, 2018
Word Count: 6137

## 1 Abstract

Simulation is a cornerstone of autonomous driving efforts, allowing testing to occur more rapidly and with significantly less risk than is possible with hardware platforms alone. Simulation systems must be able to emulate a variety of sensors including cameras and LiDARs in order to allow high-level software such as image processing and path planning to be tested. In this paper, we present a hardware-in-the-loop simulation system based on CARLA, which incorporates compute hardware identical to that used on an autonomous vehicle platform in order to provide realistic constraints regarding available processing power along with access to the sensors required to test high level software. In addition, we explore the Robot Operating System (ROS) based software framework used on the FSAE vehicle. Specifically, we detail how this software framework satisfies the requirements of flexibility, extensibility, and resiliency presented by its use in an autonomous vehicle, along with how its fulfilment of these requirements was beneficial to the development of the hardware-in-the-loop simulation system.

# 2 Acknowledgements

I would like to thank the following people and groups for their assistance in this project: Prof. Thomas Bräunl for his constant guidance; the other members of the REV Autonomous team for their insight and camaraderie; and friends and family for their support and encouragement over the course of the year. I would also like to thank the sponsors of the REV Project including the Nvidia Corporation for providing the Jetson TX1 and GPU under its GPU Grant Program, along with Synergy and Xsens.

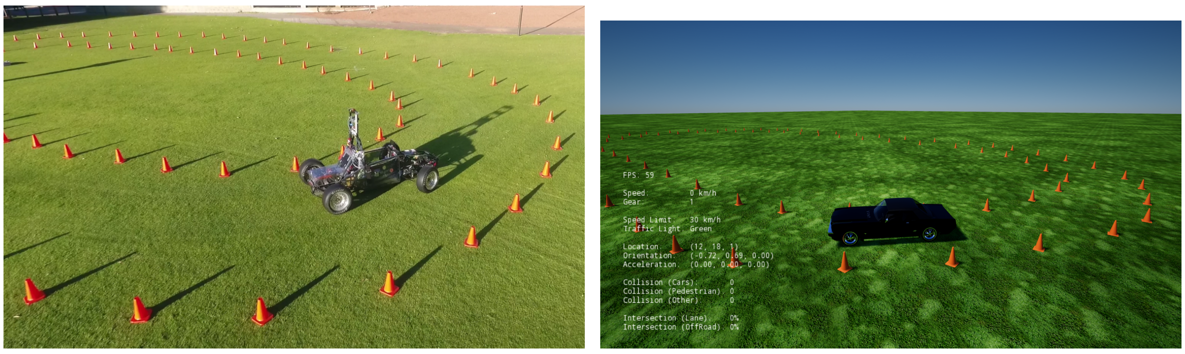
## Contents

## List of Figures

# 3 Introduction and Motivation

## 3.1 Background

The Renewable Energy Vehicle (REV) Project at UWA is currently focused on the development of autonomous driving applications.This development has occurred predominantly on a hardware platform consisting of a Formula SAE [1] race car converted to an electric drive [2] equipped with an IBEO LiDAR, a combined inertial measurement unit (IMU)/GPS from Xsens and a number of cameras for sensing, a Nvidia Jetson TX1 for compute, and full drive-by-wire capabilities [3]. At present, the goal for the driverless FSAE project increase the level of autonomy of the vehicle as it drives around a race track, from relying on waypoints placed manually through a Google Maps driven web interface [4], to relying solely on input from the variety of sensors available on the vehicle. This should result in the vehicle being capable of driving and mapping a semi-structured race track (with edges delineated by either cones, as displayed in Fig. 7, or road edges) with no prior knowledge before generating an optimised path and redriving the track at a greater speed.



```
FPS: 59

Speed:          0 km/h
Gear:           1

Speed Limit:    30 km/h
Traffic Light:  Green

Location:       (12, 18, 1)
Orientation:    (-0.72, 0.69, 0.00)
Acceleration:   (0.00, 0.00, 0.00)

Collision (Cars):        0
Collision (Pedestrian):  0
Collision (Other):       0

Intersection (Lane):     0%
Intersection (OffRoad):  0%
```

(a) (b)

Figure 1: *Scenarios used for comparing real (a) and simulated (b) LiDAR and visual cone processing outputs.*

The viability of the hardware-in-the-loop simulation system outlined in this paper, especially in terms of ease of integration and maintainability, was influenced greatly by the current software framework utilised on the FSAE vehicle. As such, this software framework will also be explored in this paper.

## 3.2 Motivation

The REV Project's Formula SAE vehicle platform is equipped with a variety of hardware safety systems and provides a number of advantages, such as being mechanically simple (resulting in low maintenance requirements and allowing for modifications to be made with ease) and being able to provide ample electrical power to the onboard sensors and hardware. However, given that is it a 250 kg vehicle capable of speeds up to 50 km/h, it is subject to sometimes onerous safety requirements. In addition, it is not a road licensed vehicle, preventing testing from occurring on public roads. These issues provide the motivation for the development of a hardware-in-the-loop simulation system, designed to allow more frequent testing in a wider range of environments with no risk while still presenting the same constraints on the available compute hardware. While the REV Project has made use of an autonomous driving simulator in the past [5], it was decided that the lack of support for LiDARs as a sensor, the outdated graphics, and the complexity involved in developing integrations with external systems provided sufficient reason to move to a more modern simulation platform.

Unfortunately, while effort has been made to improve the architecture of the software required by the SAE car [6], it was found that the resulting software remained highly coupled and lacked the extensibility and flexibility that would allow for rapid iteration in future software development efforts. In addition, the software architecture was not resilient to failure, with an error in any component capable of crashing the software, and was not designed to incorporate the requirements presented by an interface with an external driving simulator.

This paper outlines a new software architecture focused on the goals of resiliency, flexibility, extensibility and integration, which should allow future software development to iterate more quickly and with a higher degree of confidence in the base platform. It will then explore the development of the hardware-in-the-loop simulation system that was made viable by the new software architecture.

### 3.3 Overview

The remainder of this paper is organised as follows. Section 4 provides a summary of the current state of autonomous vehicles and driving simulations. Section 5.1 introduces the current software framework utilised on the FSAE vehicle. Section 5.2 introduces the open-source software used as the basis for the driving simulator, and details the integration with the software framework utilised on the FSAE vehicle. Sections 6 outlines the sensors available on the FSAE vehicle and the equivalent sensors available in the simulation software, along with the path planning algorithms currentlu available on the FSAE vehicle. Section 7 presents our experiments and results, followed by concluding remarks being drawn in Section 8 and suggestions for further work being presented in Section 9.

## 4 Literature Review

### 4.1 Background

An autonomous car is one that is capable of sensing and reacting to its environment without human intervention. SAE International defines six levels of driving automation based on the level of driver attentiveness and regularity of driver intervention required [7]. These range from systems that issue automated warnings and momentary interventions (Level 0) to systems that require no human intervention at any time (Level 5).

Based on the above definition, deployment of autonomous driving systems began in 1978 with serial production of the Anti-lock Braking System (ABS) [8], followed soon after by the development of Traction Control Systems (TCS). These driver assistance systems have progressed to include Level 1 systems such as Adaptive Cruise Control [9] and Parking Assistance [10] and Level 2 systems such as Tesla's Autopilot, Volvo's Pilot Assist, Cadillac's Super Cruise and Mercedes-Benz Drive Pilot [11].

Although not commercially available, progress is being made on higher level systems, with competitions such as the DARPA Grand Challenge [12] and DARPA Urban Challenge [13] promoting research into fully autonomous vehicles. Perhaps the most high-profile of these systems is Waymo's (formally the Google self-driving car project) Level 4 system having driven $\approx$ 8 million kilometres since 2009, with an additional $\approx$ 4.3 billion kilometres of simulated driving in 2017 alone [14].

These advances in autonomous vehicles have resulted in benefits in industry, demonstrated by projects such as Rio Tinto's fleet of autonomous haul trucks [15] and autonomous rail [16] being used to decrease cost while maximising safety.

## 4.2 Autonomous Driving Platforms

While the bulk of autonomous driving platforms are closed source, Baidu has released an open-source autonomous driving platform [17] with partners including car manufacturers such as Daimler, Ford and Honda, and technology companies such as Intel, Microsoft and Nvidia [18]. This platform is built on top of a version of ROS, an open-source project that "provides a structured communications layer above the host operating systems of a heterogenous compute cluster" [19]. In order to meet the demanding requirements of autonomous vehicles, modifications have been made to allow decentralisation, shared memory transport and Protobuf message support [20]. In addition, a reference suite containing localisation, perception, planning and control software has been made available as a part of the Apollo Auto project.

## 4.3 Autonomous Driving Simulation

Given the potential for uncertainty in the regulatory landscape to slow down progress into autonomous vehicles [21], companies such as Waymo [14], Cognata [22], rFpro [23] and Nvidia [24], as well as an open-source collaboration between Intel, Toyota and the Computer Vision Center [18], are developing autonomous driving simulators in order to allow autonomous vehicles to be trained and tested without regulatory hurdles. In addition, simulated environments allow for testing of autonomous vehicles in scenarios which occur extremely infrequently in real-life testing and which may pose a danger to those involved.

## 5 System Design

### 5.1 Software Framework

This section introduces the software framework currently utilised on the FSAE race car. Given the use of this software as a development platform in an autonomous vehicle, it was imperative that it be flexible and extensible, allowing for new components to be easily integrated, as well as resilient to software failures. This set of requirements led to a publish/subscribe software architecture being utilised, as it allows for highly decoupled software to be developed with a minimal set of shared dependencies, with each component (or series of components) needing only to conform to the expected input and output message types. By extension, this also allows for components to be developed in different languages depending on their importance and required performance characteristics, reducing the development time for simple, non critical components. The use of a publish/subscribe architecture also allows components to be swapped in and out, simplifying the testing of different solutions, and providing simple methods of logging, data capture and data replay that don't require modifying each component individually.

Based on the success of the Apollo Auto project [17], it was decided to use ROS [19] to provide the desired publish/subscribe functionality due to the resilience and performance that it displays. In addition, this provides a series of potential future upgrades, from transitioning to the Apollo platform [20] for improved performance due to shared memory transport for message passing, Protobuf message support and decentralisation to reduce single points of failure, to adopting the complete Apollo Auto platform should access to a supported hardware platform eventuate. The usage of ROS now ensures that any components developed will be compatible at any stage in this upgrade path, while also providing access to a large library of existing components and libraries, minimising the amount of supporting code and number of utilities that must be developed by the group to support common activities.

Based on the current goals of the REV Project, it was determined the software framework would initially require the nodes and message passing outlined in Fig. 2. Given the flexibility that the software framework provides this is an outline of the required functionality only, with the potential for some of the nodes displayed to be broken down into a series of smaller components. For instance, development of a variety of "Object Filter" nodes can be hastened by dividing it into two nodes as demonstrated in Fig. 3, one to simply combine the incoming object arrays into a single array ("Object Array Concatenation"), and another to perform the filtering ("Object Filter"). Development of nodes in this manner provides additional benefits, such as requiring only a single component to be modified should a new source of object data be introduced, such as a radar system.

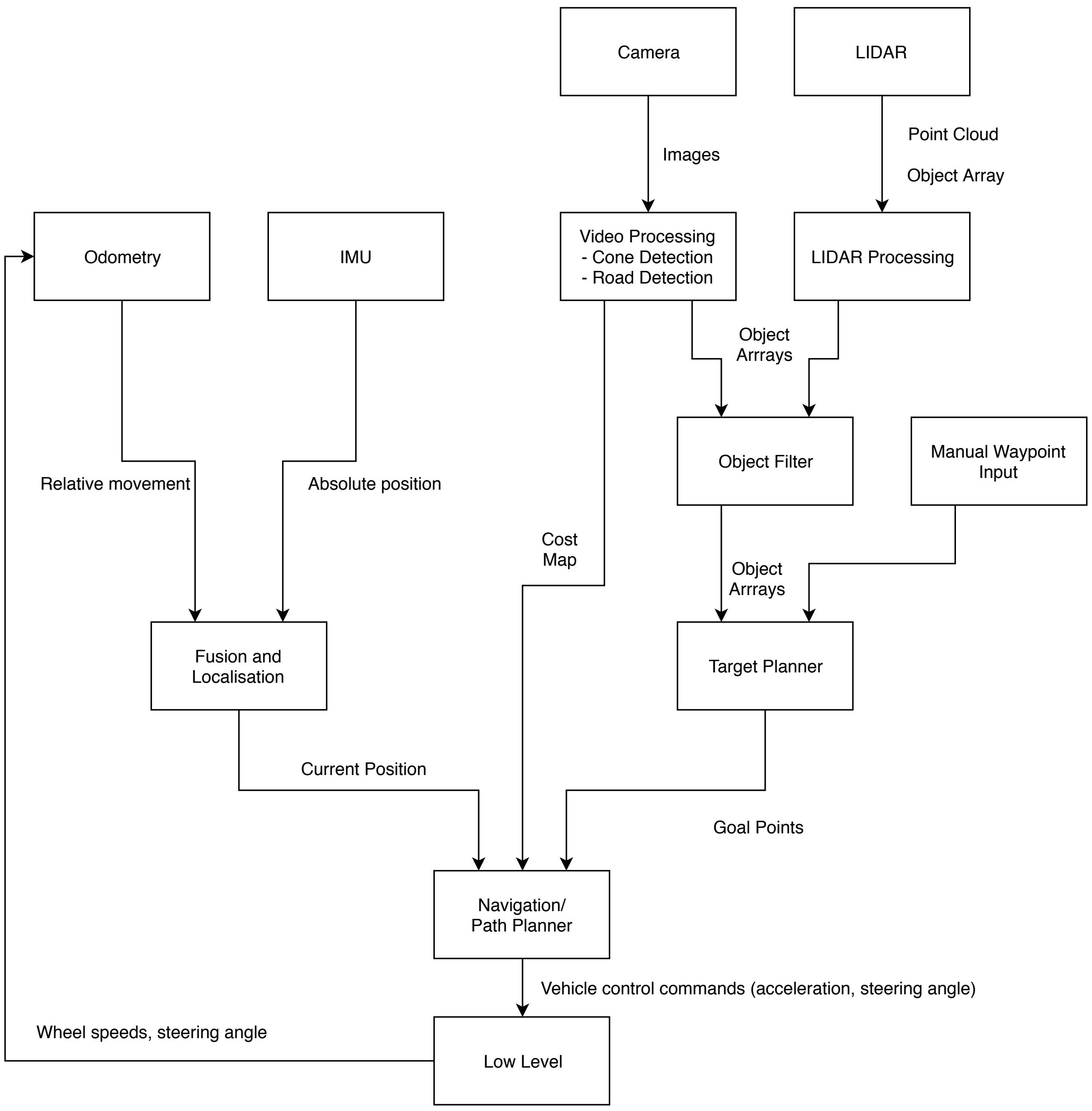


Figure 2: *FSAE vehicle ROS node structure.*

## 5.2 Autonomous Driving Simulation

As previously mentioned, the REV Project has made use of an autonomous driving simulator in the past [5], however it was decided that the lack of support for LiDARs as a sensor, the outdated graphics,

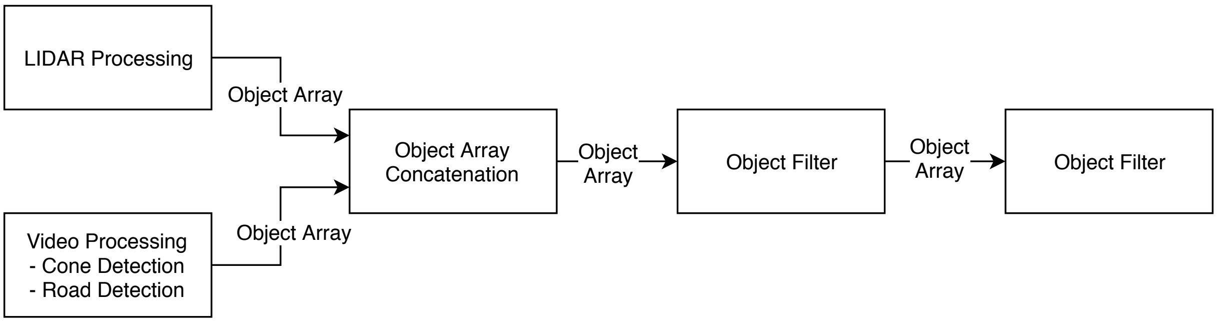


Figure 3: *FSAE vehicle ROS node structure.*

and the complexity involved in developing integrations with external systems provided sufficient reason to move to a more modern simulation platform. As such, the driving simulator developed is centered around the CARLA open source driving simulator [18] due to its providing the desired sensors and customisable scenarios without prohibitively expensive licensing or hardware requirements. By default, CARLA provides access to data from a configurable suite of sensors including cameras and LiDARs along with information regarding the current pose, velocity and acceleration of the simulated autonomous vehicle through a Python API. In addition, CARLA provides access to information regarding other simulated agents, allowing for the automated verification of results, and is developed in Unreal Engine [25], a popular gaming and simulation engine, which ensures that tools and resources are available for any future modifications to the system.

At present, the autonomous driving simulator consists of a computer (Intel i7-4770 processor, 8GB DDR3-1066 RAM, Nvidia Titan X (Maxwell) graphics processing unit (GPU)) running the aforementioned CARLA driving simulator and ROS simulation node which receives input from the Logitech G920 racing wheel over a USB connection, and performs bidirectional communication with an Nvidia Jetson TX1 over a network connection, as shown in Fig. 5. With this set up, we are able to run CARLA at upwards of 30 frames per second (FPS) without impacting the performance of the FSAE software running on the Nvidia Jetson TX1 for the simple environments testing occurs in. This setup is displayed in Fig. 4 along with the displays and racing seat used to provide a realistic driving environment.

The interface between the FSAE vehicle software framework and the CARLA driving simulator was greatly simplified due to the choice of ROS as a basis. ROS allows for inter-device communication over a network connection, and the software framework detailed in Section 5.1 allows for components or groups of components to be trivially swapped in and out. This resulted in the interface consisting of a single ROS node written in Python which retrieves sensor and environment data from the CARLA application programming interface (specifically, two camera feeds, a LiDAR point cloud and pose and velocity information of the vehicle), and published this information to the topics expected by the video processing and LiDAR processing nodes. The node then receives control data from the navigation/path planning node, which is used to create the control objects expected by CARLA for driving the simulated

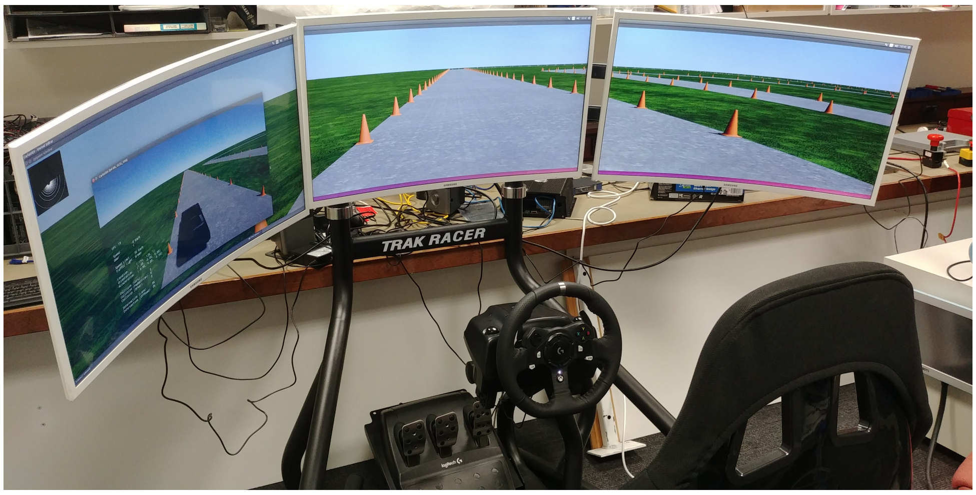


Figure 4: *Autonomous driving simulator setup.*

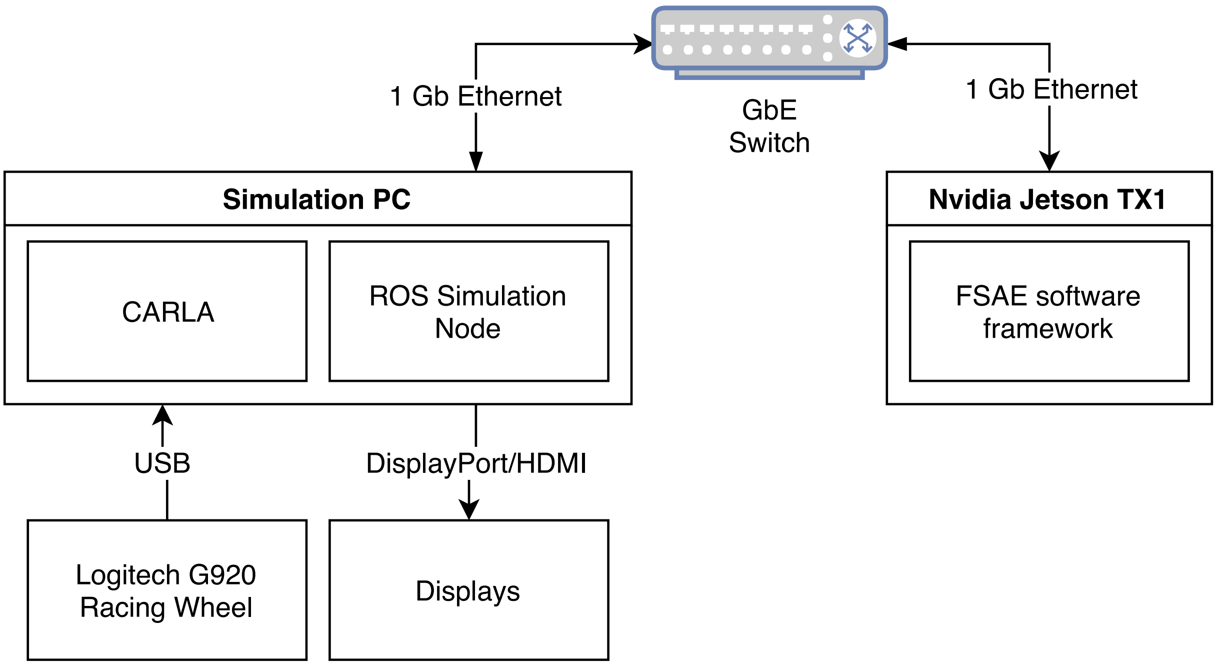


Figure 5: *Autonomous driving simulator hardware diagram.*

vehicle. This node acts in the place of the fusion and localisation, camera, LiDAR and low level nodes presented in Fig. 2, resulting in the application architecture seen in Fig. 6.

The inter-device communication enabled by ROS is critical in allowing realistic compute hardware to be used in the autonomous driving simulator. This allows the simulation software and results validation to be performed on a secondary device while the FSAE software is run on a Nvidia Jetson TX1, identical to the system installed on the FSAE vehicle. This allows for a high quality simulation to be run without

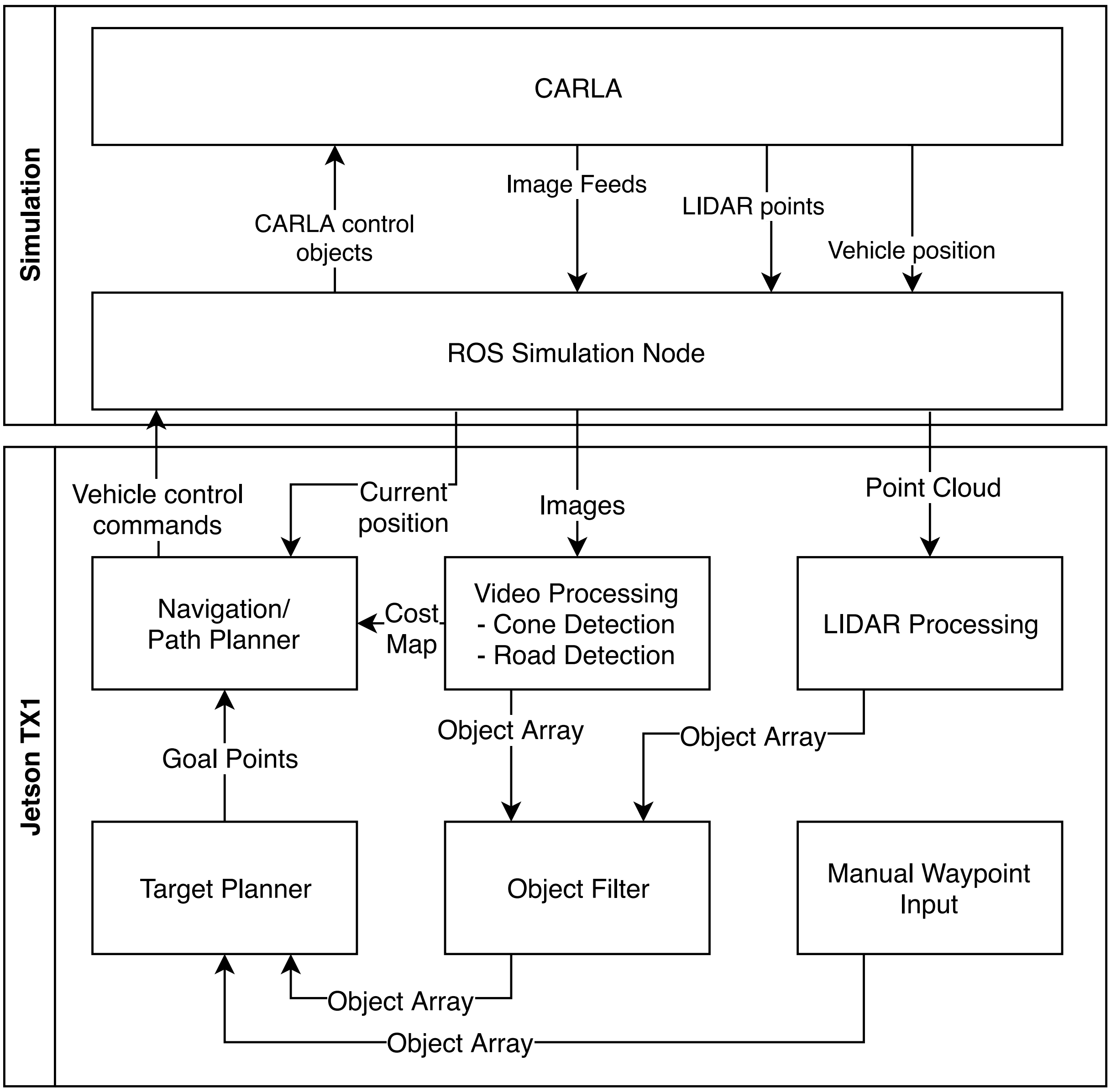


Figure 6: *Software framework architecture with CARLA simulator interface.*

negatively impacting the high level software performance, while still presenting similar performance constraints to the high level software. This ensures that software tested successfully on the simulated system will be able to perform almost identically on the real FSAE vehicle, which is verified in Section 7.2.3.

In replacing the low level node, the simulator interface also assumed responsibility for emulating a number of the safety systems provided by the low level controller, such as allowing for manual intervention to override the autonomous systems. This was achieved by handling manual input through either a keyboard or Logitech G920 racing wheel [26], and replicating the low level system's response to these inputs in the simulator interface. It is also possible to connect a small touch-screen display to the Nvidia Jetson TX1 to provide an interface to the autonomous systems identical to that found on the FSAE vehicle, allowing for user interface testing to occur in a safe environment.

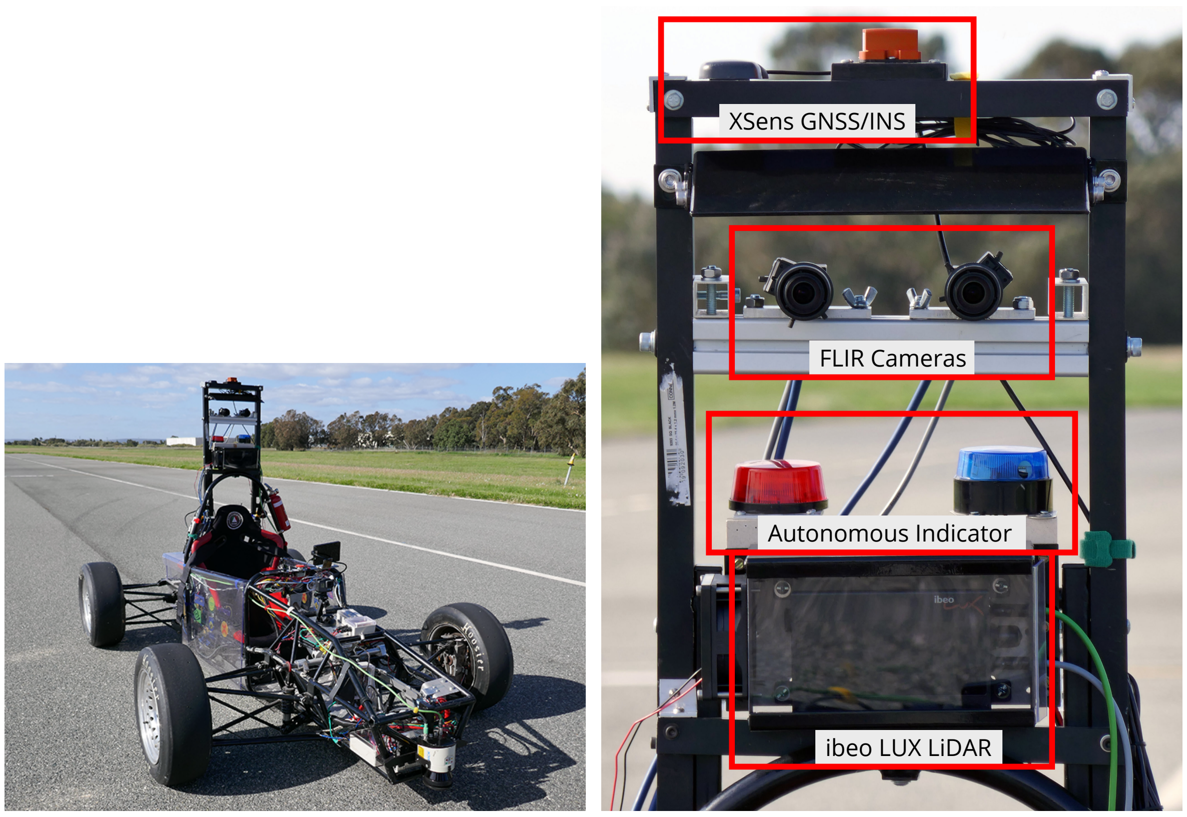
The open-source nature of the CARLA open source simulator provides additional benefits, such as an active community to provide troubleshooting and contribute features and performance and stability improvements to the project. To date, the primary benefit has been in the ability to create custom scenarios and import custom object meshes, however there is scope for actions such as creating new types of sensors should the need arise.

## 6 Navigation and Path Planning

The FSAE car uses a combination of sensors (displayed in Figs. 7a and 7b) including LiDARs, cameras, wheel odometry and an IMU, which are divided into four categories: a camera system, dead reckoning, a LiDAR system and odometry. The driving simulator provides replacements for the LiDAR and camera systems, and makes the dead reckoning and odometry systems obsolete by providing exact positioning in the simulated environment, allowing focus to be placed on camera and LiDAR systems with the guarantee that the data regarding the cars positioning will be correct. This section will demonstrate the sensors that are made available by CARLA, and draw comparisons to the sensors currently installed on the FSAE vehicle. It will also introduce the path planning algorithm which has been used as for testing the simulator.

### 6.1 LiDAR System

The cone detection algorithm currently utilised on the FSAE vehicle relies on a single front-mounted 2D SICK LiDAR [27], providing a 270° horizontal field-of-view with an angular resolution of 0.25–0.5° and a range of up to 20 m. CARLA provides a 360° LiDAR with a configurable range, number of channels, rotation frequency, and upper and lower field-of-view limits [28]. In order to emulate the `LaserScan` [29] data that is available through SICK's LMS1xx ROS driver [30], the `pointcloud_to_laserscan` ROS package [31] was used in conjunction with a CARLA LiDAR configuration with a narrow vertical field of view. This combination allows a `LaserScan` to be produced with a configurable horizontal field-of-view and angular resolution, hence matching the output produced by the physical LiDAR.



(a) (b)

Figure 7: *Formula SAE software development platform (a) and sensor rack (b)*

### 6.2 Camera System

The FSAE vehicle is currently equipped with two FLIR Blackfly GigE [32] cameras. Each of these cameras has a maximum resolution of 1288 × 964 pixels and is capable of working at 30 FPS. The cameras make use of a global shutter, removing the need for the compute hardware to perform compensation for rolling shutter effects [33], such as those presented in [34]. CARLA provides a direct analogue for these cameras in the form of the 'scene final' cameras [28]. These are also global shutter cameras, and CARLA provides facilities to configure the field-of-view, resolution and position of the cameras. Using this feature, the simulation is configured to output two image streams at a resolution of 1288 × 964 pixels each, placed the same distance apart to mirror the FLIR camera setup displayed in Fig. 7b. The frame rate of the cameras provided by CARLA is tied to the FPS that the CARLA simulator is run at. While it is possible to set a static FPS target in CARLA, this prevents the simulator from running in real time which is incompatible with the FSAE software framework. Instead, images are published to the software framework at the same FPS that CARLA runs at, with only the latest received image being stored. The visual cone detection node then retrieves this image as needed at a frequency of between

15 Hz and 30 Hz.

## 6.3 Path Planning

The FSAE control system has been implemented to deliver path planning routines to allow either driving through a series of predefined waypoints, or in between a series of traffic cones placed on either side of the vehicle. This paper will focus solely on the cone driving scenario, as this present a higher level of complexity and required processing power in order to thoroughly test the simulation system.

The current iteration of the path planning procedure uses obstacle detection of the cones to determine the correct path. The current code uses the same as [35], but simplifies it to allow for quicker calculation. Our cone driving module accepts cone locations from either the map, LiDAR or camera, classifying them as objects. Then, the vehicle navigates to drive within the track formed by cones safely without collision. Using a range of the maximum turning circle of the car, of both a left-hand turn and right-hand turn, it then looks at which predicted paths will intercept cones. The vehicle dynamics is thus limited during motion planning whereby the steering angle does not exceed 25° . Our algorithm will iterate through all cones within the car's range and calculate the best collision-free path to undertake, as detailed in Algorithm 1.

**Algorithm 1** Cone driving

```
 1: procedure CONEDRIVE(cones in range)
 2:     init steering_range to [-25,25]
 3:     for all cones in range do
 4:         evaluate collision_range with cone
 5:         exclude the collision_range from steering_range
 6:     end for
 7:     if steering_range is empty then
 8:         stop
 9:     else if all steering_range $\leq$ threshold then
10:         select largest steering_range
11:     else if all steering_range $>$ threshold then
12:         select steering_angle with minimum change in        current direction
13:     end if
14:     drive toward centre of steering_range
15: end procedure
```

# 7 Experiments and Results

## 7.1 Software Framework

Verification of the suitability of ROS as the basis of the new software framework was twofold: first proof was required that ROS was capable of meeting the desired performance requirements, and secondly there must be sufficient evidence that ROS is capable of providing the level of resiliency and fault tolerance required for safety critical applications. These verifications are detailed in Section 7.1.1 and Section 7.1.2.

### 7.1.1 Performance

Basic calculations and tests were performed in order to verify that ROS was capable of transferring data at the rate that would be required by the FSAE software. Calculating the required data transfer rate consisted of counting the number of topics required by the outline of the software framework that was established in Fig. 2, and multiplying this by the desired control update frequency. Based on the presence of 11 topics and a desired update frequency of 15Hz, we can calculate that ROS will be required to pass at least 165 messages per second as seen in Equation 1.

$$\begin{aligned}\text{Message Rate} &= \text{No. Topics} \times \text{Update Frequency} \\ &= 11 \times 15 \\ &= 165 \text{ Messages per second}\end{aligned} \tag{1}$$

In order to verify that ROS was capable of handling at least this rate of message passing, a simple application was developed. This consisted of the creation of a pair of ROS nodes, one node to publish messages and one node to subscribe to them, with both nodes set to a frequency of 400 Hz, chosen to provide additional headroom should additional nodes be introduced to the FSAE software. Both nodes then recorded the time difference between each message that it sent or received, with success being defined as having an average time between sending or receiving each message that was, within a small margin of error, less than or equal to $\frac{1}{f} = \frac{1}{400}$ seconds. Based on a sampling of 10000 points from each the publisher and subscriber, it was found that the publisher node was capable of publishing simple messages at an average rate of $\approx 178$kHz, and the subscriber capable of receiving messages and performing basic processing at an average rate of $\approx 8$kHz. In both cases, ROS is able to massively surpass the current performance requirements of the FSAE vehicle and any modifications that are likely to occur in the near future. However, should a higher level of performance be required, the Apollo platform is likely able to increase this performance through its introduction of shared memory transport, which would be especially beneficial when utilising large message types such as images.

### 7.1.2 Resiliency

There are a number of features of ROS that enable it to satisfy the requirements for resiliency posed by the software's use in an autonomous driving platform. The first and most important of these is the use of separate Linux processes for each node. By running each node on a different process, it is ensured that the failure of one node cannot result in the entire system crashing, as was observed with the previous monolithic C++ software architecture. This ensures that any bugs introduced to the system during development are contained within a single node. In this way, it is possible for the vehicle to continue to operate, albeit with reduced capabilities, in the event of a software crash.

This capability to continue operating through the majority of software crashes is complemented by the variety of methods that are available to monitor and restart ROS nodes, as well as methods of notifying other nodes of crashes, which are then able to modify their behaviour to take into account the node being unavailable for a brief time. The simplest of these is the roslaunch `respawn` parameter [36]. Setting this to `true` in the roslaunch file of each node ensures that any node that dies will automatically be respawned. In addition to respawning nodes, ROS packages are available to easily define bonds between nodes [37][38], which allow nodes to react immediately to the failures of other nodes. While no work has been done yet in implementing this feature on the FSAE vehicle due to the rate at which changes have continued to be made to the set of nodes required for driving and the use of controlled test environments, this provides the ability for graceful error handling to be implemented before testing occurs in less controlled environments, such as those involving other vehicles or pedestrians. In these instances, it would be possible to perform actions such as smoothly passing control back to a human operator, or automatically failing over from one sensor type to another, in the event of a node crashing.

While it should be noted that the ROS master node is currently a single point of failure in this system, the work completed by the Apollo platform [39] team includes the decentralisation of ROS, which removes this single point of failure by allowing other existing nodes to continue to comminicate even if the master node crashes, resulting in only a minor disruption to the system's operation. While the FSAE vehicle does not currently make use of the Apollo platform due to the basic ROS platform satisfying current testing requirements, the Apollo platform has been identified as an easily implemented upgrade path, which could easily replace the usage of ROS should the need for increased reliability be required, such as by the start of testing in scenarios involving other vehicles or pedestrians.

## 7.2 Autonomous Driving Simulation

Verification of our simulation system came in the form of experiments related to LiDAR and vision based cone detection, processor load monitoring and system response times, detailed in Sections 7.2.1 through 7.2.4.

### 7.2.1 LiDAR Cone Detection

The aim of this experiment was to verify that the LiDAR output obtained from the simulation using the method described in Section 6.1 was sufficiently similar to that generated by the SICK LiDAR available on the FSAE vehicle in order to allow cone detection algorithms to be tested on the driving simulator. This was achieved by simulating a scenario mimicing that of a previous test of the FSAE vehicle, and verifying that a similar set of cones was detected by the same algorithm used in the FSAE vehicle test. Using the real and simulated scenarios presented in Fig. 8, the outputs of the cone detection performed on each scenario, displayed as red cylinders in Fig. 9, are sufficiently similar to allow testing of higher level components such as path planning and object avoidance on the simulated system.

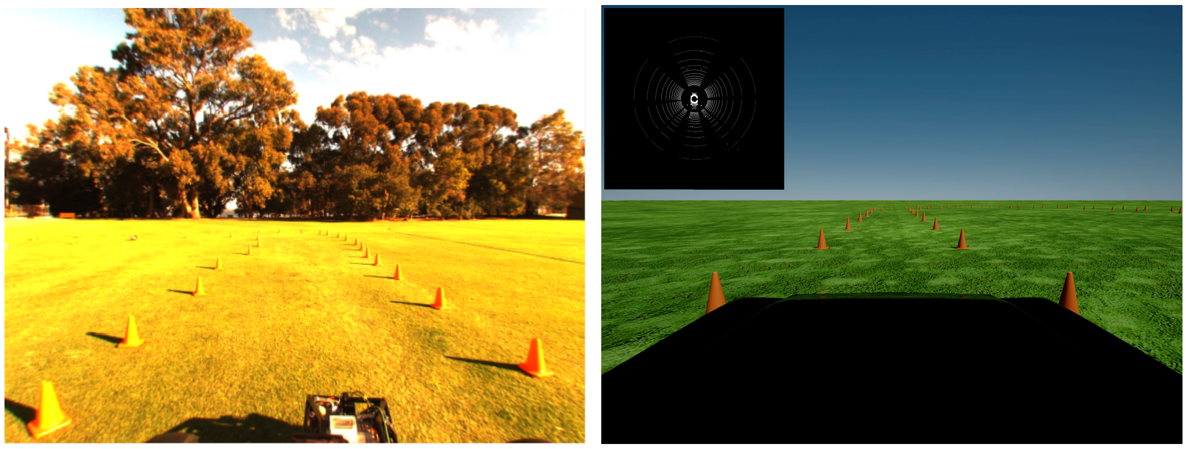

(a) (b)

Figure 8: *Scenarios used for comparing real (a) and simulated (b) LiDAR and visual cone processing outputs.*

### 7.2.2 Visual Cone Detection

This experiment aims to verify that the images available through CARLA's camera sensors (described in Section 6.2) are sufficiently similar to those generated by the FLIR Blackfly GigE cameras installed on the FSAE vehicle that a visual cone detection algorithm is capable of producing similar results on both images. A comparison of these results is given in Fig. 10. From this, it can be seen that cones are identified successfully, however with a decreased range on the simulated image. It is expected that the detection range on simulated images could be increased by incorporating some simulated images into the training set.

In order to further verify that the images produced by CARLA's camera sensors are sufficiently similar to those generated by the FLIR Blackfly GigE cameras installed on the FSAE vehicle, a comparison between the false positive (a cone is detected where no cone exists) and false negative (no cone is detected

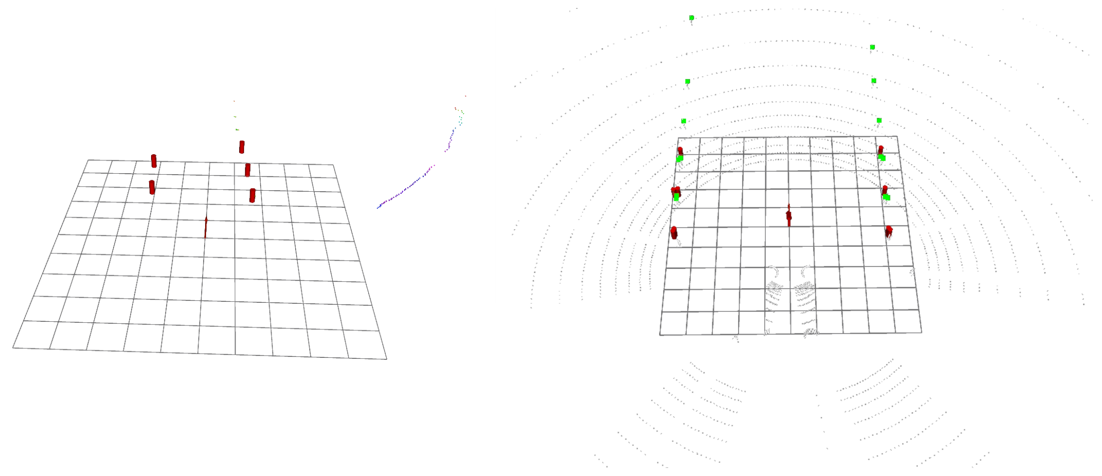

(a) (b)

Figure 9: *LiDAR based cone detection results from real (a) and simulated (b) scenarios.*

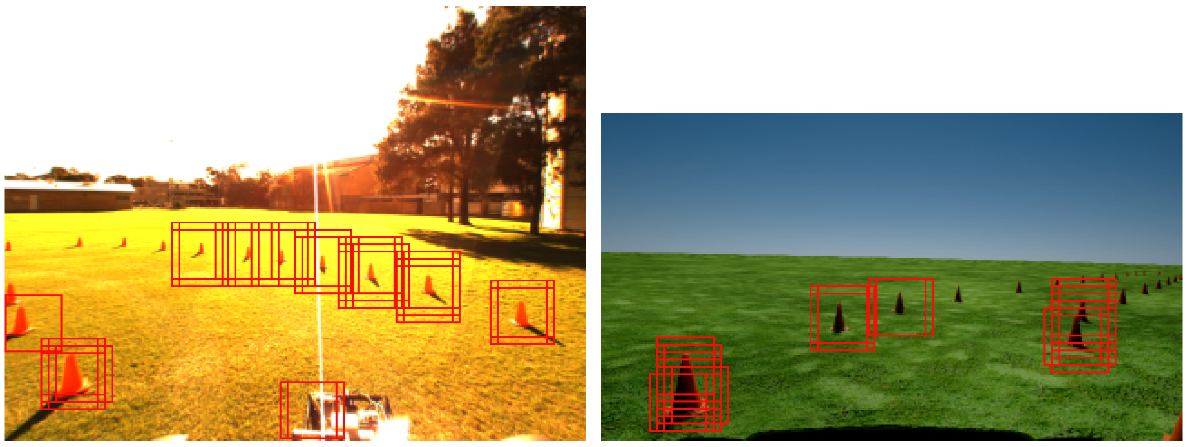
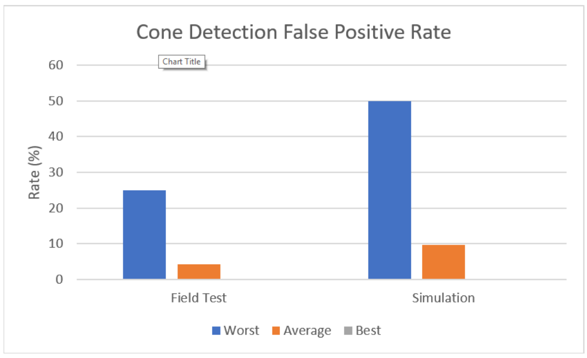

(a) (b)

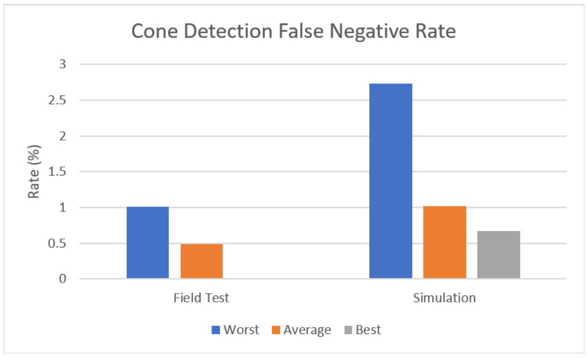
Figure 10: *Computer vision based cone detection results from real (a) and simulated (b) scenarios.*

where a cone exists) cone detection rates on real and simulated footage was undertaken, the results of which are available in Fig. 11. It can be seen in both the false positive and false negative scenarios that a trend similar to that found through visual inspection can be identified. This trend has the visual cone detection algorithm performing consistently worse in simulated scenarios, and is especially pronounced in the worst case false negative detection rate, with the cone detection algorithm having a $\approx 50\%$ false negative detection rate for simulated data in the worst case, compared to only $\approx 25\%$ for real data. As mentioned previously, it is expected that this large discrepancy could be significantly reduced by incorporating some simulated images into the training set of the cone detection agorithm. It should be noted that there is a much less pronounced difference in the average cases for both false positive and false negative detections, suggesting that in the majority of cases the outcomes from real and simulated data will be similar.

Given that the current path planner works on only a single frame, it was found that the increase in false positive and negative detections in the simulated environments did not have any impact on the success of the vehicle's traversal of the cone track. It is expected that this will continue to be a non-issue in nodes that are still to be developed, such as a global object map, as the nature of these nodes requires that they be capable of dealing with transitory incorrect classifications and detections, given that these occur when using real data as input. As such, the increase in false positive and negative detections in simulated data can be presented in a positive light, as a measure that forces all new algorithms to meet higher standards in terms of their resiliency to incorrect classifications made by nodes earlier in the framework's data flow, resulting in software that is more resilient to conditions adverse to accurate sensing and erroneous processing.



(a)



(b)

Figure 11: *Computer vision based cone detection statistics from real and simulated scenarios.*

### 7.2.3 Compute Hardware Load

This experiment was designed to verify that the use of identical compute hardware (a Nvidia Jetson TX1) in the simulation loop resulted in similar performance constraints to those presented by the FSAE vehicle platform. This verification comes in the form of a comparison of the system resources used in the real and simulated systems while performing a similar task, in this instance, LiDAR based cone detection. Results were gathered by running the `sysstat` performance monitoring tools for Linux [40] while the cone detection algorithms were operating on both systems. This utility captured the percentage of the processor utilised by user and system processes at a frequency of 1 Hz for 120 seconds to allow an average to be computed, the results of which are displayed in Fig. 12b and Fig. 12a. From this figure, it can be seen that the hardware in the simulation loop had consistently higher processor utilisation for user space processes, with an average of $\approx 25.2\%$ (Fig. 12b) compared to $\approx 20.6\%$ (Fig. 12a) for user space processes on the real system. Given that in both scenarios more than

60% of processor time is spent at idle, it is unlikely that this difference in processor utilisation by user space processes would significantly impact application performance, however additional performance profiling such as measuring individual method call times could be implemented should significant performance differences be detected which impact on the transition from the simulated to real test environments.

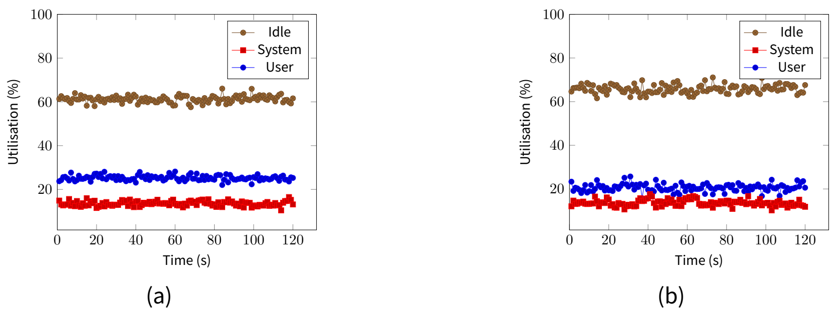


Figure 12: *Processor utilisation during LiDAR based cone processing based on simulated (a) and real (b) input.*

### 7.2.4 Response Time

Given that the simulation involves transferring streams of images and other sensor data across a network connection, verification is required to ensure that this does not introduce significant delays to the response time of the system. This was achieved by sending a control command from the simulation computer, measuring the time taken for the system to respond, and comparing this to the same measure taken on the FSAE vehicle. The results of this experiment are presented in Fig. 13, where it can be seen that the simulator had a significantly lower average response time. Given current testing occurs predominantly at low speeds this is seen to be insignificant for current use cases, although an artificial delay could be trivially added to the simulation system to better mimic the FSAE vehicle. Alternatively, there is potential for a delay specific to each simulated vehicle to be introduced through the process of adding new vehicle models to CARLA discussed Section 9.

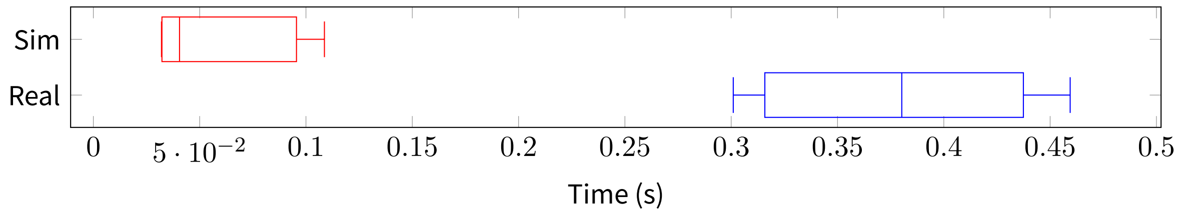


Figure 13: *Vehicle response times for real and simulated vehicles.*

## 8 Conclusion

We have presented a hardware-in-the-loop autonomous driving simulation system that is capable of simulating the various sensors available on the FSAE vehicle with sufficient detail that existing LiDAR and computer vision based algorithms perform similarly when compared to real data. We have detailed the architecture of the software framework utilised on the FSAE vehicle, and how the use of this framework was able to simplify the development of the hardware-in-the-loop simulation due to its high degree of flexibility and extensibility. We have also verified that the software framework utilised on the FSAE vehicle provides suitables performance characteristics, and explored methods by which the resiliency of the framework can be improved to ensure suitability for usage in safety critical scenarios. Our design approach of utilising actively developed, open-source projects on commodity hardware results in a relatively low-cost simulation solution that is nonetheless capable of generating sensor data at a frame rate greater than that required for the FSAE vehicle's software framework. We have shown that this system allows for software to be tested in an environment that presents similar performance constraints as the FSAE vehicle platform. Most importantly, we have verified that results generated through use of this simulation system are transferrable to the FSAE vehicle for the group's current use cases.

## 9 Future Work

Over the previous year, various members of the REV Project have worked with the new software framework to develop modules relating to visual and LiDAR based cone detection, along with a local path planning module which allows the vehicle to navigate a track with edges delineated by a series of cones. In addition, a number of sample implementations for road edge and lane detection have been generated, however are yet to be integrated with the vehicle's path planning capabilities. The integration of these independent modules should be the first priority in future works, as this will complete the current base goals of autonomously driving a track with edges delineated by either cones or road edges without prior knowledge.

In order to be capable of redriving a track at greater speed, a global path planner will be required. The most basic requirement of this is highly accurate localisation of the vehicle. Although a highly-accurate IMU/GPS device is currently available on the vehicle, it has not been integrated with the new software framework, and so is currently providing little benefit. To resolve this, a vehicle localisation node should be prioritised which is capable of providing accurate positioning, at least relative to a starting position. This will allow for a global map to be created, meaning that the vehicle is able to maintain knowledge of objects that it detects after they move out of sensor range. This in turn will allow for the development of a global path planner, that is capable of generating an optimised path around a complete track, as opposed to having the vehicle reacting only to what the sensors can detect at any single moment.

In order to support these and other high-level software modules, it is suggested that the driving simulation system be extended in a number of ways. Firstly, a vehicle model more closely resembling the FSAE vehicle available to the REV group both visually and in terms of vehicle mechanics (such as suspension, acceleration and braking characteristics) should be incorporated into the simulation. This would provide additional realism to the data generated, removing the need, for example, to account for the excessive rotation that the default simulated vehicle encounters when accelerating, braking or cornering, which results in the LiDAR detecting the ground much closer to the vehicle than occurs in operation of the FSAE vehicle. Secondly, some form of error should be introduced to the positioning data provided by the simulation system to the FSAE vehicle software. This would allow for a variety of visual and LiDAR based odometry techniques to be tested on the simulation system, with the added benefit that the exact positioning data could still be made available for evaluating the effectiveness of any solutions tested. Thirdly, a wider variety of test tracks should be developed, emulating both basic race track type scenarios, along with more realistic road driving scenarios which may include other vehicles and pedestrians. This will require modifications to be made to the base simulation software (the CARLA driving simulator) to allow testing in Australian conditions, as currently all automated traffic movement and scenario benchmarking is available only for maps where vehicles drive on the right-hand side of the road. Fourthly, a task queue system could be implemented, allowing users of the simulation to choose a sequence of scenarios to test software on, and provide a summary upon completion, allowing unsupervised testing to occur. This would rely heavily on the previous suggestion for the availability of appropriate metrics. Finally, a simple user interface should be developed for the simulation system, allowing the user to configure the simulated sensor suite, along with the scenario to be entered and the version of the FSAE vehicle software to be tested. This would significantly increase the accessibility of the simulation system, reducing the need for future developers to be intimately familiar with the internals of the simulation system in order to make basic changes. In the same vein, consideration could be given to making the simulation system available over a remote connection, allowing easier access to the system by future students. As an extension, a version of the simulation system could be made available on a cloud platform. While this would be unable to provide the same hardware-in-the-loop simulation due the inability to access a Nvidia Jetson TX1 in a cloud environment, this could allow for multiple simulation environments to be run concurrently, allowing for either faster testing of multiple scenarios by a single user, or simultaneous access by multiple users.

In addition to the above suggestions, which center around the current goals of autonomously driving a track with no other vehicles or obstructions present, there is potential for significant future works to occur with an aim to broaden the number of scenarios which the FSAE vehicle is capable of handling autonomously. This could range from attempts to implement advanced driver assistance features such as adaptive cruise control and lane keeping, to full vehicle autonomy in simple traffic scenarios, however testing of these would be significantly hindered by the lack of a road licensed vehicle.

## References

[1] S. International, “Student events.” [Online]. Available: https://www.sae.org/attend/student-events/

[2] “Self driving formula sae race car.” [Online]. Available: http://therevproject.com/vehicles/sae2010.php

[3] T. Drage, J. Kalinowski, and T. Braunl, “Conversion of a formula sae vehicle to full drive-by-wire capability,” UWA, 2013.

[4] T. Drage, “Development of a navigation control system for an autonomous formula sae-electric race car,” UWA, 2013.

[5] S. Bradley, “Automotive simulation system,” Master’s thesis, University of Western Australia, 2009.

[6] S. Evans-Thomson, “Environmental mapping and software architecture of an autonomous sae electric race car,” UWA, 2017.

[7] S. O.-R. A. V. S. Committee and others, “Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems,” *SAE International*, 2014.

[8] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner, “Three decades of driver assistance systems: Review and future perspectives,” *IEEE Intelligent Transportation Systems Magazine*, vol. 6, no. 4, pp. 6–22, 2014.

[9] P. I. Labuhn and W. J. Chundrlik Jr, “Adaptive cruise control.” Google Patents, oct~3-1995.

[10] J. L. Czekaj, “Semi-autonomous parking control system for a vehicle providing tactile feedback to a vehicle operator.” Google Patents, apr~21-1998.

[11] J. Hughes, “Car autonomy levels explained.” [Online]. Available: http://www.thedrive.com/sheetmetal/15724/what-are-these-levels-of-autonomy-anyway

[12] S. Russell, “DARPA grand challenge winner: Stanley the robot!” [Online]. Available: https://www.popularmechanics.com/technology/robots/a393/2169012/

[13] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, “Little ben: The ben franklin racing team’s entry in the 2007 darpa urban challenge,” *Journal of Field Robotics*, vol. 25, no. 9, pp. 598–614, 2008.

[14] “Waymo.” [Online]. Available: https://waymo.com/

[15] Available: http://www.riotinto.com/media/media-releases-237_23802.aspx

[16] “Rio tinto completes first fully autonomous rail journey in western australia.” [Online]. Available: http://www.riotinto.com/media/media-releases-237_23264.aspx

[17] “Apollo auto.” [Online]. Available: http://apollo.auto/

[18] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st annual conference on robot learning*, 2017, pp. 1–16.

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: An open-source robot operating system,” in *ICRA workshop on open source software*, 2009, vol. 3, p. 5.

[20] “Apollo-platform.” [Online]. Available: https://github.com/ApolloAuto/apollo-platform

[21] J. S. Brodsky, “Autonomous vehicle regulation: How an uncertain legal landscape may hit the brakes on self-driving cars,” *Berkeley Tech. LJ*, vol. 31, p. 851, 2016.

[22] “Cognata - deep learning autonomous simulation.” [Online]. Available: http://www.cognata.com/

[23] “RFpro.” [Online]. Available: http://www.rfpro.com/

[24] “Nvidia drive constellation.” [Online]. Available: https://www.nvidia.com/en-us/self-driving-cars/drive-constellation/

[25] Epic Games, “Unreal engine.” [Online]. Available: https://www.unrealengine.com/en-US/what-is-unreal-engine-4

[26] “G920 driving force racing wheel for xbox one and pc.” [Online]. Available: https://www.logitechg.com/en-au/product/g920-driving-force

[27] SICK AG, “LMS111-10100.” [Online]. Available: https://www.sick.com/au/en/detection-and-ranging-solutions/2d-lidar-sensors/lms1xx/lms111-10100/p/p109842

[28] “CARLA simulator - cameras and sensors.” [Online]. Available: https://carla.readthedocs.io/en/latest/cameras_and_sensors/. [Accessed: 12-Sep-2018]

[29] “Sensor_msgs/laserscan message.” [Online]. Available: http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

[30] Open Source Robotics Foundation, “LMS1xx.” [Online]. Available: http://wiki.ros.org/LMS1xx

[31] Open Source Robotics Foundation, “Pointcloud_to_laserscan.” [Online]. Available: http://wiki.ros.org/pointcloud_to_laserscan

[32] FLIR Integrated Imaging Solutions, “Blackfly 1.3 mp color gige poe (sony icx445).” [Online]. Available: https://www.ptgrey.com/blackfly-13-mp-color-gige-vision-poe-sony-icx445-camera

[33] S. Lauxtermann, A. Lee, J. Stevens, and A. Joshi, “Comparison of global shutter pixels for cmos image sensors,” in *2007 international image sensor workshop*, 2007, p. 8.

[34] C. Liang, L. Chang, and H. H. Chen, “Analysis and compensation of rolling shutter effect,” *IEEE Transactions on Image Processing*, vol. 17, no. 8, pp. 1323–1330, Aug. 2008.

[35] K. L. Lim, T. Drage, R. Podolski, G. Meyer-Lee, S. Evans-Thompson, J. Y.-T. Lin, G. Channon, M. Poole, and T. Braunl, “A Modular Software Framework for Autonomous Vehicles,” in *2018 29th IEEE Intelligent*

*Vehicles Symposium*, 2018, pp. 1780–1785.

[36] Open Source Robotics Foundation, “Roslaunch/xml/node.” [Online]. Available: http://wiki.ros.org/roslaunch/XML/node

[37] Open Source Robotics Foundation, “Bondpy.” [Online]. Available: http://wiki.ros.org/bondpy

[38] Open Source Robotics Foundation, “Bondcpp.” [Online]. Available: http://wiki.ros.org/bondcpp

[39] “Apollo-platform.” GitHub Repository [Online]. Available: https://github.com/ApolloAuto/apollo-platform

[40] S. Godard, “SYSSTAT.” [Online]. Available: http://sebastien.godard.pagesperso-orange.fr/