Transferable Semi-autonomous Architecture for Vehicles Operating in Corridor Crop Environments

at

The University of Waikato

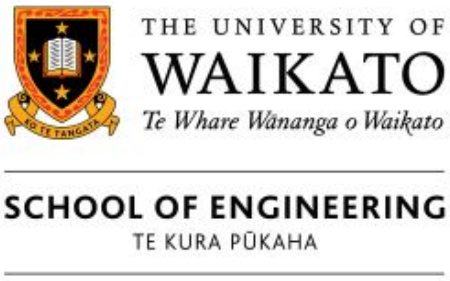
by

Jonathan Isaac Tobias

Supervised by:

Dr. Shen Hin Lim

Prof. Mike Duke



March 10, 2022

Declaration

In submitting this thesis to The University of Waikato School of Engineering, I affirm my awareness of the universities standards and honour code.

Name: Jonathan Isaac Tobias

Signature: Jonathan Isaac Tobias

Abstract

The role of autonomous vehicles (AVs) in assisting people is recognised and, therefore, is in constant development in numerous fields. Specifically, AV's ability to alleviate global stressors, including an increased potential for food shortages and the decline in workers for potentially laborious work. An area where AVs developments are particularly prevalent is in agriculture. However, the few AVs being used in agriculture are often custom built for specific purposes and require large development time as a result. This thesis aims to build and evaluate a versatile architecture that can be transferred to any agricultural vehicle, thus decreasing the development time.

This thesis presents a novel architecture known as a semi-autonomous architecture (SAA). The research involved investigating and incorporating specific sensors, and also developing a common software module to perform the localisation, navigation and mapping particularly suited for corridor crop agricultural environment. This architecture was integrated and implemented on a Yamaha golf cart, infusing it with purposely positioned sensors and supportive electronics to allow a Robotic Operating System (ROS) framework to gather information and control the vehicle. As the architecture is modular in nature, it can be transferred to different customised platforms.

To determine the efficacy of the SAA, the platform went through a field trial and simulations to test the fundamentals of an agricultural AV. This meant investigating the SAA's ability to map, plan paths, avoid obstacles and maintain a specific distance from a row. The evaluation demonstrated that both 'RTAB-Map' and 'Gmapping' could produce accurate maps of a vineyard. Additionally, grid-based planners, such as Dijkstra and A*, performed better than a sample-based planner bias-IRRT*. Regarding obstacle avoidance,

the SAA required a greater range of sensors to detect small and oddly shaped objects that could be found in a vineyard. The SAA maintained within 5mm of the specified distance when aiming to follow the row. The results from these experiments demonstrated the SAA's ability to successfully transform a Yamaha golf cart into a semi-autonomous agricultural vehicle.

Acknowledgement

Firstly I would like to thank my family and friends for supporting me through this masters. I want to acknowledge the Maara tech project and group for funding this research and providing a wealth of knowledge when assistance was needed. Important acknowledgements to cover are the likes of Dr Shen Hin Lim, who provided excellent guidance in the structuring of the research and support in academia of the project. Prof. Mike Duke provided support throughout the project, not letting any issue sway him from getting what I needed to complete it. I would also like to acknowledge the people from The University of Waikato who assisted me through this project, particularly the D.G. 12 team. This team included the likes of Andrew Tattersall, who designed and constructed the custom frame for the project, and Matthew Pebbles. Their wealth of knowledge in ROS provided excellent support throughout. Lastly, I would like to acknowledge the businesses that assisted me. Starting with Matthew Seabright from Axis7, who assisted in localisation development, Robotics Plus Limited for providing the base vehicle that the architecture was placed on, and finally, the Tuki Tuki Vineyard, owned by Pernod Ricard that was used for field trials.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Improvements in technology are to further benefit the lives of people and the world around us, with inventions such as the wheel, bike, combustion engine and one day the fully autonomous vehicle (AV). The idea that one day humans won't have to drive on the road benefits both ourselves and each other, with most accidents occurring due to human error. Many companies [1][2][3] have taken on this task, and are etching closure to full autonomy. In a relatively short time, autonomous vehicles (AVs) have come a long way and are assisting people in their day to day needs, be it while shopping or when travelling [3]. An area where AVs are assisting in, but are not as well known by the public is in the agriculture sector. There have been developments in autonomous agriculture since the 1950s [4], with more in-depth demonstrations of autonomous driving from the 1980s [5] to this very day [6]. This development in AVs for the agriculture sector is paramount to minimize laborious tasks and maximise produce, with fresh fruit production constantly increasing with 887.03 million metric tons globally in 2020 [7]. By being able to integrate technology into this field, greater stability can be brought to its production, relieving some stressors from the growers. When reviewing the current literature there are vehicles already that are capable of navigating to a reasonable level of autonomy within agriculture.

1.1 Agriculture Automation

The agriculture sector encompasses many tasks that can be improved by using an AV, such as assisting harvesters by carrying equipment, performing the harvesting itself, and

data collection for decisions on what to do or enhance for future harvest. As a whole, the agriculture environment is physically demanding on workers and financially stressful for the growers due to seasonal changes and labour shortages; these are some of the reasons that led to the research into assisting and automating work areas. This innovation is done for the same reason as other technological improvements in different fields; to make life for the growers and workers safer, efficient and consistent, with the end goal being an intelligent system where everyone can benefit and the AV being a vital role in this future.

There are a few ways to integrate automation into agriculture, one that changes the growing methods through placing crops into factory-style buildings [8], and another that is to design robotics around the current crop layout. The latter is a route that is taken in this thesis and is possibly the most sought after automation at this stage, due to its use of current infrastructure. This infrastructure can be seen within New Zealand alone, where there is 64580 hectares of wine grapes and 12710 hectares of apple orchards orchards as of June 30$^{\text{th}}$ 2020 [9], with the global amount of grapes and apples produced in 2020 being 78.03 and 86.44 million metric tons [10]. These numbers can be taken as a significant amount of investment that farmers have placed into their infrastructure and therefore will want to make the most of it. This is further verified by the large development and investment in to autonomous technology, specifically seen in tractors with John Deere producing their first fully autonomous tractor this year [6] and other companies following suite [11].

Farms often usually covering hectares of land, there will always be a need for vehicles to move tools, pick crops and transport staff. This need for vehicles is where AVs can assist in; starting in more closed areas such as pruning and harvest, with progressions to all other areas. Using AVs could significantly lower injuries and fatalities in the agriculture sector, with 90% of fatalities resulting from machinery and vehicle use in New Zealand [12]. The research into autonomous vehicles for different agriculture areas such as vineyards can potentially save lives and reduce injuries for the people working in these fields.

The current autonomous agricultural vehicles are being developed to be task specific and therefore lack the versatility to be applied around different farms and crops. Where common sensors such as LiDAR, camera, RADAR, ultrasonic, GPS, IMU, encoders and

functions such as localisation, navigation and mapping are built-in to fit to each vehicle. These sensors and functions can be built as a common and modular architecture and can be transferable across different vehicles that operate in the corridor crop agricultural or horticultural environment.

1.2 Objective

The objective of this thesis is to presents a transferable architecture that transforms a vehicle into a semi-autonomous platform, which is capable of traversing corridor crop environments.

1.3 Methodology

The main concept for this work was to integrate and implement a transferable semi-autonomous architecture (SAA) on a vehicle. This was done to fill a gap in existing research around transferable autonomy for agriculture vehicles. As such, this meant that both real world application and simulation were used appropriately to collect data representative of the SAA's performance. The University of Waikato provided the area for initial development, with greater understanding coming from a field trail at a vineyard in Hastings, New Zealand. The simulation results were achieved through the physics engine 'Gazebo' [13], that made it possible for experimentation on vineyard-like worlds, and rapid changing of worlds to investigate different aspects of the SAAs autonomy. The experimentation was targeted at challenging the different fundamentals of autonomous driving for agriculture, such as, mapping, path planning, obstacle avoidance, and a potential need for maintaining accurate distance from a row. To investigate these characteristics of autonomy all experiments involved adjusted a single parameter of the test, which was then accompanied by a test in a more realistic vineyard environment. These experiments were then run multiple times to make sure results were consistent and that useful information could be extracted from the data. The use of Robotic Operating System (ROS) [14], and its 'bag' system allowed for the visualisation and recording of sensor data during the experiments. This made it possible to replay data from historical testing on multiple configurations of

software allowing for greater comparisons. The use of standard deviation, mean values, and general statistics allowed for the data to be analysed to determine accuracy and speed of the autonomous characteristics. This quantitative analysis was also assisted through qualitative analysis to provide another approach and re-iterate the message from the quantitative results. This approach was taken to verify the SAA ability to raise a vehicle autonomy level from 0 to at least a 3[15] making it semi-autonomous. Without this investigation of the different components this understanding would not be achieved.

1.4 Contribution

- A semi-autonomous architecture that is capable of integrating into a vehicle, and providing the ability to navigate and map corridor crop environments.
- A ROS architecture that is able to control an ackermann based vehicle.
- A Gazebo simulation for testing autonomy in a vineyard environment.
- A comparison between grid and sample based path planning algorithms for a corridor crop environment.
- A behaviour tree structures to manage the ROS nodes, allowing for changes in the state while navigating.

1.5 Overview

This thesis has in total five chapters that cover the different aspects of the research that was done for the SAA. The literature review that assisted in determining the need for this thesis is outlined in chapter 2. Following this, the manners in which the specific architecture was developed and integrated are outlined in chapter 3. This architecture is then tested and reviewed in chapter 4, with the concluding remarks and future work found in chapter 5.

Chapter 2

Literature Review

This chapter investigates existing literature around autonomous vehicles. These findings identify gaps in current literature that determine the need for a transferable semi-autonomous architecture (SAA) for agricultural vehicles. Autonomous solutions for road, agriculture, and factory tasks were reviewed to get a comprehensive understanding of suitable sensors. This is further expanded to investigate the mapping and navigation methods that allow for the vehicle's autonomous nature.

2.1 Road Vehicles Current Level of Autonomy and Suitability for Agriculture

Many companies are looking into commercial AVs for everyday drivers, such as Tesla [16][17] , Toyota [2], Waymo [3][18], Audi and many more [19]. A way to classify the autonomy of AVs is by using the SAE levels of automation, as seen in Figure 2.1 from [20]. As of 2019, there were no commercial vehicles above level 3 [21]. When comparing level 5 to level 3, there is not a significant difference in levels; however, the technological gap is significant, showing that there is still a lot of development left in the AV field. Even though companies have no high-level autonomy vehicles, they still provide good overviews of the current and potential vehicle systems.

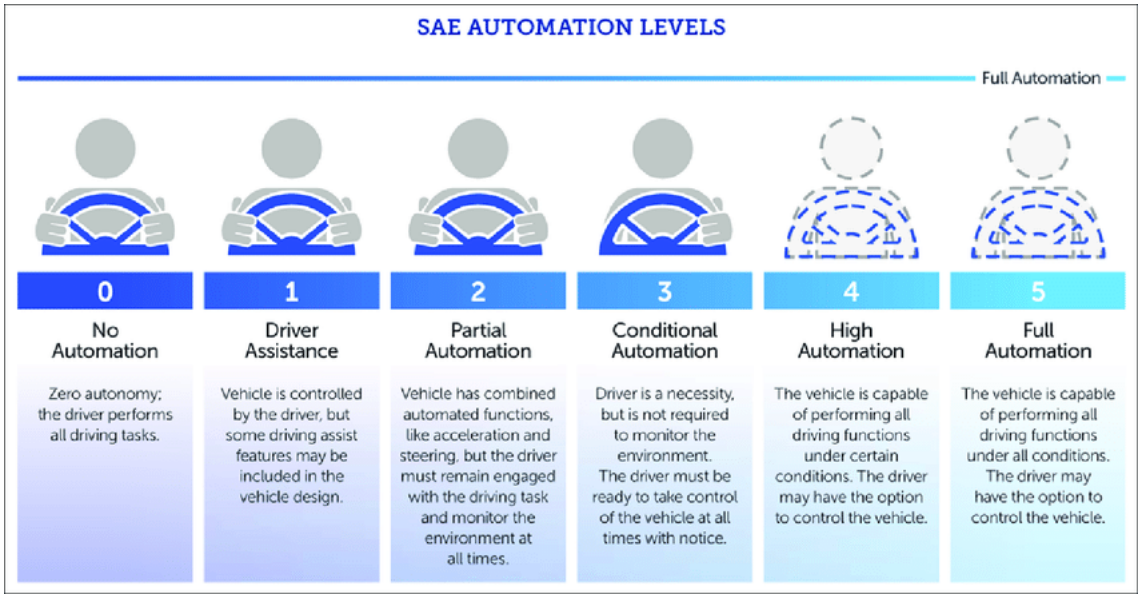


Figure 2.1: SAE levels of automation.

Toyota produced a white paper [2], where it covers their outlook on AVs, the key aspects of AVs, deployment of personally-owned vehicles, and the current technology on their vehicles. This company looks to improve on the current safety system that incorporates RADAR to detect obstacles, and reach an autonomy level of 2 in 2021. This work looks to be in tandem with the development of an autonomous mobility service or "MaaS", which is meant to be of level 4 autonomy when finished and act as a ride share service. A platform designed for this purpose called "e-Palette" was used during the 2020 Tokyo Olympics, but was stopped due to an incident occurring while travelling [22]. This incident is not the only one, as within the years 2015-2020, there was 316 collision reports and 303 in 2021 from the California state alone [23]. These incident reiterates the further development still required in the autonomous vehicle space. The white paper mentions the types of sensors they believe are required for an AV. These sensors are cameras, sonar, LiDAR, RADAR, GNSS, IMUs, wheel speed and angle monitors. This wide range of sensors is seen in other research through this review. Therefore this provides an important starting point for sensor decisions if one of the biggest companies in the world also sees the potential in having all of those sensors.

A few studies have taken on the task to investigate these sensors for both highway and

rural environments [24]. This paper is based on the highway, and rural environments within the Netherlands and further considerations are required when reviewing this research.

2.2 Sensor Requirements for Road Vehicles

2.2.1 Theoretical Analysis of Sensor Requirements for Road Vehicle

The paper [24] firstly investigates the minimum requirement of road vehicles and further assists in determining the agriculture environment requirements. These categories can be defined in the vehicle's operational design domain (ODD) [25], with the end goal of having it work day or night, sunshine or rain, and maintain its path while avoiding obstacles. The ODD results [24] starting with the highway environment demonstrate that for transport vehicles in high-speed road usage, there is a significant focus for obstacle detection directly in front of the vehicle. The analysis showed that obstacle information only needed to be known up to 4.55 m lateral, 67 m backward, and 134 m forward for the highway scenario. This analysis was based on local road speeds, lane sizes, and road curvature but still provided helpful information for most highways and demonstrated how they were calculated, allowing it to transfer to other scenarios such as a vineyard. The emphasis on front-facing detection in the highway environment is understandable as the most significant risk or danger comes from a potential obstacle stopped in front, such as sudden traffic or a crash. This risk is further back from data found from the Waymo safety report [3], which showed that 29% of crashes are a rear-ended scenario. In contrast, the sides and back detection mainly determines if a lane change is possible. The highway environment demonstrates the required sensor parameters for high speed and confined movements, which does not apply to a vineyard environment. An AV speed while pruning or harvesting is likely slower than an average person's walking speed. This environment provides insight into sensor requirements due to the vineyard rows acting as "road lanes", but a similar analysis is seen in the rural setting.

The rural environment provides many more challenges as there are more significant

amounts of traffic in both vehicles and pedestrians while also encountering much more complicated driving scenarios in the form of different intersections. This difficulty is easily seen through the percentages of crashes at intersections being 24%, being the second highest being rear-ends [3]. The rural environment looks to have more in common with a vineyard environment, with many scenarios being the same or similar. The research looked into a total of eight settings for the rural environment and presented surprising results, with relatively long-distance detection required in every direction. These distances for the sensors to cover was just over 150 m forward and to the sides, and backwards only requiring just over 60 m [24]. This extensive coverage is due to the rural environment expecting speeds up to 80 km/h and requiring the vehicle to perform overtaking maneuvers which can, in turn, make the speeds being dealt with doubled due to oncoming traffic.

The literature looked into currently available sensors on the market to cover the defined regions of interest from the prior two analyses. It determined that sensors such as LiDAR or RADAR be used for long-range detection, with RADAR potentially requiring sensor fusion to overcome poor angular resolution. At the same time, the LiDAR has good resolution but can struggle in poor weather conditions. The camera sensors would be mainly used for object and lane detection or potentially be fused with RADAR to provide a better field of view. The ultrasonic sensors will be used for short-range maneuvers such as parking and will also assist the other sensors at short range due to them not detecting obstacles at very close range. The literature did note that there are no sensors that could accommodate the requirements for overtaking a vehicle with potential oncoming traffic in the oncoming lane due to the net speed at which the traffic would be approaching. This claim is somewhat challenged through the safety report by Waymo [3], who present their LiDAR, who can see "almost three football fields away" or approximately 270 m. Given that the LiDAR is not publicly available means the papers [24] claim can still stand.

As noted prior, one of the main reasons for the extensive range required by the sensors in the research is the speed of the environments. The speed meant that there was less time to determine if an emergency brake, lane change or lane sharing was required. Since the vineyard vehicle is likely to operate at much slower speeds, it may allow for not as many

long-range sensors such as LiDAR and RADAR, which can decrease the computation and cost for the AV.

The sensors and reasoning for sensor placement can be transferred to an AV that will operate in a vineyard. Still, due to the vineyard being a more closed environment than the open road, the same amount and sensor choices may not be required. This deems that further research for an appropriate ODD of a vineyard AV is defined and testing around this design to narrow down a sensible sensor selection. Current road AVs do provide a greater understanding and a good starting point for both sensors and the task that the vehicle needs to handle to be fully autonomous.

2.2.2 Practical View of Sensor Selection and Placement for Road Vehicle

The BRAiVE vehicle (BRAin driVE) [26] has an extensive range of sensors consisting of ten cameras, five LiDARs, one GPU+IMU unit and an E-stop system. These sensors are placed in specific areas to provide 360° of coverage for the vehicle. The camera system involves four cameras behind the windshield acting as two stereoscopic pairs with different baselines, providing both obstacle detection and depth information for the vehicle. The obstacle detection from the front-facing cameras combines three LiDARs on the front bumper, with two planners (2D) LiDARs facing diagonally out with one multi-plane (3D) LiDAR facing forward. The purpose of these LiDARs was to provide obstacle detection and assist in localization. The multi-plane LiDAR is an IBEO Lux that allows measurements through atmospheric clutter like rain and dust. This ability to still detect objects in adverse conditions would be needed while operating in a vineyard, with dirt in the air and the potential for bad weather to be a high possibility. The other sensors are placed around the vehicle to assist object detection and localization. The BRAiVE vehicle has built-in redundancy of sensors, but there seems to be potentially a lack of diversity in the sensor selection. This observation comes from noticing that the only sensor that can work well in adverse conditions is the multi-plane LiDAR which only faces forward. Having the one sensor that can detect objects in poor environmental conditions

facing forward is understandable. This is the main direction of travel and the area at which obstacles will be coming at the vehicle the fastest. With that being said, some scenarios could require more robust sensors, for example, lane changing in heavy fog or rain.

A commercial vehicle that presents their sensors and sensor placements is Waymo [3]. The developed their vehicle to have 360° of vision, and has the same sensor choices as determined by Toyota in their white paper [2]. These sensors include a range of LiDARs and Camera systems around the vehicle, a 360° RADAR on top of the vehicle, an IMU and GPS for localisation and also supplemental sensors such as audio detection to hear emergency vehicles.

The sensor decisions and placements can potentially work well in a vineyard environment. Still, there may be a need for a more extensive range in sensors and similar to the analysis in the prior paragraph [24], there may not be a need for the number of sensors for a vineyard AV.

2.3 Defense Advanced Research Projects Agency: Autonomous Vehicles

The Defense Advanced Research Projects Agency (DARPA) hosted two AV competitions to drive the innovation within the field. These two competitions were the Grand and Urban challenges, and targeted different scenarios for an AV.

2.3.1 Grand Challenge

A review of two different vehicle systems demonstrates the steep development of AVs because of the Defense Advanced Research Projects Agency (DARPA) Grand Challenge. The vehicle needed to be prepared to travel up to 175 miles (282 km) in under 10 hours through the Mojave desert. The following paragraphs demonstrate similar ideas and differences in preparedness for the competition.

A vehicle that competed in 2004 and the 2005 Grand Challenge was 'RASCAL' [27][28], which had implemented a multi LiDAR configuration with two horizontal and vertical SICK planar LiDARs. Combined with the LiDARs, the vehicle was meant to have an array of ultrasonic sensors, pinhole cameras, and RADAR for obstacle detection while incorporating DGPS (Differential Global Positioning System) and IMU (Inertial Measurement Unit) for localisation. The localisation system used an Extended Kalman Filter (EKF) to utilise each localisation sensor fully. This meant dead-reckoning was done using the IMU between GPS readings or while the GPS signal was lost. The GPS provided data at 5 Hz and had an accuracy of 10 cm while the differential signal was received and had anywhere from 1-3 m when it wasn't. This variance in positioning is much more significant than what the Stanley team deemed suitable to complete the 2005 course, mentioning even at a displacement of 0.66 m [29] in some regions of the course was enough for in-completion. Given that 2004 and 2005 were both at the same venue, [28] it can be said that this variance, even if RASCAL had no issues, was unlikely to complete the course. During the build of RASCAL, the RADAR had not been delivered and therefore was left out, meaning no results were shown for this competition. The RADAR was later tested before the 2005 competition, showing promising results in detecting metal objects (e.g. fences). However, it was still not fully integrated [28] and therefore could not provide any results on the RADAR. The LiDAR system provided essential obstacle detection up to 80 m but showed technical issues with the vertical LiDARs not being used in qualification. This system was later attempted to be approved upon, with shielding from external light (which was not used due to the metal shields causing measurement errors) and positional changes for the sensors, placing one of the LiDARs on top of the vehicle to help detect road boundaries. It wasn't apparent in the paper [28] if this would be used in the 2005 competition. RASCAL in 2004 was presented as a rushed, not thoroughly tested AV, and this was apparent with the problems it faced leading to and within the competition, only completing 0.75 miles (1.2 km) of the race. The following year of development on RASCAL looked to provide a more robust system through testing practically and in simulation (using Gazebo), but with minor changes to the sensory system through the use of a stereo camera set up instead of the single pinhole camera. These changes overall were not enough to help RASCAL complete the course [28].

The Stanford vehicle named 'Stanley' had not competed in the 2004 competition but had won the 2005 competition, unlike RASCAL. The following takes an in-depth review of the vehicle and the design decisions that helped it win the competition. Stanley's base model was a diesel-powered Volkswagen Touareg R5, which was then retrofitted with electronics to allow for autonomous control. The obstacle detection and mapping sensors comprised five SICK planar LiDARs (all at slightly different tilt angles), an RGB camera (tilted slightly down), and two 24 GHz RADARs [29]. The localisation sensors involved a 6 Degree of freedom (DOF) IMU, GPS, and dual GPS. The software used to manage the localisation were two Unscented Kalman Filter (UKF) models to create predictions based on the measurements in two different scenarios, one where GPS is available and the other while GPS is unavailable. This localisation method meant the vehicle could be without GPS for up to 2 minutes and still maintain centimetre accuracy [29].

Within the paper [29], it can be noted that the vehicle relied significantly on the point cloud data from the array of LiDARs. This comes from the LiDARs providing 25 m's of detection while also verifying data for the camera to determine a drive-able area. If the camera failed, it would take over that role and slow the vehicle down to compensate for the reduction in detection range. This reliance on one type of sensor can present issues with resilience to harsher weather conditions, but due to the competition being one of the first large scale stepping stones to further AVs, it can be deemed suitable for the competition.

These sensors were placed facing forward of the vehicle on a custom made roof rack for the vehicle. The purpose of the sensors' pose was due to the race only involving forward [29] direction of travel in a reasonably static environment. This was taken as far as having other vehicles put on "pause" (stopped) if one vehicle was about to overtake another; this meant the vehicles only had to plan for stationary obstacles.

When comparing the sensor systems of the two vehicles, they both have similar concepts, using separate sensors to handle the different tasks. This is seen through sensors

covering various ranges of obstacle detection while utilising a GPS as a primary source of positional data aided by an IMU through a specific type of non-linear Kalman filter. The main difference that benefited Stanley over RASCAL included the accuracy of the localisation system, ability to detect drive-able areas, ability to track the road's centerline (compared to following way-points blindly), and the ability to adapt to the loss of sensor data. Each aspect is critical in autonomous driving and will require an AV to operate in a vineyard environment.

The fact that the Grand Challenge was done in a static environment, as stated in this paper [29], then mentions that this is the most crucial problem raised from this challenge. Saying that "Stanley is unable to navigate in traffic" and that for an AV to succeed, the ability to travel in dynamic environments has to be addressed.

Overall, the vehicles that competed in the Grand Challenge present both sensor and software solutions for taking on such terrain, but they differ significantly compared to a vineyard environment. With this, the localisation sensors and software to manage them applies to the agriculture environment and is worth looking further into. The obstacle detection sensors are useful with the vineyard environment still requiring mainly forward driving and accurate obstacle avoidance. Therefore the use of sensors such as LiDAR (that can provide 3D point cloud) and a vision system would be able to provide adequate information about the rows and also be able to classify obstacles. Due to possible tasks of an AV within a vineyard being more than just continuous driving, it may be required to employ robust close-range sensors such as RASCAL, which used an ultrasonic sensor array. This would increase the chance of stopping when more dynamic obstacles are around and assist during any reversing of the vehicle.

2.3.2 Urban Challenge

The DARPA Urban Challenge took on the problem of traversing dynamic environments, which had the 11 vehicles travel through three different missions, with each having six submissions. Multiple contestants would operate on the same course simultaneously and

up to 50 other human drivers, attempting to make the urban environment more realistic while still not incorporating any living obstacles such as pedestrians and animals. By having a more dynamic environment than the Grand Challenges in 2004 and 2005, it was found that the vehicles that competed in the Urban Challenge had a range of sensor systems, varying types of sensors and placements of these sensors [30][31]. Good examples of sensor selection and layout were the 1st and 2nd place contestants, 'BOSS' (by Carnegie Mellon University) [30] and 'Junior' (by Stanford) [32].

BOSS used many sensors, including cameras, RADARs, a Differential GPS, IMU, wheel encoders, and different types of LiDARs [30]. The BOSS sensory system was designed to provide the necessary coverage and redundancy for urban travel. This is seen in the vehicle's positioning and range of sensors, having at least two sensors in the primary directions of travel (forward and backwards), with the majority facing in the forward direction. It can be noted that the two sensors facing rearward are a LiDAR and RADAR, providing redundancy even for adverse weather conditions. The majority of sensors are facing forward due to the need to detect obstacles and road shape while driving, with assisting LiDARs on the sides of the vehicle for road feature detection. This placement and combination of sensors provided the understanding for the internal systems to detect and track obstacles (dynamic and static) whiles also localising the vehicle relative to and estimating the road shape. This greater understanding of the position and possible obstacles allows the behaviour manager of BOSS to determine the next best course of action. The behaviour manager can be separated into three sections 'Goal Selection', 'Lane Driving', and 'Intersection Handling'. Each has lower-level systems to handle scenarios in which the Urban Challenge placed the vehicle. Overall, this design and preparation of the vehicle led to the BOSS winning the competition, with relatively minor incidents during the competition. These events included minor sensor issues in LiDAR calibration and perception system incorrectly estimating an on-coming vehicle, causing a quick position adjustment. The paper stresses the importance of the road-map localisation system and that without this system, BOSS would have been off the road or in opposing traffic, with the peak error distance being 2.5 m's from the normal GPS localisation. The paper mentions that at the current stage of development, off-the-shelf

sensors are insufficient for proper urban driving due to range and sufficient coverage, which is similar to the conclusion by the paper [24], with rural areas. Interestingly, during the competition, the human drivers quickly adapted to the AVs, providing some evidence to demonstrate that integration of reasonably simple AVs in a vineyard environment is not such a significant step for workers.

Junior's team was comprised of mainly the team who created 'Stanley', which can be seen through sensor selection. Junior used multiple sensors but still relied heavily on one type of sensor, the LiDAR; this is very similar to Stanley's, but what differs is the types and position of these LiDARs. The vehicle had a total of eight LiDARs. Seven of them are supplementary to detect close curbs, detect lane markings and cover the blind spots of the primary sensor (the Velodyne HDL-64E) due to self-occlusion. Besides the LiDARs, Junior also incorporated five RADAR, which assisted the main Velodyne LiDAR as an early warning system [32]. One sensor that was left out compared to Stanley was a camera; this is a questionable decision as a camera can provide a substantial amount of information about the environment compared to other detection sensors. The use of a camera may have assisted during the competition when Junior was attempting to pass 'Little Ben'[33] but failed to detect a drive-able area, [32] unlike Stanley was able to do. If this competition had progressed to incorporating living obstacles and maintaining the level of autonomous driving, Junior would likely need the use of a camera to determine the difference between particular objects. For localisation, Junior employed similar sensors to Stanley, using Dual GPS for position and heading (with the use of Base Station Corrections), 6 DOF IMU, wheel encoders. This provided the vehicle with positioning typically below 0.1 m and 0.1 degrees of error. Due to the success of the Junior, it can be said that the sensor system was also very successful in localising and detecting obstacles. There can still be a lack of sensor diversity if harsh weather conditions are faced and the environments become more complicated with other types of transport and living obstacles. A similar system could work very well in a vineyard due to the complexity of the environment being the same, if not less so. Therefore the Junior sensory system presents a reasonable basis for what could be placed on an AV in a row crop environment and should be considered during the design. The behaviour manager for Junior was a finite state machine (FSM) (containing

13 states) that would handle the control of any situation Junior would encounter. Overall, Junior was able to handle the competition very well using the FSM, engaging with around 200 other vehicles and going through many environments (intersections, parking lots, dirt roads). Only minor issues arose, such as mentioned above with 'Little Ben' and Junior not understanding what to do when stuck in that situation and pulling alongside a waiting vehicle thinking this vehicle was stopped.

When comparing the two vehicles, BOSS[30] and Junior[32], they differ mainly slightly in sensors, with BOSS using cameras to assist in detection. Both vehicles worked exceptionally well, shown in their completion times and low amount adversities during the competition. The main differences between the two vehicles come down to the software, both tackling the same problems but with slightly different approaches. Both papers agreed that better recognition of objects and vehicle intentions is required, with the main scenario being a stopped vehicle or a parked vehicle. In summary, all software parts worked well in the urban environment, but it would be interesting to test them in vineyards to see if they can provide equal success.

The two competitions, Grand and Urban provided excellent stepping stones for AV development, starting with an appropriate static environment (but still challenging course) transitioning into a much more dynamic one. Overall when looking at the Urban Challenge vehicles, they would be able to take on the Grand Challenge. When looking at the sensor system of the vehicles, they were all very similar and with similarities in environment styles between the Urban challenge and vineyard like environments. The Urban Challenge vehicles should be considered for designing an AV for the agriculture sector but not directly copied as this would likely be too much for a less open (, more static) environment.

2.4 Autonomous Guided Vehicles

AVs used within warehouses, factories, and controlled hazardous zones are commonly autonomous guided vehicles (AGV's) [34][35][36]. These vehicles have specific predefined paths that can be navigated. The routes are physical (wire induction, magnetic strip, coloured line, walls, etc.) or virtual (plans on a pre-mapped area stored in a computer)[36]. This type of AV could be used for a vineyard AV and potentially provide valuable solutions

to the more closed areas (such as within the rows) that a vineyard vehicle will face. In the following sections, the methodology and systems used in research will be investigated for potential usages within the agriculture sector.

Some of the more common AGV technology available was covered in [36], which covers six different sensor solutions for AGV navigation within nuclear fusion facilities. These solutions involved a mixture of physical and virtual paths using wire induction, magnetic strip, optical guidance, on and off-board lasers, motion capture, and a magnetic-gyro grid. The paper compared the solutions of different criteria centred around the needs for the navigation of a nuclear facility, making sure the AGV would be able to navigate off the path if required. These criteria included the following with the associated weights (out of 100), technical feasibility (40), robustness (30), cost (10), the ability for replacement (10), and commercial off-the-shelf (COTS) (10). After the comparison, the solution that had the best score was a combination of wire induction and optical guidance as primary navigation, with a secondary system in off-board laser guidance. This was due to the sensors being relatively cheap, easily replaceable, and different from each other, making a more robust system. The issues with the individual systems still affect the combined solution, but now there is built-in redundancy for any failures due to electromagnetic interference or wearing of painted or taped lines. The second-best solution was a tie based on the criteria between just wire induction guidance and onboard laser guidance. This paper concluded that the combination of wire induction and optical guidance, with off-board laser guidance, was still the best, followed by the full laser guidance solution using onboard laser guidance as to the primary system and off-board laser guidance as secondary. This paper [36] provides beneficial information on the potential AGV systems and a suitable method for comparing these systems. It would've been good to see the weights applied to the sub-criteria as this can likely change the outcome from the "trade-off" comparison.
The research was directed at creating a navigation system that can allow the vehicle to move off the predefined paths is a concept that would have to be implemented in a vineyard environment, with workers and other machines in the area.
The best solution from this paper is questionable for testing in an outdoor environment. The setup suggested using the optical sensors is unlikely usable for tracking a paint or

tape strip on the ground due to dirt and general wear and tear due to environmental factors. The wire induction guidance system could work within a vineyard and was ranked equal to the laser system based on the criteria. Still, due to a vineyard not being perfectly flat and the ground conditions constantly changing, this would require a more significant amount of testing in these outdoor environments.
The laser-based solution shows promise for a vineyard environment but only for the primary navigation sources, with the off-board laser system likely not working well in outdoor environments due to poor weather conditions, the field of view (FOV), line of sight (LOS), and the size of coverage required on vineyards and farms alike. The laser system does use external reflective markers for localisation in the research, which could become dirty, faded, or overgrown while being outside. Systems later on in the review use a similar localisation method but can use biological markers (tree trunks or posts) instead of placed ones, making this solution viable, with further comparisons in marker types.

The onboard laser navigation from the paper [36] above uses similar principles to a wall follower vehicle. The vehicle maintains a specific distance from a wall(s) or wall-like objects. A couple of papers go into systems to track a wall, mainly a vineyard, orchard, or any similar corridor row crop.

The camera-guided AGVs are still a possible concept for environments such as vineyards with fixed structures along the path (rows), creating corridors like movement constraints. The guidance method uses the corridor shape to distinguish the path to follow compared to using coloured tape or paint, as seen in other papers [37][36]. The work presented in [37], is aimed at using a single mono-camera to perform navigation in corridor-like environments.

An AGV would be a suitable solution for the rows within a vineyard. Still, a vineyard is more than just a single plot of rows, and it is unlikely to be a total solution for an AV within a complete operation vineyard.

2.5 Agriculture Autonomous Vehicles

As part of this review, there needs to be an insight into the research done for AV's in the agriculture sector. This should provide information on methods and sensors tested in other areas that could be applied to a vineyard. Similar layouts to vineyards are other fruit orchards, such as an apple orchard, where a study looked into a low-cost AV that assisted workers. This assistance was provided through the platform being a structure that could be stood on and moved with them through the orchard [21]. This AV relied heavily on a 2D LiDAR to localise and detect objects within the orchard. The system used the LiDAR (positioned at the front of the vehicle) to measure the distance to the trees (within the row), which allowed the vehicle to find the centre-line of the row, therefore localising itself in the row. Using a LiDAR provided the accuracy and reliability for the researchers to test the primary purpose of the study, which was to compare an MBC to a standard PPC for an orchard environment. The sensor was used to determine the end of rows by checking for free space greater than 90° of the vehicle front, while the start of the row was detected through reflective surfaces placed on the first poles of the rows. These methods used to navigate the orchard rows are simple ideas and kin to an AGV mentioned in the previous section. This research demonstrates how an AV would operate using a row follower method with reasonably good results, moving through multiple rows and the vehicles having covered 350km over different orchards and vineyards. Still, there is potential for improvement by adding a relatively inexpensive camera, a computer that can run object detection, and now there are GPS units that are much cheaper than the papers suggest. This would give the AV a greater understanding of the objects it detects through the LiDAR. Having these additions means not relying too heavily on encoders that have the potential for slip error and an increasing error over time, limiting the vehicle to only the row environment with the position being zeroed at the start of each row to eliminate this error. A previous paper published [38] for the same project showed information on work efficiency showing an average of 42.5% improvement in over ten different tasks performed on orchards that the AV assisted. This is one result that further solidifies the potential in the agriculture sector for AV's.

The paper [39] demonstrates a custom-designed vehicle to facilitate the requirements of a kiwifruit orchard. The vehicle evaluated three LiDAR's and three types of cameras for obstacle detection. The LiDAR's consisted of two single-plane sensors (UTM-30LX and a SICK LMS111) facing the sides and one multi-plane sensor (VLP-16) facing forwards. The cameras that were tested included a time-of-flight (TOF) (Basler TOF640-20 GM-850NM), 3D stereo (Intel RealSense R200) and traditional 2D cameras (Basler Dart daA1600-60uc, Flir CM3-U3-13S2C-CS, and a Logitech C920).

The LiDAR's core intention was to use them for detecting structures within the orchard, such as posts trunk and hedges. This was challenging due to the 2D LiDAR's detecting either the canopy or the ground due to the uneven surface of the land. This made these LiDAR's only suitable for close obstacle detection, while the 3D LiDAR could still be used through post-processing, choosing the laser scan from the 16 that provided the best information about the surroundings.

When experimenting with the TOF camera as a forward-facing sensor, it produced less than desirable results in most weather conditions. This was expected to be caused by the sparse amount of reflective objects within a suitable range while driving. These results were shared by the stereo camera, which had a complete loss of range data during operation. Interestingly this stereo camera is an active sensor that uses an infrared (IR) light to illuminate the surroundings, with the reflections measured by an IR lens to calculate depth. Because of this, the assumption was that the failure was due to the ambient light interfering with this signal. This poses a question to use a passive stereo camera such as the ZED camera [40] or self-made system as seen in [41]. The 2D cameras experiments showed that the Basler had the better image sensor and fastest update rate of 71 Hz. The FLIR was close seconds but had a much slow update rate of 15 Hz, while the Logitech web camera was not usable due to motion blur and update rate of 7 Hz.

Therefore, the VLP-16 LiDAR and Basler camera were the final choices from all object detection sensors. The fact that a passive depth camera was not looked at during the evaluation is an oversight by the researchers, with no reasoning behind not including one.

Even so, the results from the investigation in obstacle detection provide insight into these sensors for agricultural usage.

The vehicle used wheel encoders, IMU, a digital compass, and GPS to assist in localisation. The paper covered the issues found with each of these sensors; the encoders and IMU had accumulation errors over time, the compass was affected by the metal structure with the orchard, and the GPS had lost signal within the canopy. The primary localisation sensor that was not useful during this research was the GPS unit, with data being either sparse or incorrect during canopy navigation. These localisation sensors are seen in other literature reviewed and have proven successful; therefore, it would be helpful to get results for similar sensors in a vineyard environment.

2.6 Autonomous Vehicle Communication

A smart vineyard is an idea that incorporates all the new technology going into vineyards and farms alike to improve the understanding of the crop and to make the whole farming process easier. The AV will be part of this smart vineyard and looks to be a crucial role within it, as it will be able to help collect data and act on this data [38][39]. The integration of AVs into a smart vineyard will require communication between the vehicles, IoT (Internet of Things) devices, and a central hub to relay the information to a web interface for easy management [42] [43][44]. This communication between these blocks is V2V (Vehicle to Vehicle) and V2I (Vehicle to Infrastructure). The potential for improvements in AVs from the use of such communication could overcome the shortcomings of visibility from current obstacle detection sensors, as mentioned in previous sections for urban [30] and rural areas [24]. These thoughts stem from the papers such as [45][46][31], which demonstrate that vehicles will be able firstly know about and then prepare for road congestion, collisions, and potential collisions from vehicles that are not within direct line of sight with companies such as BMW, Ford, Audi, Mercedes, Opel and Volkswagen performing tests on there own vehicles in 2012 [45]. V2V and V2I seem like a very promising technology for AVs and vehicles in general, with the likely first step being to integrate this into vehicles to assist human drivers. When level 4-5 AVs arrive, the systems will be in place to use

this technology. An example of this development and integration is seen in the paper [46], which demonstrates the use of smartphones to compute the distance and speed of vehicles that could be potential hazards to itself (ignoring in-active vehicles). This information is then communicated through the cell phone network to warn drives in the area if required.

The same ideas can be used in agriculture, with farms becoming more hi-tech every year, introducing IoT devices to monitor livestock [47][48], crop [44][49], and produce. At this stage most V2V and V2I research is based around road vehicles, while the agriculture sector does show interest and usage of IoT systems it lacks this direct development. The usage of IoT frameworks does present a great starting point for AVs to be integrated into agriculture with the V2V and V2I communication, with research being done on what this communication will look like [50].

2.7 Navigation, Localisation and Mapping

2.7.1 Global Planners

The grid-based planners covered in this review are Dijkstra and A*. The Dijkstra [51] and A* [52] planners are similar in construct, with the A* algorithm being seen as a more refined version of Dijkstra [53]. The paper used the Dijkstra planner [54], which made a path for a small indoor robot through an office space. The entire trajectory of the robot involved planning from one way-point to another, with this trajectory providing a suitable path that made the navigation successful. A version of the Dijkstra planner was also implemented to produce a path for an AGV that would allow an industrial vehicle [55] to navigate between set areas as efficiently as possible, attempting to lower energy requirements while carrying loads. This meant the planners' cost was based on energy consumption, which would take into account distances, slopes, and weight of the load. This research did demonstrate that using the algorithm for planning the paths for these vehicles can minimise energy consumption. These results show the versatility of the algorithm and suitability to AGVs because of how its normal cost function works. This concept of planning based on energy consumed could be applied to vineyard naviga-

tion, but likely for low fuel or energy moments where the AV requires to re-fuel or re-charge.

Following the Dijkstra planner, A* was heavily covered in the paper [56]. The research task was to provide a clear understanding of why the A* planner is so widely used, covering 40 papers with each requiring to include pathfinding problems and compared to other planners. The main finding from the research demonstrated that many applications of the A* planner in practice are not the classical form of the algorithm. Most instances of the planner are modified for the specific task or environment, demonstrating the versatility of the A* planner. This draws parallels to Dijkstra, acting as good general starting points and allowing for development to specialise the planner for the required task. A practical example of the A* planner was shown in the paper [57], where a humanoid robot was tasked to navigate indoors, with the goal to be a human assistant. The research focused on having the robot plan and re-plan if moved manually using the A* algorithm. The A* planner used during the research was the classical method, with a Manhattan distance heuristic. This setup proved successful for this project, with the planning algorithm working successfully. A practical example of the A* planner can be seen in the paper [58], where a small two-wheel robot was tasked to navigate a warehouse. The A* planner that was used held relatively faithful to the classical method of the planner. The planner was tested in Matlab and on the robot, with both reaching their targets.

When comparing Dijkstra and A*, the paper [56] presented that the standard A* planner performs two thirds faster than Dijkstra because the planner is uninformed and blindly searching.

The paper [56], mentioned that the classic version of A* is not keeping up with the requirements of the robots being developed. This is seen in non-holonomic vehicles using these planners, as the vehicle cannot make discrete actions in all directions. Hybrid A* is a non-discrete version of the A* planner with kinematic constraints, limiting the search areas to what a vehicle will be able to perform. Therefore this can be seen as the A* planner but focused and refined for non-holonomic vehicles. This planner [59] demonstrates the real-world application of this planner in the urban DARPA challenge.

This planner made it possible for the vehicle "Junior" to navigate safely through urban streets, parking lots, and re-plan when blocked roads were found.

The sample-based planners covered in this review is Informed RRT* and its core predecessors. This planner builds off of RRT*, which is a more refined version of RRT. The RRT planner is still used in some cases, such as in 3D space seen in the paper [60], but due to it not leading towards an optimal path isn't suitable for making adequate 2D plans. The paper [61] presents a detailed analysis between multiple sample-based planners, including RRT and RRT*. This paper presented results showing that the RRT* planner all ways converged to an optimum solution while RRT on average was 1.5× if the RRT* cost. An interesting finding in the paper [61] was that the search area for connections should follow a radius of $k \times log(n)/n$ where 'n' is the number of samples and 'k' is a constant to scale the radius. This made the algorithm not asymptotically optimal but increased its computational efficiency while providing a potential near-optimal path. Another addition to RRT* is Informed RRT*, which shares all the characteristics of RRT* but incorporates a reducing search radius once the goal is found. This algorithm was presented in the paper [62]. The results found from this paper demonstrated a significant decrease in computational time to find near-optimal solutions with a target cost, variation in start to goal length, and the ability to find narrow a narrow path. This planner was tested purely through simulation for this paper and in a relatively open area.

These planners have demonstrated great potential through the literature review in this section. Dijkstra, A* and their respective variation covered in this section have proven to work well in set areas. These areas can be turned into a grid-like system, with vineyards sharing this capability proving that they could be used successfully in a vineyard. While Informed RRT* demonstrates a steady time to find near-optimal paths in small and large areas. Due to the literature not covering row-type environments, it will be useful to fill this research gap.

2.7.2 Local Planner

The TEB planner is a predecessor of another planner called "elastic band" (EB), which works to find the shortest distance to the global plan while avoiding obstacles. It does this by using "contraction forces" and "repulsive forces" to deform the plan so that the global plan is followed while avoiding obstacles [63]. This tracking and obstacle avoidance method was seen in the paper [54], where a small omnidirectional indoor vehicle was able to navigate through an area size of 100 m×40 m. This area included obstacles and people in narrow corridors.

The TEB planner adds a temporal element into the calculation for the optimal path while also taking into account the dynamic constraints of the robot using the planner [64]. This is important as the shortest path is not always the fastest, which is found even more so when the vehicle has restrictions on movement, such as a non-holonomic vehicle. The TEB planner is seen to plan effectively in both simulations and experiments with a real robot to navigate to way-points while avoiding all obstacles [64]. This paper does mention that this is still a local optimisation and does not guarantee global optima. The work from the previous paper was developed further in [65], aiming to make the TEB plans more accessible for "car-like" robots. The paper compared the TEB performance against Reed-Shepps planning in an obstacle-free environment. The outcome showed that the two methods coincided with each other, with both maintaining similar path lengths over varying minimal turning radii. The paper also presented experimentation of a "Lego EV3 robot", performing parallel parking and navigation through the obstacle-filled area. Both experiments demonstrated the planners' ability to avoid obstacles while producing adequate paths for a car to traverse, even with some of the parallel parking looking human-like.

A practical use of the TEB planner is seen in [66], where it was used to auto-park a small single-person vehicle in a car park for both "reverse-in" and parallel parking. The research presented informative results, with the first being that it was best to split the task into two stages. These stages are precisely what a human driver would do to get the same result. First is the setup to park, which involved driving up to the parking spot being

ready to reverse in and then the second stage was the implementation of parking itself. The results from the parking came from both simulation and real-world experimentation. Both presented high success rates, with reverse-in parking providing 96% for simulation and 90% for the real world, while parallel parking provided a 90% and 75% success rate. This paper also mentioned that the failures included failure to create a path, which would likely be solved through a re-planning action.

The research done on the TEB local planner shows excellent promise for car-like robots to navigate obstacle-filled environments. The research showed TEB's ability to navigate in tight areas but didn't present testing in a vineyard environment, which should be investigated further.

For a substantial amount of autonomous vehicles, a map of the area that will be operated in is required to do informed planning for navigation. This section covers different literature on mapping packages and will be analysed for operational performance and applicability to work in a vineyard environment. The main mapping methods reviewed in this section include 'Gmapping', 'Cartographer' and 'RTAB-Map'.

2.7.3 Mapping Packages

SLAM is the act of 'Simultaneously Localising And Mapping' and is the basis for the mapping methods being reviewed in this section. The first mapping package is 'Gmapping', which uses a 2D laser scan to create 2D occupancy maps. The algorithm for 'Gmapping' is based on the Rao-Blackwellized particle filters for grid maps [67]. This package is known as the default mapping package for ROS systems [68] and is mention to be one of the most used mapping solutions [67]. 'Cartographer' was made by Google [69], and uses point cloud data from a 3D LiDAR to create 2D occupancy maps. Unlike 'Gmapping' the 'Cartographer' package is unable to use a particle filter algorithm as the amount of data is to great, which in turn would create a significant increase in computational requirements. Instead the package breaks the mapping into two stages a global map and smaller submaps. The submaps allow for great localisation accuracy within that

area, and once a submap is complete it is used later on for loop closure to create the global map [69]. 'RTAB-Map' is covered in the paper [68], along side other mapping packages including the prior two in this section. This paper shows the versatility of 'RTAB-Map' in comparison to the other two packages by being able to uses all forms of LiDAR data, depth cameras and a combination of the two. From this data the package is capable of creating a 2D occupancy map and a 3D version. 'RTAB-Map' works similar to 'Cartographer' but uses different loop closure methods. These methods include attempting to match current image data to previously recorded scenes and using iterative closest point (ICP) for LiDAR data. The overall system of 'RTAB-Map' is covered in detail in the paper [68]. Results from this paper [68] are both qualitative and quantitative through displaying the 2D occupancy maps of real world data from the MIT Stata Centre and the use of a mathematical tool called 'absolute trajectory error' (ATE) [70]. The maps created by RTAB-Map through different sensors and configurations clearly show that the LiDAR-based 2D maps are much sharper and more precise. The quantitative section is presented through ATE calculations and compares the accuracy of the localisation of each mapping method, including 'Gmapping' and 'Cartographer'. 'RTAB-Map' outperformed the other two covered in this review when using LiDAR data, with one data set providing a max ATE of 0.05 m for 'RTAB-Map', 'Cartographer' with 0.11 m and 'Gmapping' with 0.19 m. These results were gathered so that 'RTAB-Map' used slightly different odometry, which was said to improve the other odometry used. This was suitable as it is part of the 'RTAB-Map' package, but the result from 'RTAB-Map' when using the same odometry presented a max ATE of 0.26 m, making it fall behind the other two.

In this review, the focus will be on the performance of each package while using LiDAR data. The paper [67] compared three SLAM packages, two of them being 'Gmapping' and 'Cartographer'. They were tasked to map an indoor environment for a rescue robot. The packages' environments were tested in were compact and maze-like and were created in simulation. The maps created by the two packages were almost identical in size, but this could be expected because the test area was only a 5×5 m area. The research took another mapping package to real-world testing called 'Hector-SLAM', which also performed similarly to the other packages. This package isn't being reviewed due to not

utilising any external odometry sources and solely relying on visual features for this information. This, in turn, will likely cause difficulties in low feature areas such as vineyards.

A comparison between multiple SLAM packages for mapping a vineyard was presented in the paper [40], where two of these packages used were 'Gmapping' and 'Cartographer'. The results were gathered by mapping five rows positioned side by side, with the length of the rows not mentioned but looking to be of substantial size. From this testing, 'Gmapping' was considered the best algorithm for the scenario due to its stability, even though the map may bend or curve slightly. The 'Cartographer' package was deemed the most accurate in mapping but would fail at times to generate the map making it less reliable.

The results from each of the papers proved the mapping capabilities of each package and each presenting potential or have already done so in mapping a vineyard. Further development can still be seen in these areas of research through solidifying results and pushing the software to work in different scenarios and environments.

2.8 Summary

Table 2.1 presents all the consistent sensors found through this review and their potential usages, with sensors seen in only one or two papers not being shown.

Table 2.1: Key sensors found during review.

Sensor	Object Detection	Localisation
LiDAR	✓	✓
RADAR	✓	
Camera	✓	✓
Ultrasonic	✓	
GPS		✓
IMU		✓
Wheel Encoders		✓

The purpose of this chapter is to investigate existing literature around autonomous vehicles. These findings identify gaps in current literature that determine the need for a transferable semi-autonomous architecture (SAA) for agricultural vehicles. Autonomous solutions for road, agriculture, factory tasks were reviewed to get a wide understanding of suitable sensors. This is further expanded to investigate the mapping and navigating methods that allow for the vehicles autonomous nature.

This chapter reviewed existing literature around autonomous solutions for road, agriculture, and factory tasks. It is clear from this review that a SAA would fill a currently underdeveloped area for autonomous vehicles in the agricultural sector. Many of the autonomous vehicles within agriculture are task specific and may not provide an indefinite focus on developing a SAA that is transferable to other vehicles. However, the literature provided a solid selection of appropriate sensors that can be used in the development of a SAA, as is presented in Table 2.1. These sensors were deemed to be suitable for a SAA, due to their prevalence in company and project platforms. To make full use of these sensors, adequate mapping and navigation solutions are required for the SAA. To construct a suitable navigation system for the architecture, a local planner and three global planners were reviewed. The local planner, 'TEB', demonstrated its ability to produce near optimal paths for a car-like robot, while minimizing the time and distance required to maintain the global plan. 'Dijkstra', 'A*', and 'RRT*' were found to be appropriate as solutions for many tasks, but often required adjustments to enhance its performance for a specific task. As such, these planners are suitable for SAA development, but may require adjustments to enhance their efficacy. A critical analysis of three mapping packages was undertaken, these were, 'Gmapping', 'Cartographer' and 'RTAB-Map'. The review of these packages focused solely on the ability to create maps using LiDAR data. However, 'RTAB-Map' was able to produce maps using not only LiDAR data, but camera data as well. This was shown to improve its performance when compared to the other two packages, but not by a large amount. Therefore, the SAA may benefit from the integration of 'RTAB-Map'. In the following chapter the results from the review were solidified by integrating them into the architecture. This presents the details of the integration process and what was required to make the architecture work with a vehicle.

Chapter 3

Integration of Semi-autonomous Architecture

This chapter presents the semi-autonomous architecture (SAA) for agricultural environments, encompassing physical and software components. The SAA fulfils specific criteria, such as the ability to localise, map an area, detect obstacles, plan a path, and have it able to adapt and move through the planned path safely. This was achieved by incorporating into the SAA different obstacle detection and localisation sensors, communication devices, and a warning system. A golf cart was chosen as the platform for the SAA to be integrated into, with its setup being focused on handling scenarios found when navigating a vineyard environment.

3.1 Semi-autonomous Architecture (SAA)

The novel SAA is capable of transforming any four-wheeled agricultural vehicle, allowing it to map and navigate autonomously through a corridor crop environment. With this ability, the SAA will be able to assist on many farms due to its modular nature. This was achieved by explicitly implementing sensors and electronics for a vineyard environment, as it was deemed a good representation of corridor-like crops. This was done by gathering information on what was expected to be seen in a vineyard environment and planning accordingly. The following section covers all hardware components of the SAA, which is shown in Figure 3.1 that allowed the research to be performed.

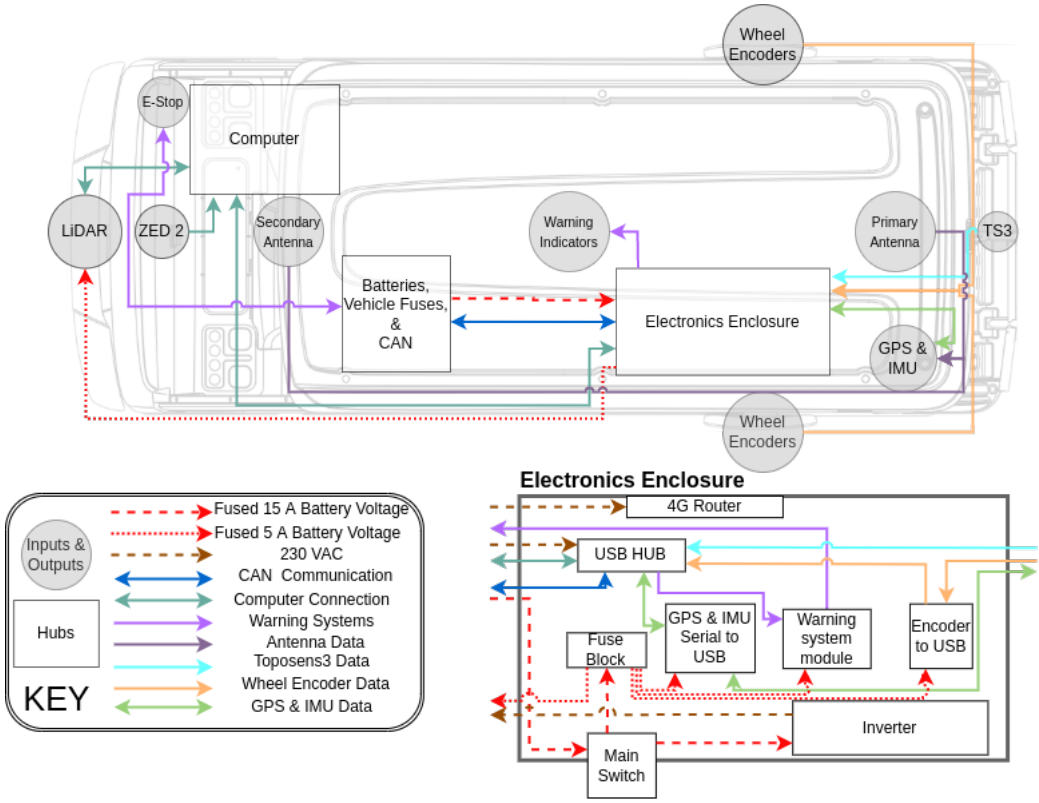


Figure 3.1: Top view of the hardware layout that creates the SAA.

The implemented version of the SAA hardware is presented in Figure 3.1, which provides a good understanding of the sensors and electronics positioning on the golf cart. Figure 3.1 also illustrates connections to allow for power and different types of communication between each section of the SAA. These connections are represented in the 'KEY', with power lines represented by dotted arrows, while data lines are solid. The arrows' direction determines how the power is distributed or how the components share data.

Table 3.1: Hardware used within the SAA.

Equipment	Power	Communication	Localisation	Mapping	Navigation
Computer	✓	✓	✓	✓	✓
3D LiDAR			✓	✓	✓
Stereo Camera					✓
Ultrasonic Sensor					✓
Encoders			✓	✓	✓
Dual GPS			✓	✓	✓
IMU			✓	✓	✓
4G Router		✓			
USB 3.0 Hub	✓	✓			
Micro Controllers		✓			
Inverter	✓				
USB to CAN		✓			
Warning Equipment		✓			

Table 3.1 presents the different electronics and their purposes in the SAA. The computer is currently used to supply power to the ZED2 camera through USB and handle the communication from all peripheral devices. The information gathered by the sensors allowed the computer to implement the localisation, mapping and navigation. The computer is supplied with environment information from three sensors, the LiDAR, stereo camera and ultrasonic sensor. The LiDAR is the primary sensor used in all tasks that the SAA needs to execute. In contrast, the other two sensors are used only for obstacle detection during navigation, as seen in Table 3.1. The SAA also incorporates wheel encoders, dual GPS (Global Positioning System), and an IMU (Inertial Measurement Unit) to provide localisation data on the vehicle. Because of this, they are used in all aspects of the SAA's tasks. This is due to the vehicle needing to know its pose at all times while mapping and navigating. An important factor of using dual GPS is that it eliminates the need to calibrate a magnetometer to provide a heading. This is important because if the architecture was transferred to a large vehicle, the motions required to calibrate

a magnetometer requires mobility [71] that may prove difficult and tedious for such a vehicle. The SAA incorporates components for communication and power sources for the modules within the architecture. The communication can be split into three sections; communication to the vehicle, sensors, and the SAA users. A 'USB to CAN' device was used to relay drive information to the vehicle, allowing for direct communication and control. In combination with communicating to the vehicle, the ability to receive information from multiple sensors was done through a powered USB hub. For the SAA to receive information from external sources, an internet connection was implemented through a 4G router. A simple warning system was set up to ensure the SAA is safe for workers to be around or on, allowing the SAA to communicate simply with people. The inverter was used to assist in powering the electronics by taking the 12 V battery voltage and providing a mains voltage, as seen in Table 3.1. This power supply was then used to power the 4G Router, computer charger, and USB Hub, which powered many other devices.

3.1.1 Obstacle Detection

Obstacle detection sensors that were used in the SAA include 3D LiDAR, stereo camera, and an ultrasonic sensor as seen in Table 3.1. These sensors provided the ability to assist in various problems that the AV faces, such as mapping, localising, and detecting hazards in the surrounding environment. The choice of the sensors depends on their purpose for the AV, whether to recognise features, detect near and far obstacles, or handle harsh environmental conditions. In most cases, one type of sensor is not adequate for detecting and understanding the world around the AV, requiring a combination to create a robust detection system.

The 3D LiDAR provides high accuracy, extensive range and FOV (Field Of View), making it an ideal primary sensor for the SAA to achieve all autonomous tasks. A stereo camera was used to assist object detection through computer vision systems and provide decent depth information. This sensor can provide more detailed information about an object than a LiDAR, allowing for greater object recognition and, therefore, more intelligent decision making. The stereo camera can also provide distance measurements,

which can be used to back up the LiDARs data and provide the capability to be used on its own as a source of primary information. Another object detection component within the SAA is an ultrasonic sensor tasked with assisting the other two sensors, providing a more robust source of information at close distances.

3.1.2 Localisation Hardware

The integration of a dual GPS (Global Positioning System), IMU (Inertial Measurement Unit), wheel encoders and a LiDAR was done to create a solid and stable localisation for the AV, as illustrated in Table 3.1. These sensors provide highly accurate pose data about the AV, which was paramount with this pose being the baseline for all further measurements. The GPS and LiDAR both provided an absolute pose (position and orientation), however, for different frames of reference. The GPS provides a pose relative to the starting position of the vehicle, while the LiDAR provides a pose relative to the map origin. In comparison, the wheel encoders and IMU provide fast but error-prone information that monitors the immediate vehicle state in terms of linear velocity, and linear and rotation acceleration.

3.2 Autonomous Platform

The SAA was implemented on a G30Es Yamaha golf cart [72] as demonstrated in Figure 3.2 to allow for the investigation of the SAA's performance. The golf cart was initially designed to transport players around the golf course using an inductive guided system [36] to follow a control line underneath the ground. The g30Es is 1.26 m wide and 3.1 m long, which was an ideal size for a vineyard, with row sizes varying from 1.5 - 3.05 m [73] [74]. This was verified by real-world testing done on a vineyard in Hastings, which had a row spacing of approximately 2.4 m.

Figure 3.2: Yamaha G30Es golf cart after modifications for autonomous driving.

The golf cart was designed to be an AGV (Autonomous Guided Vehicle) and was used as such while on the golf course if required. Because of this, it was equipped with a CAN bus (Controller Area Network) communication system for controlling and monitoring the vehicle's peripherals. The CAN bus allowed for an external computer to have access, which made the integration of the SAA simpler when compared to retrofitting electronics to the vehicle for steering and speed control [29]. The other peripherals included information of the vehicle's state, such as main battery voltage and vehicle speed, which were used for the warning system and other internal checks for the vehicle's controller. A custom frame was designed and built that allowed for sensor mounting and wiring, as shown in Figure 3.2. The frame was needed because the original golf cart did not include mounting positions for external sensors. With the features mentioned above, the golf carts presented a suitable platform for an AV that would operate in vineyard environments, with further investigation into the advantages and disadvantages of such a platform covered in the following sections.

Table 3.2 presents the exact electronics used to create the hardware component of the SAA.

Table 3.2: Core electronic equipment.

Equipment Type	Equipment Name	Quantity
Computer	Dell G3	1
3D LiDAR	VLP-16	1
Stereo Camera	ZED2	1
Ultrasonic Sensors	Toposens 3	1
Encoders	RE36	2
Dual GPS & IMU	INS-DU	1
GPS Antenna	BT-160	2
4G Router	RUT240	1
Powered USB 3.0 Hub	J5create USB	1
Micro Controller	Teensy 4.0	1
Micro Controller	Arduino Micro	1
Inverter	1200 W Inverter	1
USB to CAN	Korlan USB2CAN DB9	1
Warning Equipment	RGB Lights & Buzzer	1

3.2.1 Power Supply

Two lead-acid batteries provided power and control for the golf cart; a main 48 V battery for the DC drive motor and a secondary 12 V battery for steering control and internal systems. The stock 12 V auxiliary battery (50B24L) was changed to a higher power version (DIN75ZLMF) due to the increase in power drawn from it with the implemented SAA. This change was needed for any decent amount of testing, with the original only lasting 1-2 hours (depending on usage) before browning out when trying to steer. This is compared to the new battery, which would last for at least three times as long before requiring charging. In combination with replacing the battery, further investigation went to understanding the current being drawn by the steering motor. It was found that large current draws would occur while steering, and also, after steering, a large current draw would remain 5 to 10 A. The main issue for the battery was the constant current

after steering; this was fixed by developing a 'Vehicle_Monitor' ROS node to manage the steering engagement. Disengagement occurred when no motion was noticed after a timeout and re-engaged when a velocity topic was sent to the vehicle controller node.

3.2.2 Enclosure and Wiring

Most of the electronics (besides obstacle detection sensors) that were integrated into the SAA are contained within an electronics enclosure near the rear of the vehicle as shown in Figures 3.2 and 3.1. The enclosure had a clear lid that provided ease for visual inspections of the electronics, and an acrylic base plate was used to provide a custom layout of the electronics. This provided a central hub for devices to be connected and power to be distributed. For the safety of workers and the vehicle, every power line leaving a 'Hub' was fused (on the positive side). This was done using a 15 A blade fuse from the 12 V battery and 5 A fuses in a fuse block within the enclosure, which is presented in Figure 3.3.

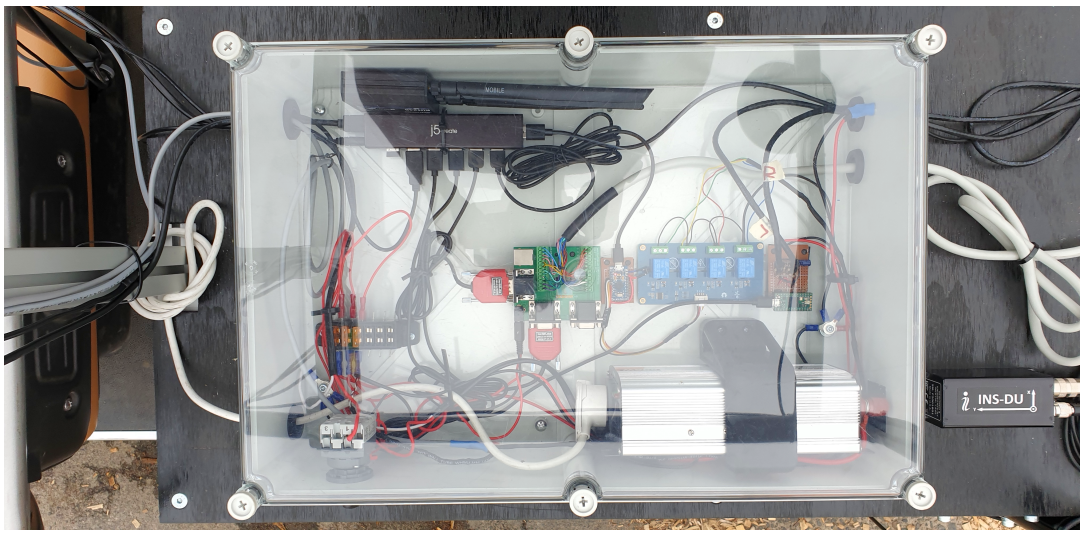


Figure 3.3: Top view of golf cart electronics enclosure.

3.2.3 Warning and Safety System

Communication was added to the SAA through a three-light and buzzer warning system controlled through an Arduino Mini and relay block using the 'Vehicle_Monitor' ROS node. This makes the vehicle a lot safer. It can let anyone around or operating the vehicle

know its current state, giving them time to respond to it before operation begins. This warning system can be seen at the top of the vehicle in Figure 3.2. This communication is vital as there can be teams of many persons and machines in an agricultural environment, making it paramount for communication between these team members for safe and efficient operation.

The current safety previsions for the golf cart include the stock physical key to turn the vehicle on and place it into autonomous mode, the vehicle brakes, and the front bumper which stops the vehicle if hit. Additionally, an emergency stop button was positioned in the developer's table, as seen in Figure 3.4, as well as a switch on the enclosure to enable and disable the new electronics system as displayed in Figure 3.3. The emergency button was wired just before the main fuse (on the vehicle's fuse block), which connects the power to the vehicle's control system. This completely disables all internal systems, stopping the vehicle immediately when triggered.

Figure 3.4: Developer station with emergency stop button within reach.

3.3 Sensors

All sensors can be grouped into two categories, active and passive. Active sensors emit something into the environment, such as sound waves or EM radiation and then measure the reflections to determine information about the object it came into contact with. Passive sensors detect natural emissions such as daylight, ambient sounds and gases.
The sensors can also be further grouped by what information they provide. For example,

a sensor that provides information about the robot's environment (e.g. surrounding obstacles) are exteroceptive, while sensors that monitor the internal state (e.g. battery voltage) of the robot are proprioceptive [75].

3.3.0.1 LiDAR

The primary obstacle detection sensor was the 3D LiDAR shown in Table 3.2. This sensor, as seen in Figure 3.5 provides a point cloud of data from 16 separate IR lasers [76]. Due to the vehicle's main direction of operation being forward, the LiDAR was placed at the front of the vehicle, centred horizontally, and finally positioned at 0.7 m from the ground after field trials.

The Velodyne sensor has its own ROS node and URDF making it simple to access data once it has been set up. The main difference between this sensor to others is that it is connected to the computer through an Ethernet connection and not USB, making its setup slightly more difficult than others. The setup process involved using the Ethernet connection to configure the LiDAR. However, most of the settings needed to be set in the LiDARs launch file as this also configured the LiDAR, with the main settings being the FOV (3.61 rads) and update speed (20 Hz).

Figure 3.5: Velodyne-16 LiDAR (VLP-16).

3.3.0.2 Stereo Camera

The camera used for the SAA was the 'ZED2' seen in Figure 3.6, which had pre-trained object detection models for humans, animals and vehicles. Understanding what obstacles are can greatly improve the level of autonomy for an AV by providing significant information that can be used to make better-educated decisions. Stereo cameras provide this capability in conjunction with the distance of the obstacle.

The camera was placed above the LiDAR on a ball mount, allowing for any angular adjustments during testing. The ZED2 had its own ROS node and URDF file, which allowed for a seamless integration into the system.

Figure 3.6: ZED2 stereo camera.

3.3.0.3 Ultrasound

The prior two sensors are the primary obstacle detection and recognition sensors, but both present operational limits in certain weather conditions. The ultrasonic sensors readings can become deteriorated from bad weather; nevertheless, it is more robust and able to supply a reading in such conditions. Therefore the ultrasonic sensors can provide close field coverage for constant obstacle detection even in non-ideal conditions [77]. The sensor used on the platform was the TS3 Toposens (prototype version, v1.0) seen in Figure 3.7 [78], which was unique from other sensors as it provided a point cloud of information. Using this sensor allowed the platform to detect obstacles and determine their rough position from the vehicle. The integration of one TS3 sensor was simple due to there being a ROS driver for the sensor made by the developers [79]. The main task of incorporating this sensor was adjusting the configurable parameters, which included the number of pulses, echo rejection threshold, peak detection window and noise indicator threshold. These parameters were adjusted for coarse object detection, as the sensor's prominent role is to assist in obstacle detection and not determine the exact shape of the object.

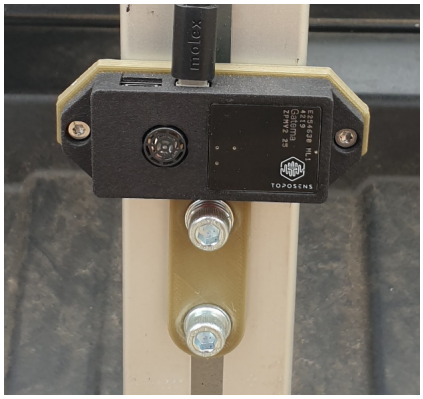


Figure 3.7: Prototype TS3 ultrasonic sensor.

3.3.0.4 Dual GPS & IMU

A dual GPS and an IMU were part of the same sensor module from "Inertial Labs", called INS-DU. This contains both exteroceptive (GPS) and proprioceptive (IMU) sensors, making it an important module for the vehicle to be autonomous. The primary role of the IMU was to monitor the motion of the golf cart through its accelerometer and gyroscope. At the same time, its magnetometer was not used due to the GPS providing the heading of the vehicle. The GPS works by having a primary antenna at the front of the vehicle and a secondary antenna at the rear, as illustrated in Figure 3.8. This allowed for a heading to be determined (while stationary) based on the position of each antenna. For best performance, the two antennas needed to be in line with each other, through the centre of the vehicle length-ways, and for the distance between to be as long as possible [80].

The INS-DU had the capabilities for RTK (Real-Time Kinematic) corrections, which was used throughout the testing of the vehicle. To get RTK corrections working, the system used a 4G router for internet access to receive corrections from the 'PositioNZ' network [81], which is a collection of GPS base stations around New Zealand [82]. To get this operating correctly, a third party software called 'str2str' was used to transfer the GPS corrections to the INS-DU unit. The 'str2str' package would convert the 'NTRIP' message from LINZ into serial data sent to the INS-DU to apply the corrections. The modules were placed near the rear axial with the orientation as specified by the manufacturers, which is depicted in Figure 3.8.

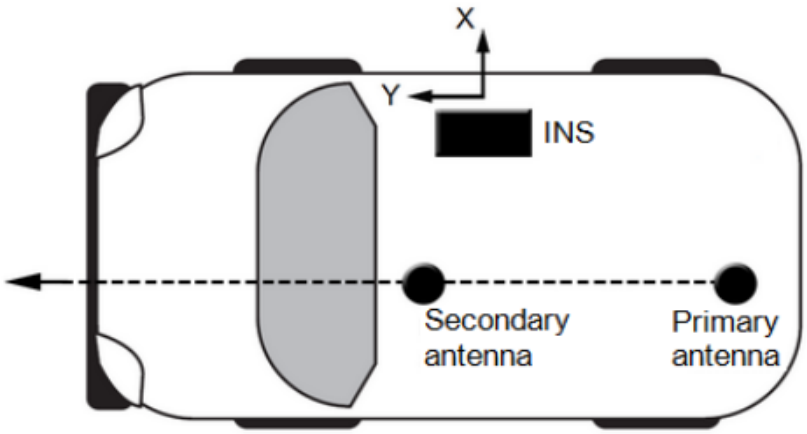


Figure 3.8: INS-DU layout recommendations.

[80]

The INS-DU came with software to help integrate the module into the vehicle. This worked by setting the position of the antenna distances from the IMU and noting if there was any deviation in antenna alignment. The software also allowed the user to define the output data, meaning only the data that was used by the system was being published, therefore allowing faster update rates. Changes to the provided ROS node were done so that the user defined data was sent correctly for other packages to use.

3.3.0.5 Wheel Encoders

The 'RE36' encoder was used to obtain accurate speed measurements of the vehicle. This information was paramount in understanding how far it had travelled, would travel and if the vehicle was going at the desired speed. This all leads to better control and safety of the vehicle during operation. The encoders were connected to the rear wheels using 3D printed mounts, as seen in Figure3.9.

Figure 3.9: Wheel encoder mounted.

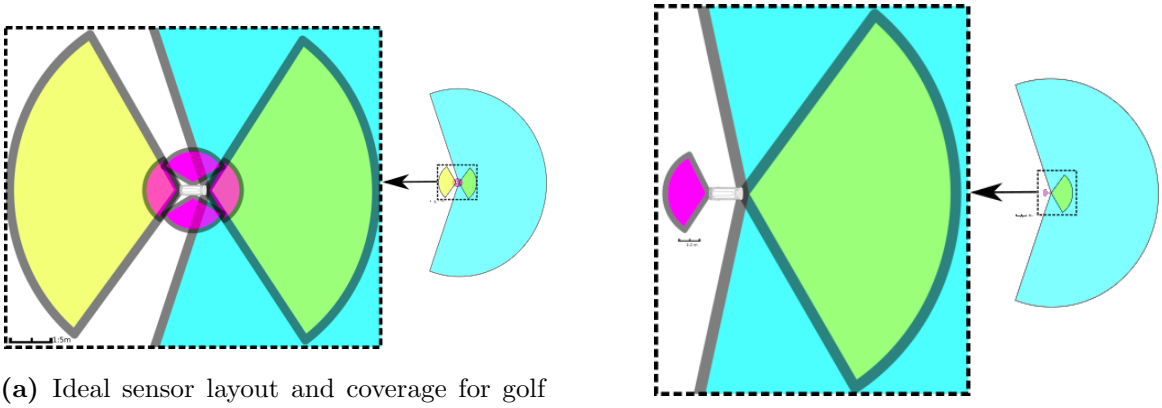
The wheel encoder's pulses were counted by a Teensy microcontroller and transmitted to the main computer. Before these signals reached the Teensy, they had to be stepped down to voltage levels the Teensy can handle and filter for any unwanted voltage spikes that could damage the input circuitry. The schematic for this filter circuitry is seen in the appendix as Figure A.1. Once the Teensy has counted the pulses and a minimum time has passed of at least 50 ms, a 1's complement checksum [83] is created from the data and is sent with the data as seen in Figure 3.10.

Start of Frame (SOF)	Packet Length	Left Wheel Pulses	Right Wheel Pulses	Update Period	1's Comp Checksum	End of Frame (EOF)

Figure 3.10: Wheel encoder data packet from Teensy 4.0, with each block being 32 bits.

3.3.1 Positioning of Sensors

The position of a sensor on the vehicle was determined by the sensor's characteristics and intended role for the vehicle. Figure 3.11a, demonstrates what the vehicle's operational



(a) Ideal sensor layout and coverage for golf carts.

(b) Current sensor layout and coverage for golf carts.

Figure 3.11: SAA sensor layouts based on the golf cart and the ODD from vineyard environments.

design domain (ODD) would be if the SAA were fully implemented. This was deemed a suitable sensor layout for a vehicle operating in a corridor crop environment. The vehicle should perform all tasks in most weather conditions using this configuration. By having the forward-facing LiDAR and stereo camera as primary sensors, backward-facing stereo camera as reversing obstacle and object detection and the ultrasonic sensors acting as redundancy and close obstacle detection. This final sensor layout would have hopefully provided an SAA that could perform at an autonomy level of 4 [15].

A minimalist approach was taken for development by including one of each sensor for testing purposes. This layout coverage is seen in Figure 3.11b. The use of the custom-built frame and adjustable MiniTec made it simple to move sensors around during testing but maintained the same layout seen in Figure 3.11b. The TS3 ultrasonic sensor was positioned at the vehicle's rear for any reverse obstacle detection. In contrast, the ZED2 and VLP-16 were positioned at the front due to this being the primary direction of travel.

3.4 Robotic Operating System - ROS

ROS provided the foundation for the software development for the SAA. ROS allowed for communication between sections, making it easy to integrate sensors and systems. Because it was open source and had a large community, it allowed for faster development.

The computer used Ubuntu 18.04 as an operating system (OS), which worked well with the ROS version, 'Melodic' [14], with it already having significant development and support. Overall, ROS provided the software framework necessary for developing the SAA, and its use remains prevalent throughout the rest of this chapter.

3.4.1 Communication

The communication of data between sensors and the computer was done using the ROS communication architecture through messages called 'topics' and is depicted in Figure 3.12. An exception to this was the wheel encoders which used a Teensy to record and send the pulse counts to the computer over serial communication in a predefined packet structure, which was presented in a prior section in Figure 3.10.

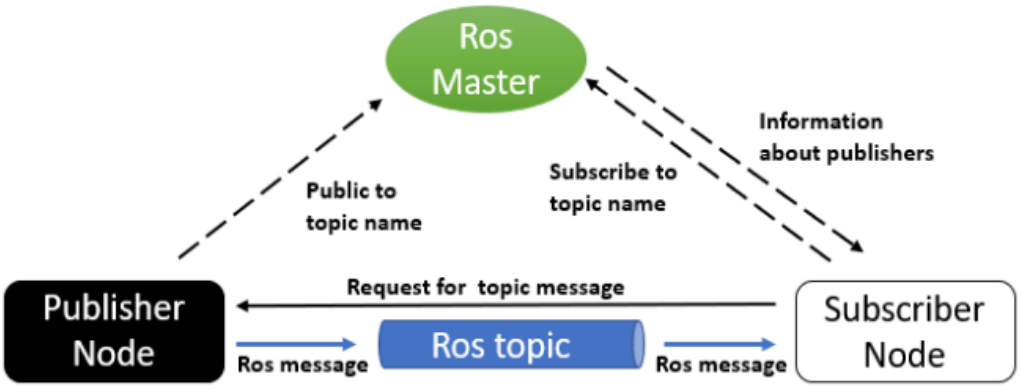


Figure 3.12: Overview of ROS commuication.

[84]

3.4.2 URDF

The URDF (Unified Robotic Description Format) was used to allow the computer to understand all the elements of the AV, such as the size, weight, and pose of onboard sensors. The URDF made for the golf cart, and SAA was close to the properties of the actual vehicle without overcomplicating the system. Tools that assisted in the development of the URDF involved 'RVIZ' and 'RQT' packages. They allowed for visualisation of the

URDF models, seen in Figure 3.13 and the transform tree (TF tree) of the vehicle, with one of the examples found in the appendix as Figure A.3.

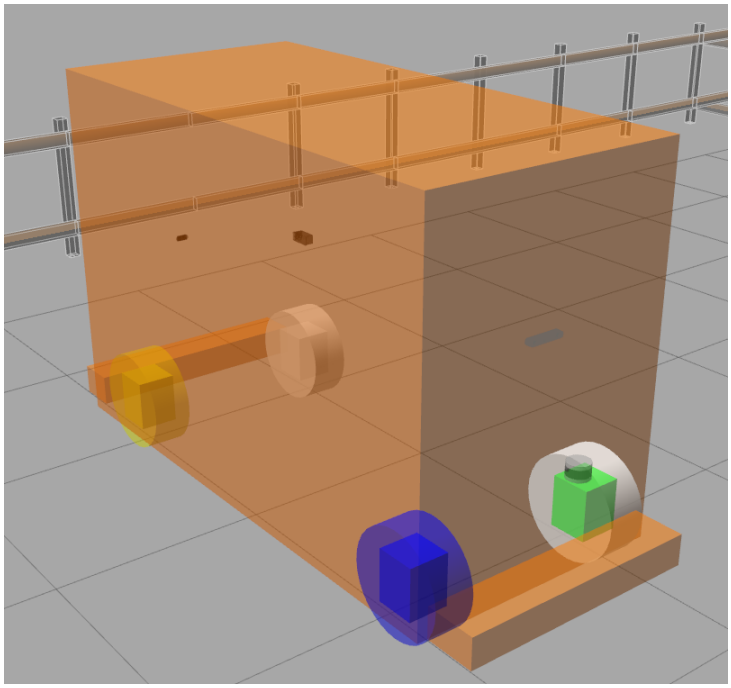


Figure 3.13: Visual representation of the URDF model.

3.4.3 Golf Cart Control

A driver that incorporated a ROS node was created by Autoware-AI called 'ymc'. This driver allowed for communication through the CAN bus system of the golf cart to control the steering and speed and provide information on the vehicle's state. The drivers communication is depicted in Figure 3.14, from [85].

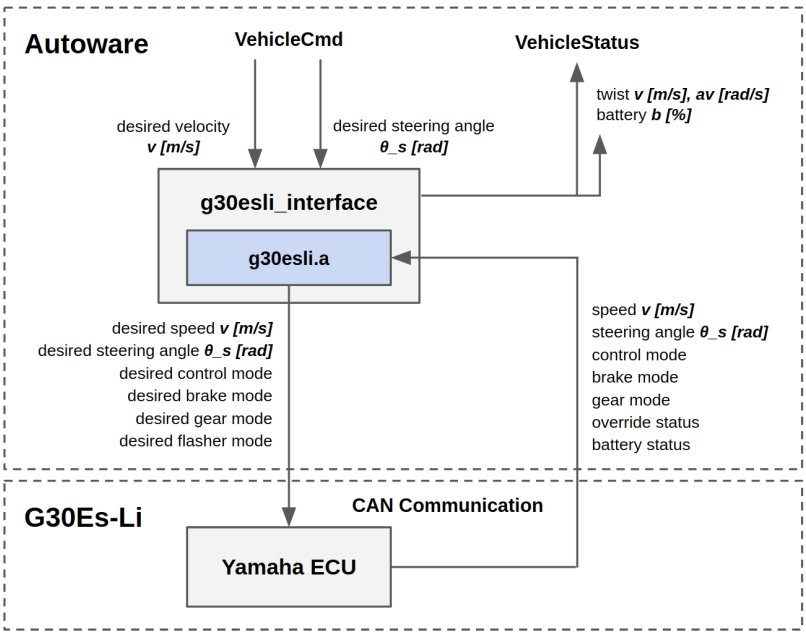


Figure 3.14: Autoware-AI 'ymc' driver.

The golf cart has its own internal system with unknown checks on the commands before executing. Because of this, a significant amount of time went into finding what actions the vehicle would deem suitable. The problems faced with using the golf cart meant a new version of the 'ymc' driver was created, using the one from Autoware-AI [85] as the basis but adapting the code to work better for the golf cart. Through this period of integration, the code was changed to include setting the minimum speed to 1.08 km/h (0.3 m/s) and an initial reverse speed of 3.6 km/h (1 m/s), which would then drop to the wanted speed as soon as the vehicle start moving. This was done because the vehicle had a minimum speed of 1 km/h to move forward and backwards, but when reversing, the start speed needed to be 3.6 km/h or higher for consistent operation. Once the vehicle's speed could be controlled, another change was implemented to allow for better control over direction changes. This change in code made the vehicle stop, wait for a set time (2 s) and then set the wanted speed and direction whenever the direction of travel was changed. The issue that triggered this change was the golf carts system that used two commands to change direction, one to stop the vehicle and another to tell it the speed and direction of travel. The vehicle characteristics of the golf cart would have caused issues while the vehicle was trying to navigate autonomously, as the other ROS nodes

that take control of the steering and speed are not aware of them.

Other changes to the driver code focused on steering and speed control. This was achieved using incremental control and not allowing any steering while the vehicle was stationary. This smoothed the power draw while steering and reduced steering during periods of large friction with the ground. The main reason for these changes was that the 40 A steering motor fuse would fail frequently. This occurred because the golf cart's steering control was very aggressive, in combination with the 'ymc' driver executing steering commands while the vehicle was stationary. The aggressive steering control caused a large current to be drawn (+40 A). This current would be held by the golf cart's steering controller, causing the fuse failure. The golf carts steering controller seemed to be using something similar to a Proportional controller (from PID). The current would drop by removing weight from the steering wheels, leading to the assumption that the steering motor was still trying to get to the correct position.
The integration of the golf cart into the autonomous system showed the challenges and limitations that come with using such a vehicle. Even though it was designed to follow line by itself if needed, it was not designed for constant steering and or reversing.

3.4.4 Mapping Architecture

Figure 3.15 depicts the general architecture for the mapping system of the SAA. This begins by collecting sensor data and pre-processing it so that it is better suited for the mapping package and or to make it more accurate. This then flows into the mapping package and finally the creation of the occupancy map through the 'map_saver' ROS node.

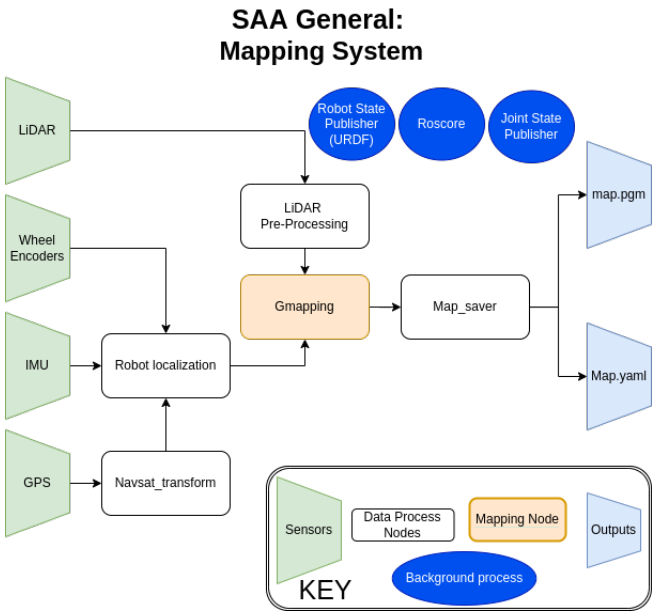


Figure 3.15: Flow diagram showing the general mapping system.

3.4.5 Pre-processing Data

Pre-processing of sensor data was developed to allow for better mapping and navigation. One version of this was LiDAR segmentation, which attempted to help the mapping and navigation process by filtering out the ground plane while still detecting a low lying obstacle. A ROS node (called 'localisation_helper') was created to provide the starting position of the vehicle within the map. This was done by getting the map's origin, the vehicle current pose and calculating where they are relative to a known point in the area, known as the datum. This was achieved by using the heading of the vehicle and maps origin for the difference in yaw, while the GPS points of each component were converted into UTM format to calculate the position. By doing this, the vehicle was provided with a good initial understanding of its localisation, allowing it to navigate with greater accuracy from the start.

3.4.6 Localisation Software

There are two different times localisation is used. First is the localisation needed while mapping the environment, and the second is while navigating a pre-mapped area. The

localisation used while mapping involves knowing the vehicle's absolute pose relative to the starting pose; localising within a map is understanding where the vehicle is currently within that map, using obstacles and prior knowledge to provide localisation. In both scenarios, the vehicle uses its localisation sensors to provide a solid understanding of its pose. This is not done by looking at the sensor data individually but using all of it to predict the best possible pose of the vehicle. The following section covers the software used to provide this localisation and its implementation for the AV.

3.4.6.1 Localisation Within a Map

The 'Adaptive Monte Carlo Localization' (AMCL) package [86][87] is the current method of localising within a mapped area. This package was chosen due to its support in the ROS community and it being prominent within existing literature [88], demonstrating its potential. The package only relies on odometry data and the use of laser scans, which is provided by VLP-16 after it is converted from a 'point_cloud' to a 'laser_scan'.

3.4.6.2 Navsat Transform

This package allowed the GPS sensor to be used in the 'Robot Localization package' by converting the latitude and longitude information into UTM (Universal Transverse Mercator) format, referenced to the starting pose of the vehicle. With this, the data from the GPS was passed to the EKF in meters relative to the starting position.

3.4.6.3 Robot 'Localisation' Package

The 'Robot Localization' package was integrated into the SAA, which allowed for all localisation sensors. This is possible as the package implements an Extended Kalman Filter (EKF). This provided the ability to receive multiple sensor outputs to predict a better estimation of the vehicle pose (position and orientation) compared to using a single sensor.

Two configurations of the package were created, one for mapping and another for navigating, as seen by the TF tree in Figure 3.16. The main reason for this was the

'Gmapping' package having to supply the map to odom link, forcing the 'Robot Localization' to work around it.

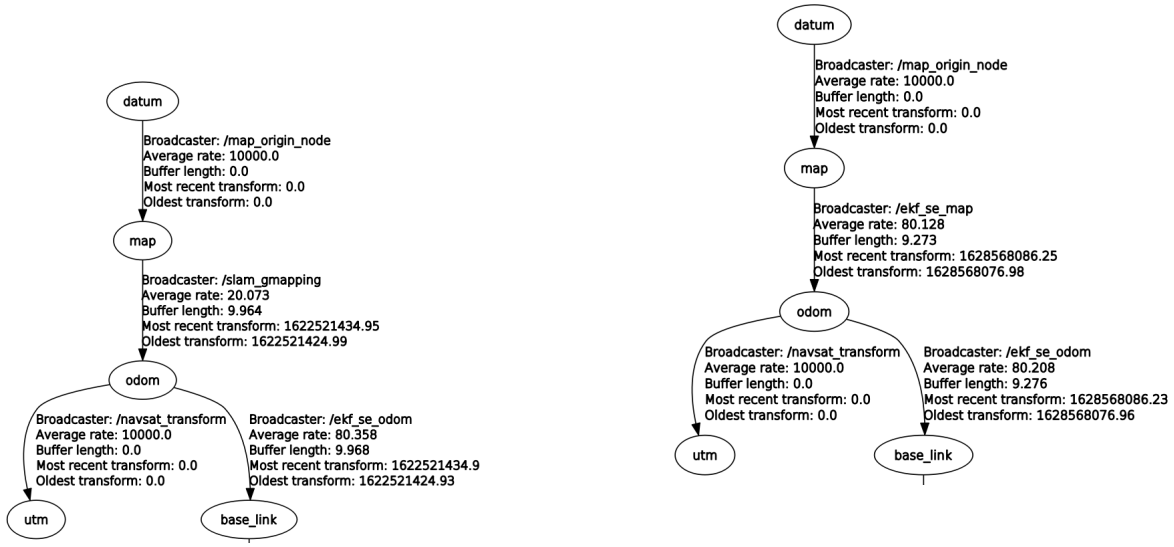


Figure 3.16: Top links for both the mapping and navigation configurations of the 'Robot localization package'.

The 'Robot Localization's' co-variance matrix was tuned to produce better predictions for the vehicle's pose. This involved adjusting these values until the package's odometry was steady and produced reliable results when compared to other sensors. The use of the documentation [89] provided great assistance in bringing this package into the SAA. General experimentation with the co-variance matrix through 'bagged' data brought a greater understanding of the package and fine-tuning of parameters.

3.4.7 Gmapping and RTAB-Map

Two mapping solutions were implemented, each with a core ROS mapping package. The first solution was centred around the 'Gmapping' package, which uses laser scan data while mapping. The second is 'RTAB-Map', which could use a range of sensors and could also be used to assist in navigation. The mapping process used was simultaneous localisation, and mapping (SLAM) [90], which bases obstacles positions of the vehicles pose and then re-uses these new references to localise and stitch the map together. Each package takes its approach to SLAM but holds the a similar concept.

3.4.7.1 Gmapping

The 'Gmapping' package was implemented due to it being used frequently in existing literature [40][67][91], where it has been shown to provide solid, and consistent maps. This package takes odometry information and a laser scan to create a 2D occupancy map, which can then be saved by the 'map_saver' node into a .pgm file and .yaml file. The YAML file holds information about the PGM file, which is an image file showing the mapped area. The flow diagram presented in Figure 3.17 demonstrates the packages used during mapping and how they interact with each other.

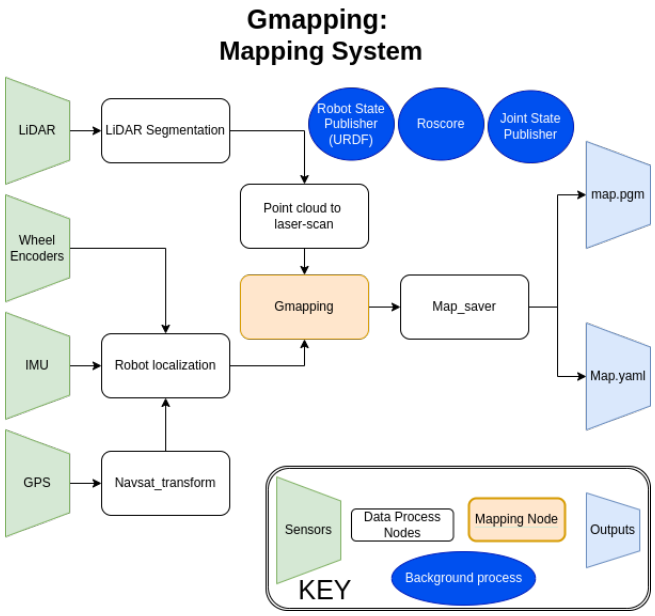


Figure 3.17: Flow diagram demonstrating the Gmapping system.

The flow of data during the mapping package through the use of arrows, going between the different inputs, packages, and outputs. A example of this is the VLP-16 point cloud being pre-processed into a laser scan, which allowed the mapping package to retain the depth information from all of the sensor data.

3.4.7.2 RTAB-Map

'RTAB-Map' is a ROS package [68] that was used during the development of the AV due to it providing a large number of possibilities in sensor usage while mapping. The package

is designed to primarily be used by camera sensors (Stereo and TOF). However, it also allowed for the use of 2D, and 3D LiDARs or even a combination of a LiDAR and camera [92].

The chosen sensor for mapping was the VLP-16 LiDAR, as it was the primary sensor, producing the most accurate data and providing a significant FOV and range. It was also found quickly that it would be unlikely to use the camera with RTAB-Map due to the limitations of the SAAs computer running out of usable memory to store all the data from the mapping process. The flow diagram presented in Figure 3.18 demonstrates the packages used during mapping and how they interact with each other.

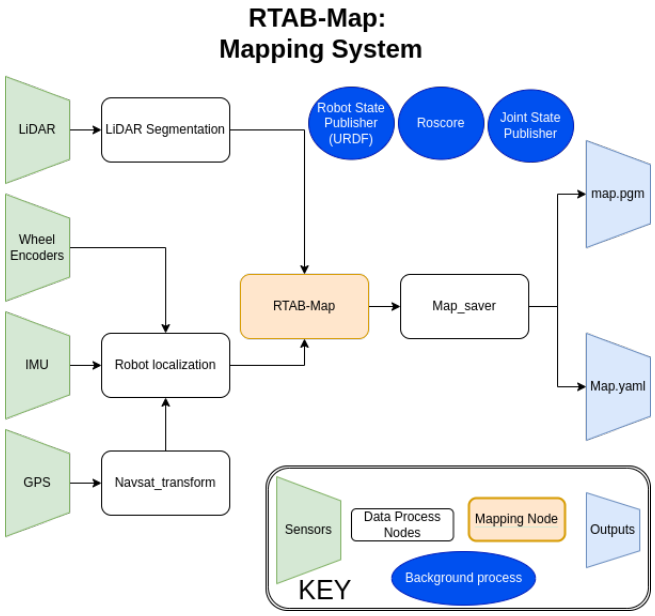


Figure 3.18: Flow diagram demonstrating the RTAB-Map system.

Figure 3.18 illustrates the mapping solution for 'RTAB-Map'. This diagram depicts the flow of data just like Figure 3.17 for 'Gmapping'. It should be noted that 'RTAB-Map' does produce its own mapping file that can later be used for navigation, but this was not used during development.

3.4.8 Navigation

Planning through and understanding the environment from prior knowledge and local information from sensors is paramount for any AV, with this all covered by the navigation

systems of the vehicle. ROS has a default template to base a navigation system on called the 'Navigation Stack', which provides an example of the used components to build such a system. The components that created the SAA's navigation system can be seen in Figure 3.19 and will be covered in the following sections.

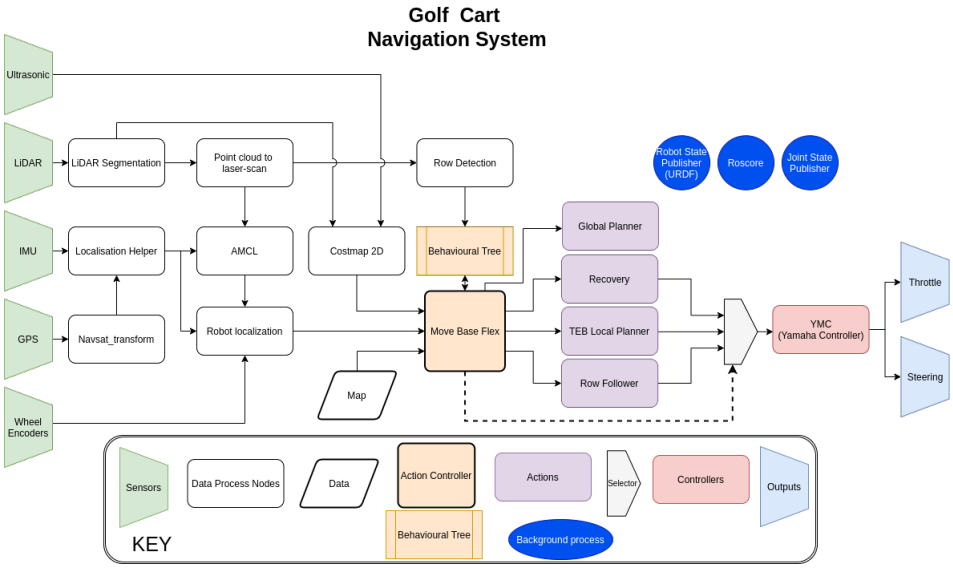
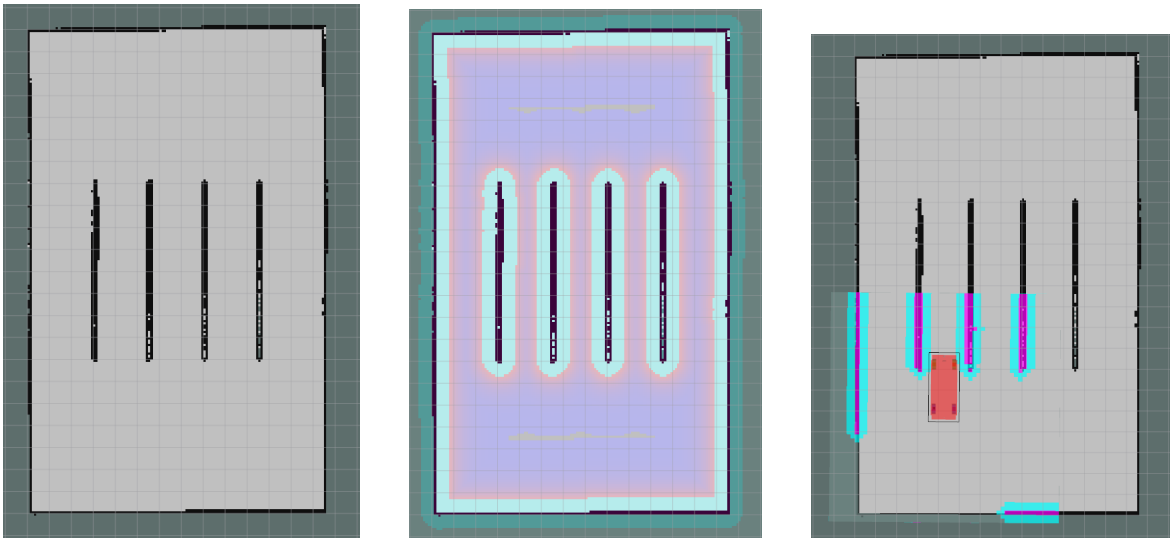


Figure 3.19: Flow diagram showing the navigation system for the AV.

Figure 3.19, demonstrates the flow of data as Figure 3.17 does, however, it is clear that the navigation process has more involved due to what the SAA has to undertake for safe and informed travel. Each node represents a component within the navigation system, and the colour and shape of the node provide further information into its role. An example of this is the 'Move Base Flex' node, which is the action centre of the system.

3.4.8.1 Costmaps

A costmap was used to inflate obstacles from a 2D occupancy map created from the environment. This is due to the environment not always being static and the need to create paths that are obstacle-free for a vehicle to travel. The package used to provide this feature is 'costmap_2d' [93], which uses a layer method to produce the new costmaps (global costmap and local costmap) based on the supplied occupancy map, seen in Figure 3.20. The layers of concern for costmaps are the static layer, inflation layer and obstacle layer. The default plugins from the costmap_2d package were used for the static



H **(a)** Occupancy Map. **(b)** Global costmap. **(c)** Local Costmap.

Figure 3.20: Map examples where black pixels represent obstacles (highest cost) and the colour transition from blue (high cost) to grey (free space) represents the change in cost value.

and inflation layers, while for the obstacle layer, a plugin was used from the package 'spatio_temporal_voxel_layer' (STVL). The static layer creates the global map from the received 2D occupancy map, while the obstacle layer detects any obstacles within a local area to create the local map. This is all then inflated through the inflation layer.

The costmap used for active obstacle avoidance is the local costmap, which monitors the area around the vehicle for localised planning. The local costmap used the same plugins as the global costmap but differed in parameter values due to their different purposes. The new STVL plugin allowed for a similar or better configuration of the input sensors compared to the default one and allowed for a time decay of the seen obstacles, which solved issues with the default plugin.

The local costmap size was restricted to an 11x11 m area centred at the 'base_link' (centre of the rear axial), with a resolution of 0.2 m as shown in Figure 3.20c. The limit in size was done because the local costmap has to handle a large amount of data from sensors. It was found that changes in size and resolution of this costmap would significantly affect the computation required to create the map and, therefore, would make the system not usable or fail if set not suitably. Because of this, the size was set so that it would function while still providing a usable map. An issue with the local costmap is that its orientation

does not change with the vehicle's orientation and that the centre was required to be placed at the vehicle's centre of rotation (centre of rear axial) due to the local planner using this point for navigation. A lot of monitored areas were commonly not used, and therefore wasted needed computational power and limited what could be done.

3.4.8.2 Global Planners

A global planner aims to produce an obstacle-free path from the start to the goal. The 'navigation stack' used two grid-based planners, Dijkstra [51], and A* [52], which are very similar to each other with both finding the optimal path but differing in how they search for this path. Both planners use the costmap to provide the travel cost ('G cost', $g(x)$) for each move away from the starting position. Each move is made by finding the next lowest cost cell and moving to it. Because this uses the costmap and changes to the costmap parameters can potentially affect how well the planers will operate.
The Dijkstra path planner was the simplest to use due to it not using a heuristic ('H Cost', $h(x)$) in its cost function to find the goal, and is defined by

$$f_d(x) = g(x) + c(x) \tag{3.1}$$

where as mentioned prior $g(x)$ is the travelled cost and $c(x)$ the cost of the next cell to be searched. Because of Dijkstra's lack of direction while searching, it will search every cell until the goal is reached. This greedy characteristic makes the algorithm potentially slower, especially as the map gets larger. The A* algorithm used is from the ROS 'navigation stack' and defined by

$$f_{A*}(x) = g(x) + d * neutral\_cost \tag{3.2}$$

where $g(x)$ is the travelled cost, d is the distance to the goal, and $'neutral\_cost'$ is a scalar for the distance. This placed a significant emphasis on tuning this cost function through the $'neutral\_cost'$ parameter, where large values will force the algorithm towards the goal, narrowing the search and vice versa for small values. Because of the $'neutral\_cost'$ parameter, it meant an incorrectly tuned cost function could be detrimental to the A*'s performance. As mentioned prior, the ROS planners relied on the costmap for 'G cost' values which was an unknown characteristic at first and caused non-ideal paths with jagged lines due to a strict costmap, and therefore the costmap had to be changed to

allow for smoother paths to be created. These planners also use a gradient descent based path smoother to enhance the drive-ability of the paths being created. A change made to the default A* algorithm (from the navigation stack) was also done; this involved how the 'H cost' was calculated. The initial way the 'H cost' was calculated was through the 'Manhattan distance' equation defined by

$$d = (|x_1 - x_2| + |y_1 - y_2|) \tag{3.3}$$

where d is the distance to the goal, $|x_1 - x_2|$ is the absolute horizontal distance and $|y_1 - y_2|$ is the vertical distance to the goal. This was changed to a 'Euclidean distance' equation defined by

$$d = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)} \tag{3.4}$$

where the variables have the same meaning as the 'Manhattan' equation and allowed the planner to plan paths diagonally.

The prior two-path planners would suffice for navigating within a crop environment. However, another path planner was developed for the testing purpose that takes a different approach to find the path. This path planner is the Informed RRT* (IRRT*), which is based on the work covered in the paper [62]. The cost function for IRRT* is defined by

$$f_{IRRT*}(x) = n(x) * d_p + g(x) \tag{3.5}$$

where the 'nodes' cost is $n(x)$ and is given by the costmap based on its non-discrete position. This is then multiplied by the distance taken to travel to it d_p. Like the other planners, the cost function also includes the cost of past nodes in the variable $g(x)$. The algorithm also works differently by re-assessing nodes within a neighbourhood zone to change links based on the most optimal path. This brings the algorithm closer to A* through having thoughtful adjustments to find the optimal path. The neighbourhood size calculation followed the algorithm 1, where the size was determined by the number of nodes that have been placed, and if the neighbourhood was too big or getting too small, it was capped at specific values based on the planner step size.

A few minor additions were made to the IRRT*, such as a configurable bias that allowed for faster growth in the direction of the goal. This was done by checking if a new node was ready to be placed based on the bias value. When a biased node is required, its

Algorithm 1 Bias-IRRT*: Neighbourhood Calculation

Require:

neighbourhood_constant $\leftarrow 45$

k = $\frac{log_{10}(Number\_of\_nodes)}{Number\_of\_nodes}$

neighbourhood_radius = neighbourhood_constant $\times \sqrt{k}$

if neighbourhood_radius = 0 **then**

neighbourhood_radius = neighbourhood_constant

else if *neighbourhood_radius* > (step_size + min_step_size) **then**

neighbourhood_radius = step_size + min_step_size

else if neighbourhood_radius $\leq$ min_step_size **then**

neighbourhood_radius = min_step_size

end if

position is set to the position of the goal forcing the planner's algorithm to create a node in the direction of the goal. The next addition included a Bezier curve path smoother for more drive-able plans, which was noted in the article [94] to be an efficient path smoother for mobile robots. Finally, the last addition implemented was a cost threshold. A node could not be placed within a grid cell if this cell's cost was above the threshold.

The IRRT* planner uses a random sampling-based search method compared to the grid-based search method like the other two planners. The potential benefit of this planner is that it can search large areas fast, and once the goal is found, it narrows down the search area allowing it to find the optimal path by re-assessing the current node positions.

3.4.8.3 Local Planner and Path Tracker

The local planner that was used was the 'teb_local_planner' (Time Elastic Band) [64][65], which attempted to create paths for a car-like vehicle that tracked the global plan while avoiding any obstacles. Using this package involved setting many parameters providing information about the vehicle, goal tolerance, trajectory, obstacle avoidance, optimisations for specific characteristics, parallel planning, and footprint model. The vehicle was represented by a straight line within the TEB package, using an obstacle avoidance parameter to set an area for the actual vehicle to be within and limit the distance to

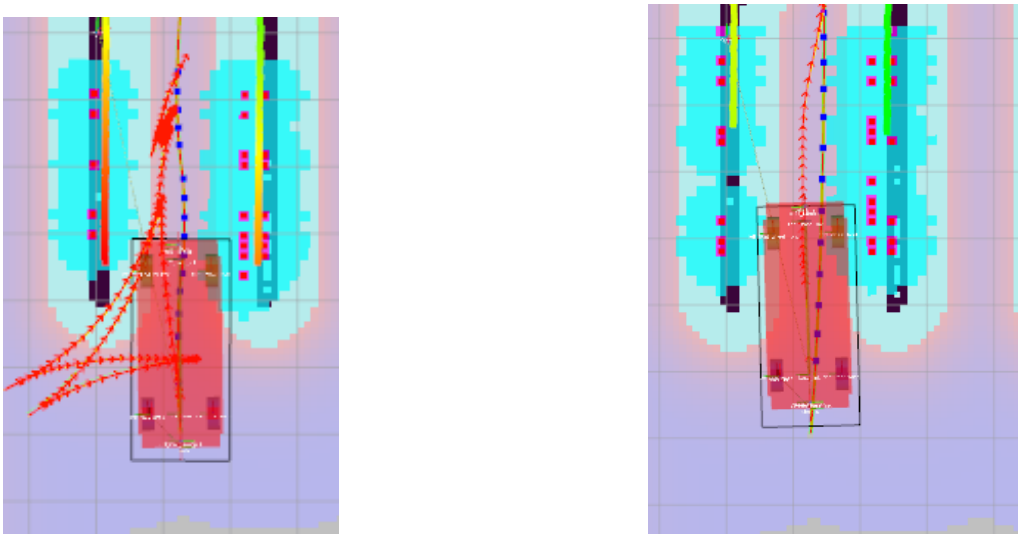
obstacles. Using this model placed less computation strain on the SAA, as it did not need to compute a lot of different geometries for the vehicle's shape. Other parameters were restricted due to computational power, such as the look-ahead distance at 15 m and the number of parallel plans that could be compared in operation, which was set to only 2 to avoid local minima. The look-ahead distance is relative to the base_link, which is positioned at the centre of the rear axial and is also restricted by the local costmap size. This made the maximum distance the planner could see in front of the vehicle approximately 8 m. The TEB local planner had many parameters to tune, which made the integration process tedious. This took a significant amount of time to tune and provide settings that would make the motions of the AV smooth and with reasonable intelligence while trying to limit the computational strain. Besides computational strain, the main difficulties were oscillations around the goal, oscillations while path tracking, large amount inefficient and illogical turns, and fasting changing of potential plans. The oscillations around the path were fixed by a combination of things, including making the 'ymc' steering controller smoother (mentioned in section 3.4.3) while also making sure the turning speed and acceleration were set appropriately based on the equation

$$\omega = \frac{\tan(\delta) \cdot V_x}{L_b} \tag{3.6}$$

where ω is the angular velocity, δ is steering angle, V_x is vehicle speed, and L_b is the wheelbase of the vehicle [95].

The TEB planner was allowed to reach the goal at a non-zero velocity within a goal tolerance of 0.33 m and a yaw tolerance of 0.1 radians. This was done to stop the oscillation around the goal, with the expected reason being the golf cart's inability for fine control due to its minimum speed characteristic.

The optimisation for the 'turning radius' was reduced, while the turning radius was increased from 3.5 to 4.5 radians to improve the inefficient turning. This was unexpected, however, it was mentioned in [64] that setting the optimisations larger can lead to greater accuracy in the vehicle navigation, but can also incur a much rougher trajectory by doing so. Reducing the optimisation for 'turn radius' meant the new path could be less constrained, and a steadier driving path could be generated, as seen in Figure 3.21.



(a) Minimum 'turning radius' optimisation set to 100/1000.

(b) Minimum 'turning radius' optimisation set to 20/1000.

Figure 3.21: Effects of minimum 'turning radius' optimisation.

3.4.8.4 Recovery Actions

The chosen recovery actions include resetting the costmaps using the 'clear_costmap_recovery', the ability to physically move away from any obstacles around the vehicle, which was done by the 'car_maneuver_recovery' plugin [96] and re-planning the global plan once the other actions are performed. The purpose of these actions is to free the vehicle when stuck while trying to plan or navigate.

3.4.8.5 Move Base and Move Base Flex

The action centre of the navigation system is a ROS package called 'move_base', which handles creating the global and local costmaps, calling the global planner, executing the local planner to follow this path, and triggering recovery actions if required. This package is the core component in making the vehicle autonomous. This is done by taking in localisation and obstacle information and ensuring the correct ROS packages receive this information. The integration process for 'move_base' was to understand what it is and make sure each package used by 'move_base' was configured correctly. The 'move_base' package was a great tool in managing many complicated parts of the navigation system. The control that 'move_base' had on the system made it simpler to integrate but limited

the flexibility of what the vehicle could do. The package 'move_base_flex' [97] was a direct replacement for 'move_base', allowing for control of the different navigation packages through action calls which is presented in Figure 3.22. The package also made it possible to use multiple recovery actions and planners if wanted, while also controlling the actions managed by a behaviour tree covered in a later section.

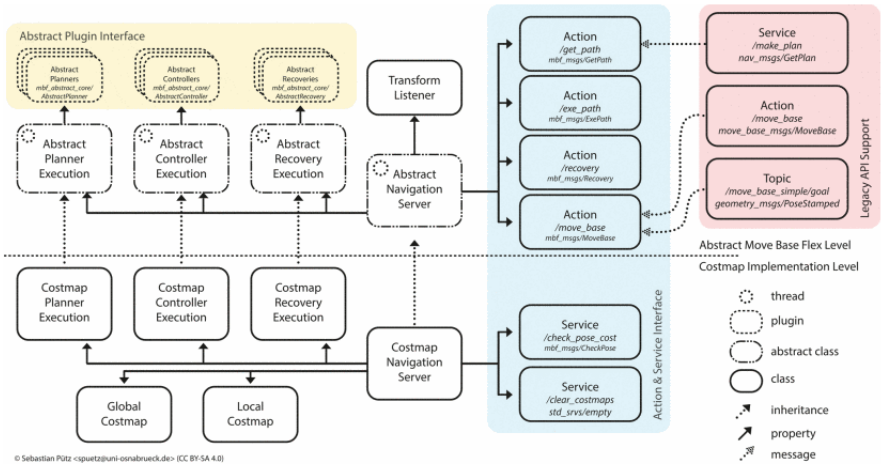


Figure 3.22: ROS move base flex navigation stack

[97]

3.4.8.6 Row Following

Two separate python scripts were created to allow the vehicle to travel down the row while maintaining a specific distance from a predetermined side of the row or to go directly down the centre. The first script was designed to be a row follower that used a PID loop to control the vehicle's steering, which maintained the vehicle's distance from the row. This controller used the LiDAR data as the source of distance information and is covered in algorithm 2. The second script was made to detect when the vehicle was in a row, triggering the row follower to start or finish. The development of the two scripts was relatively straightforward, with splitting the LiDARs information into five separate zones, left, front-left, front,front-right, and right. Each section is paired with another symmetrically, with each pair having a different size compared to the others based on their purpose. The front detection aims to detect any possible obstacles the vehicle may hit. This, in turn, will make the vehicle stop and wait for the object to be moved while

the sides are used to detect the rows for both scripts.

Algorithm 2 Row Follower Overview

Require: ▷ Set Constants

$MAX\_STEERING\_ANGLE \leftarrow 0.628$ rad

$WHEEL\_BASE \leftarrow 2.14$ m

$speed = 0.5$ m/s

$MAX\_V_z = \frac{(\tan(MAX\_STEERING\_ANGLE) * speed)}{WHEEL\_BASE}$

while Running **do**

get_lidar_readings() ▷ Averages and sections the LiDAR readings.

if $mode$ = Follow Centre **then** ▷ Check mode of operation.

$error = lidar\_left\_side$ - $lidar\_right\_side$

else if $mode$ = Follow Right **then**

$error = wanted\_distance$ - $lidar\_right\_side$

else

$error = lidar\_right\_side$ - $wanted\_distance$

end if

if $error \leq 0.1$ **then** ▷ Sum error when close to target.

$error\_sum$ += $error$

end if

Get Proportional: $p\_value = KP \times error$

Get Integral: $i\_value = KI \times error\_sum$

Get Differential: $d\_value = KD \times \Delta error$

$V_z = p\_value + i\_value + d\_value$

if $V_z > MAX\_V_z$ **then**

$V_z = MAX\_V_z$

end if

$Vavg_z$ = Average(V_z)

Send: $Vavg_z$

end while

3.4.9 Behaviour Controller

A behaviour controller package is used for greater control over the AV's actions. There were two types of controllers known at the time of development, a 'finite state machine' (FSM) and a behaviour tree (BT). The two packages that were chosen for testing and had the capability of implementing one of the frameworks were 'SMACH' (FSM) and 'PyTrees' (BT). Both used the ROS 'actionlib' framework to control what the AV did, which allowed for easy adjustments to code structure and a better understanding of the flow system control. The method used for the SAA was the behaviour tree, through the use of the 'PyTrees' package, due to its continued support and flexibility when compared to SMACH. An example of the behaviour tree designed for the SAA can be seen in Figure 3.23.

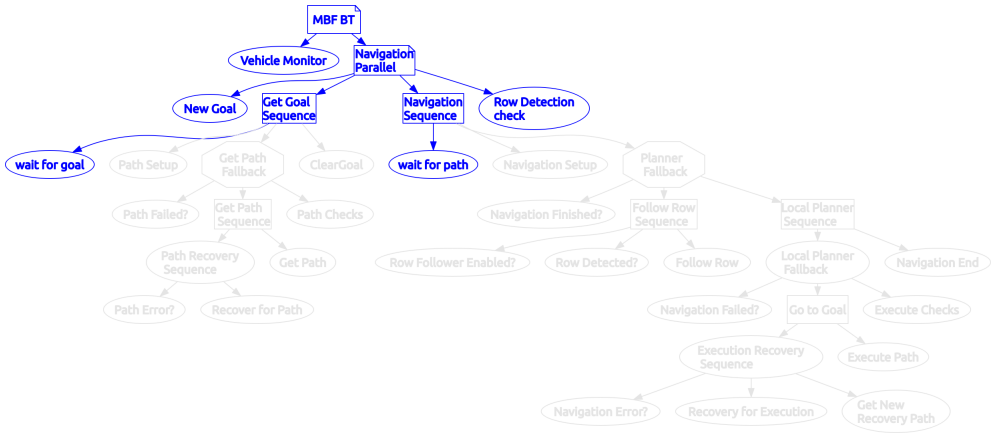


Figure 3.23: 'PyTrees' navigation behaviour tree.

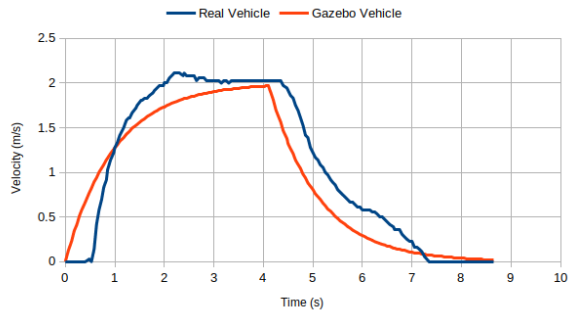
3.5 Simulation

The tool used was Gazebo, as it integrated well with ROS and used the URDF for simulation. The URDF was developed further, adding Gazebo plugins for vehicle control and simulation of sensor data.

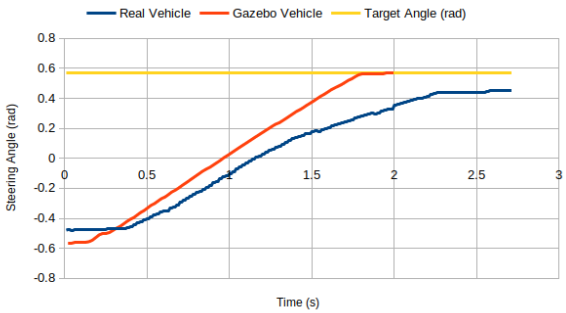
The vehicle control for the simulation needed to be as close as possible to the real-world vehicle for the testing to have importance. This was managed by a modified version of the 'Ackermansteer' plugin [98]. The plugin was changed to represent the golf cart with greater accuracy by including its unique characteristics; these were maximum steering

angle, maximum drive speed, minimum drive speed forward and backwards, and steering control and speed control like the 'ymc' driver mentioned in a prior section. After making these adjustments to the plugin, it was important to get the velocity and steering profile of the simulation as close to the real world vehicle as well. This was done by adjusting the provided PID and effort values in the plugin parameters and comparing the simulated steering and velocity profiles to what would be expected from the vehicle.

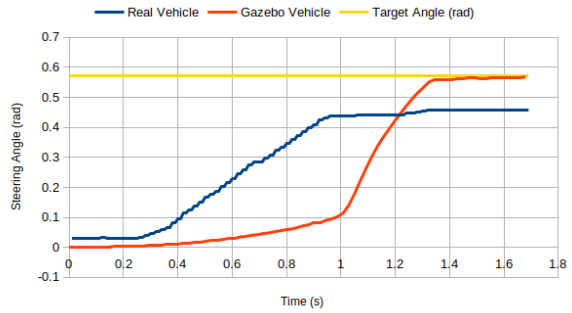
Figure 3.24 demonstrates how close the simulation is to the real-world vehicle.



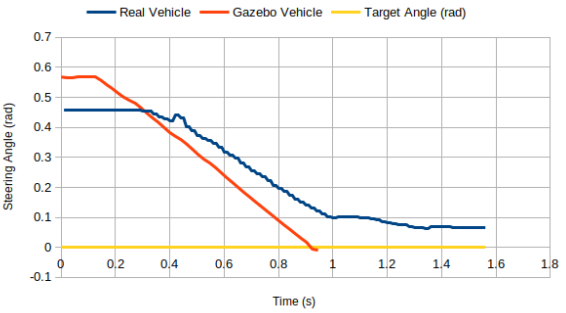
(a) Velocity comparison with target speed at 2 m/s.



(b) Steering adjustment anti-clockwise to clockwise.



(c) Steering adjustment straight to anti-clockwise.



(d) Steering adjustment anti-clockwise to straight.

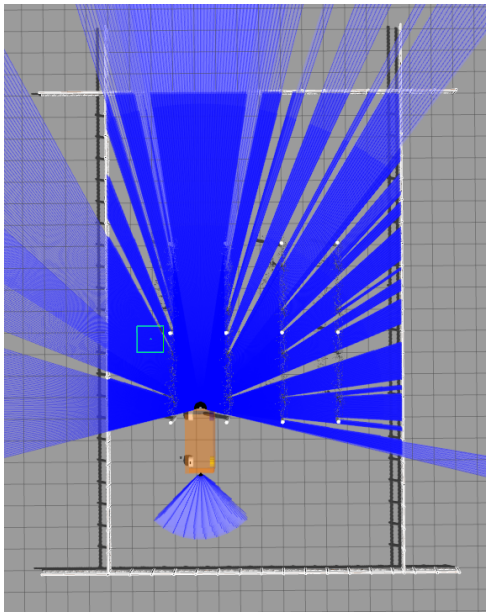
Figure 3.24: Real and Gazebo velocity and steering profiles.

An interesting note about Figure 3.24 is that the actual vehicle never reaches the correct steering position. This is likely due to the friction seen in the steering system and with the vehicle's controller tolerance.

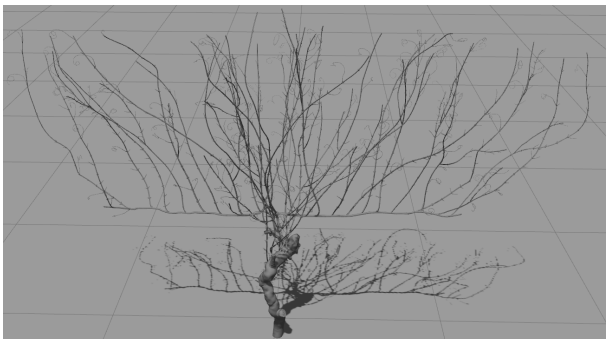
The representations of localisation sensors such as the GPS, IMU (INS-DU), wheel encoders were implemented in the simulation. The Gazebo plugins used to represent the INS-DU involved the 'libhector_gazebo_ros_gps' [99] for GPS and the default Gazebo IMU plugin 'libgazebo_ros_imu_sensor' [100]. The representation of the wheel encoders was done through the drive controller, which was a modified 'Ackermansteer' plugin [98] to

better represent the golf carts characteristics. The 'Robot Localization' package was used to provide better odometry by using all of the sensors covered in this chapter. For object detection, the 'Velodyne' package contained a Gazebo plugin for the VLP-16, which was also used to represent the TS3 sensor by adjusting the sensors parameters to match the sensor's datasheet specifications. Currently, the only sensor to not be simulated is the ZED2, which is not fully integrated into the SAA and is required to test the current autonomous driving framework.

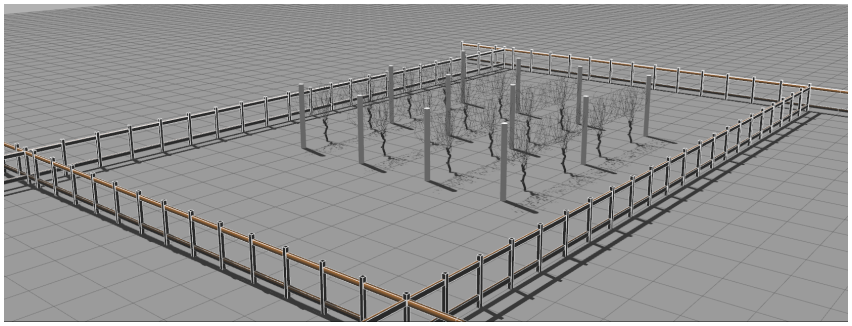
To provide valuable testing and experimental data, a simulation of a vineyard was created and designed to represent a real vineyard in New Zealand, Hasting, that was used for the first field trial. The vines used in the rows for the Gazebo world were 3D scanned from real vines, which The University of Canterbury created. The vineyard world was designed to be configurable (using xacro scripts), allowing for variations of every aspect within the vineyard. If a new vineyard needs to be simulated, this can easily be done by changing some parameters. An aspect of the simulation is the vine model's high quality. This quality places a greater strain on the computer to simulate the environment and therefore limits the size of the vineyard that could be created, with a Figure 3.25 demonstrating the viable size for simulation.



(a) URDF of the golf cart in the vineyard world.



(b) 3D scanned vine used in the vineyard world.



(c) Isometric view of the vineyard world.

Figure 3.25: Gazebo World

3.6 Summary

This chapter presented the intended and implemented SAA for a vehicle to map and navigate a vineyard environment.

The SAA was implemented on a golf cart, which was seen as a good vehicle due to its size and built-in control system through 'CAN'. During integration, it found that the vehicle incorporated a few non-ideal characteristics such as a minimum speed, an initial reverse speed, aggressive steering and the need for two commands to change direction. Most of these issues were overcome through the integration process besides the minimum

speed, which had potentially caused issues in the field trials and testing covered in the next chapter.

To ensure the platform was able to communicate with internal components of the SAA, the vehicle and people around it, different communication devices and a warning system were used. This made it possible to collect data from sensors but also control the vehicle and let workers understand the vehicle's current state.

The SAA consisted of an obstacle detection sensor layout targeted at the ODD expected from vineyard environments. The designed ODD layout was constructed from one 3D LiDAR, two stereo cameras, and four ultrasonic sensors. This amount of sensors was not implemented, and instead, one 3D LiDAR (Velodyne-16) and one ultrasonic sensor (Toposens TS3) was used for all of the testing, which provided front and back obstacle detection. The LiDAR was used as the primary sensor, facing forward for obstacle detection allowing it to assist in mapping, navigation and localisation, while the ultrasonic sensor faced backwards, letting it assist during navigation. This was deemed to be suitable for testing the sensors and SAA. The stereo camera was added to the golf cart but was unable to be integrated fully into the SAA.

A combination of specific localisation sensors was brought together to fit into the SAA, which allow the vehicle to understand its current state and where it is within the world. This group of sensors involved a Dual-GPS unit with RTK corrections, IMU and wheel encoders, providing highly accurate pose, rotational and velocity data about the vehicle. The hardware architecture provides the platform's ability to understand itself and the surrounding environment while also allowing for communication between SAA modules. The hardware was all brought together using the ROS network, and this combination made it possible to transform the golf cart into an AV. This ROS network consisted of sensors drivers, an Extended Kalman Filter, a behaviour manager, mapping solutions, costmaps, global planners, local planners, and finally, the vehicle driver.

Once a significant portion of the SAA was achieved on the golf cart, a model of the vehicle and a section of the vineyard from field trial (and other worlds) were created in Gazebo. This allowed for further development without the need for testing at the vineyard itself. In the following chapter, the implemented SAA is tested in the field and through simulation to provide informative results on the level of autonomy for the SAA.

Chapter 4

Semi-autonomous Architecture Evaluation

For a vineyard environment, specific criteria should be considered when designing an AV, such as mapping an area accurately, being able to travel safely through open and closed areas, the ability to make efficient decisions and maintaining a specified distance from a row. The previous chapter demonstrated the design and integration process for transforming the Yamaha golf cart into an autonomous platform with the SAA. In contrast, this chapter presents the setup, results, and expectations while testing the SAA through simulation and investigation in a field trial.

4.1 Experiment Overview

The experiments within this chapter were designed to test the main components of an AV while providing greater insight into the performance of the current SAA. The experiments were designed based on the findings from the field trials, with the outcomes from these trials highlighting missing components from prior development. The following experiments investigated the SAA's ability to map, plan paths, and evade obstacles in large and confined areas.

4.2 Field Trials

Testing the SAA in an agricultural environment allowed for a greater understanding of the challenges the vehicle may encounter. The first site was a blueberry orchard visited early on in the research. This provided a reasonable basis for the ODD of the SAA for row crop environments. The second site was in Hastings, New Zealand and incorporated tests of the SAA, with some of the data being reviewed within this chapter. To determine the SAA's current ability to map and navigate, the field trial involved mapping a vineyard area, rows and open spaces, then navigating these areas.

The section chosen from the vineyard for field tests was approximately 60 m long. This small section was easy to drive manually but caused issues while mapping and trying to autonomously navigate due to it not being flat and having overgrown grass through the row. This tall grass interfered with the LiDAR readings, even with LiDAR segmentation and therefore had to increase the height of the LiDAR and its detection height. While attempting to map the small section, an issue arose with the mapping package 'RTAB-Map', causing a rapid change to a different solution using the 'Gmapping' package, which worked well.

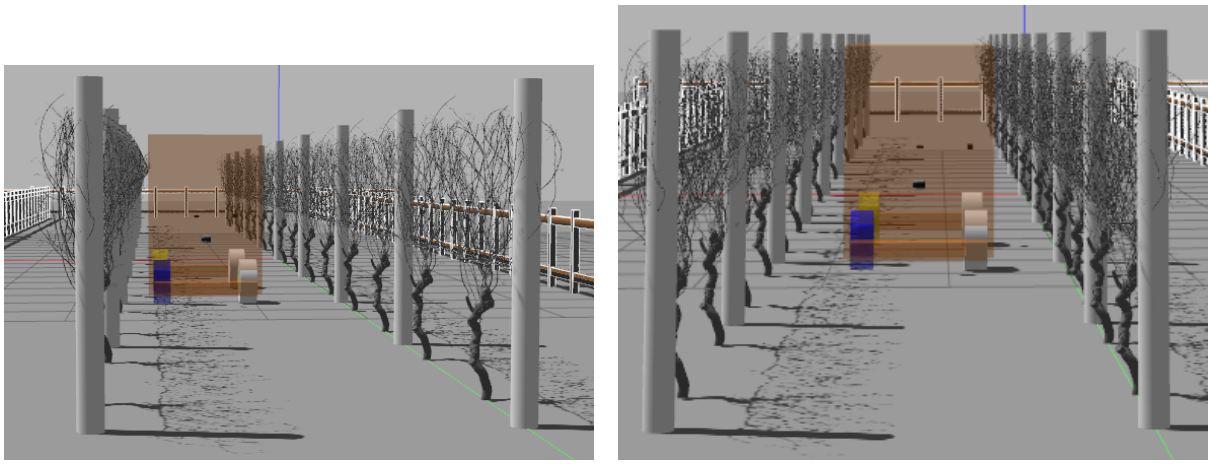
With the mapping solution in place, the vehicle attempted to navigate down the row. The navigation failed, with the vehicle constantly adjusted as it tried to follow the plan. Because of the issues on this small and far from ideal vineyard section, testing was moved to the larger section (approximately 375 m long), which was much flatter, and the grass was not as overgrown. This section produced better results, with the golf cart able to navigate partially (approximately 10 m) down the row without getting stuck. The initial thoughts were that the golf cart characteristics of a minimum speed and delayed braking caused the navigation issues on the small section. At the same time, the small hill added to the local planner's difficulty to control the golf cart. This is still believed to be an issue for the golf cart when it will have to navigate a sloped row and will need to be tested again. Another issue found after field testing in the simulation was that the costmap parameters were too aggressive, making the paths taken more course and harder for the local planner to follow while in the row.

During field testing, the SAA was tasked to autonomously map and navigate in a large

open area which it was able to do successfully. A limitation of the SAA's obstacle detection was found during this testing, with it being unable to detect a small non-permanent electric fence. This inability to detect such fences lines would make it think this space can be traversed and, in turn, cause it to crash, likely leading to the halt of the navigation, destruction of the fence and damage to the AV. The use of sensors such as RADAR could provide the necessary detection for such barriers to be detected.

4.3 Maintaining Row Distance

A "row_follower" script was developed to maintain a certain distance from a particular side of the row or to track the centre of the row using a PID controller, as seen in Figure 4.1. The input was based on the LiDAR readings from the side of the vehicle, and the output of the system was the steering angle, while the vehicle's speed was kept at 0.5 m/s (1.8 km/h), controlling the steering angle.



(a) Experiment ran with the target position set to 0.2 m from the side of the vehicle.

(b) Experiment ran with the target position set to centre of the row.

Figure 4.1: The PID controllers ability to correct for large deviations from the target distance.

The setup for the experiment involved creating a 32 m long and 2.5 m wide row (from the centre of one vine to the centre of the other) in Gazebo as seen in Figure 4.2. This environment was mapped using the 'Gmapping' solution and used a planner that creates only a straight line plan to begin navigation. The start position of the vehicle was just inside the beginning of the row so that the 'row_follower' would execute straight away.

The controller was made to track both '0.2 m' and '0.8 m' from the right side of the row and compared this method to a slightly different one for tracking the centre of the row. The methods used to track the centre line involved using only one side (right) by setting the distance to track half of the row width, while the other method attempted to balance the left and right sides.

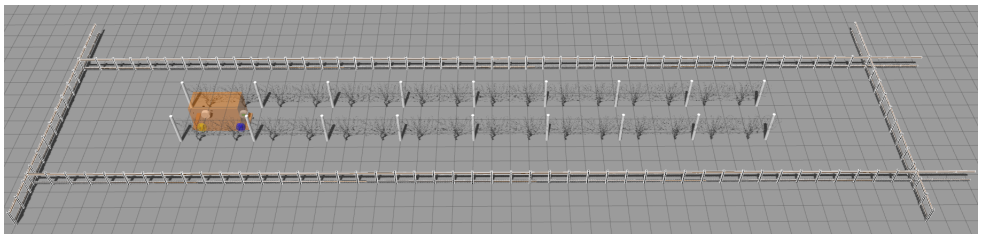
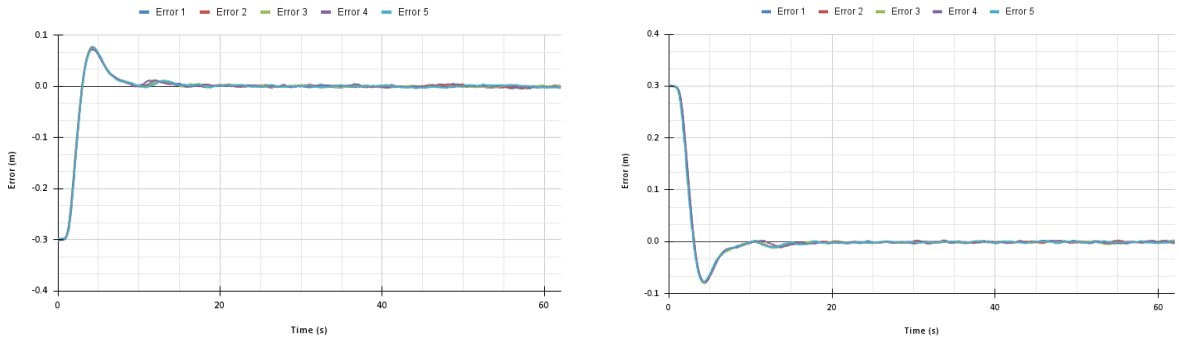


Figure 4.2: Single row Gazebo world that was used for testing the 'row_follower' script.

The experiment was done to determine the controller's accuracy, stability, and reactions to track specific distances and whether it could be deemed adequate for use in a vineyard environment. Each experiment was done five times to get averages of the final results and see how the controller would vary over multiple runs.

Potential difficulties with the controller include; sampling a potentially sparse object (vine), what size of the LiDAR's FOV is used for sampling, and for the vehicle to fix the error, it has to change its heading and therefore what the LiDAR sees. These factors can cause unwanted behaviour even if they have attempted to be managed in code.
The start of the experiments for '0.2 m' and '0.8 m' were used to test the controller's reactions to correct for a large error from the wanted distance, as seen in Figure 4.3. The choice of '0.2 m' and '0.8 m' as the limits of distances for the experiment was due to issues arising from the initial correction and the navigation down the row. When using closer distances (0.1 m or 0.9 m), the vehicle would often collide with the vine during the correction, or if able to correct, it would collide when driving down the row. Therefore the choice of '0.2 m' and '0.8 m' seemed to be suitable for testing and were able to demonstrate the controller's ability.



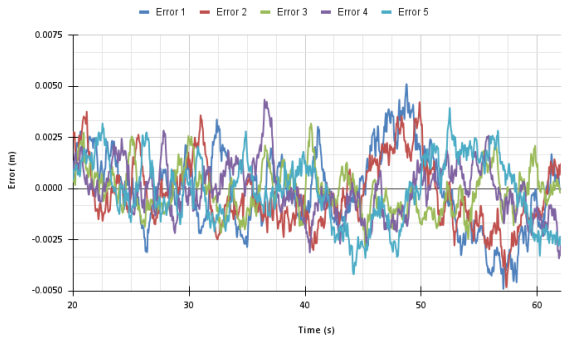
(a) Starting in the row, with the target distance is set to 0.2 m from the side of the vehicle.

(b) Starting in the row, with the target distance is set to 0.8 m from the side of the vehicle.

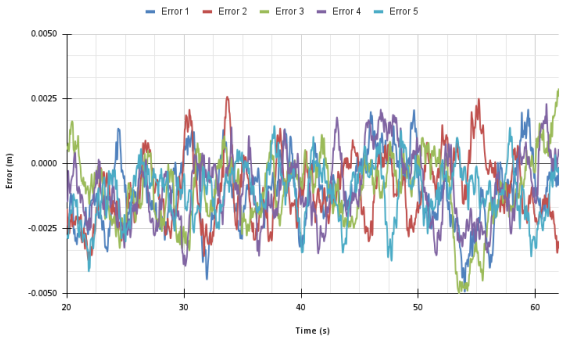
Figure 4.3: The PID controllers ability to correct for large deviations from the target distance.

The response seen in the experiment looks to be of adequate speed getting to the target distance consistently around five seconds and stabilising around ten seconds, with minor overshoot and little to no oscillation from the significant correction.

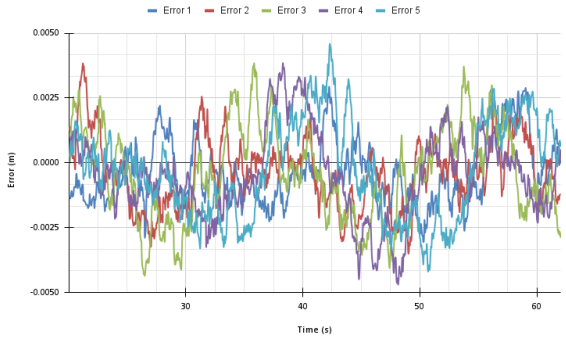
The experiments fully stabilised after approximately 20 s as seen in Figure 4.4, a proper analysis could now be done on how well the ”row_follower” controller can track a certain distance.



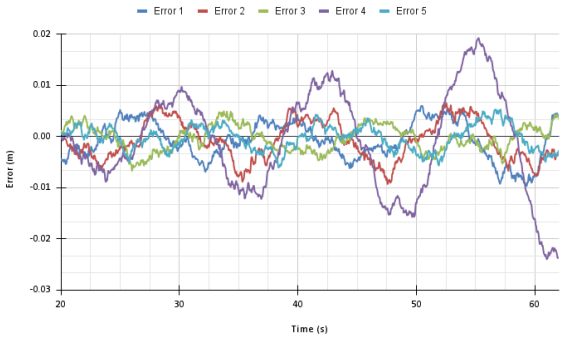
(a) After 20 s of following the row at a distance of 0.2 m.



(b) After 20 s of following the row at a distance of 0.8 m.



(c) After 20 s of following the row at a distance of 0.5 m.

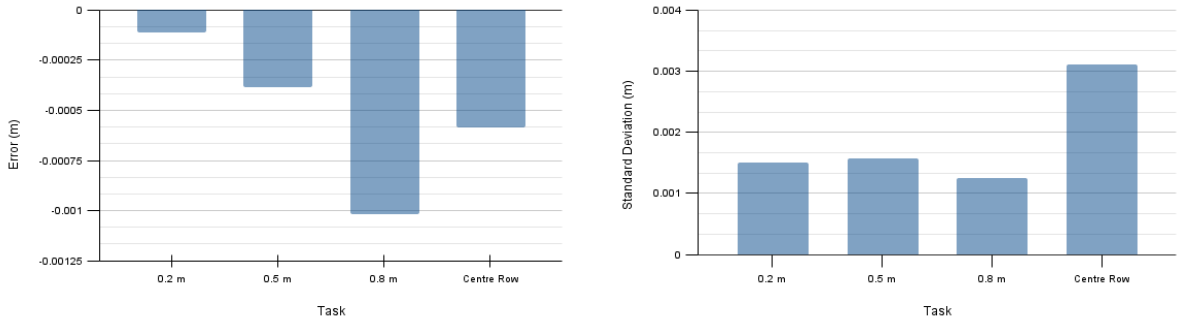


(d) After 20 s of following the centre of the row using two-side monitoring.

Figure 4.4: Stable period of the four different experiments, with each showing five different runs.

Figure 4.4 demonstrates the accuracy of the controller by keeping the error within 5 mm and with the majority of the time being less than 2.5 mm. There is a possible reason for the higher performance of the controller; the first is that the Velodyne Gazebo plugin had a noise value of 8 mm, which is less than the expected accuracy of ±30 mm. This is unlikely to be an essential factor due to the averaging done on the LiDAR data, negating many fluctuations in the readings.

After further analysis with the same data, Figure 4.5 presents the core information about each experiment.



(a) Mean error for each task.

(b) The standard deviation for each task.

Figure 4.5: Effects of the method used and distance from the vine on the expected error during row following.

When comparing the controller methods for following the centre of the row, the difference between the two mean errors is 0.3 mm or 56% in favour of only monitoring one side. The expectation was for the method that monitors both sides to perform better, but this information alone is not significant enough to make claims.

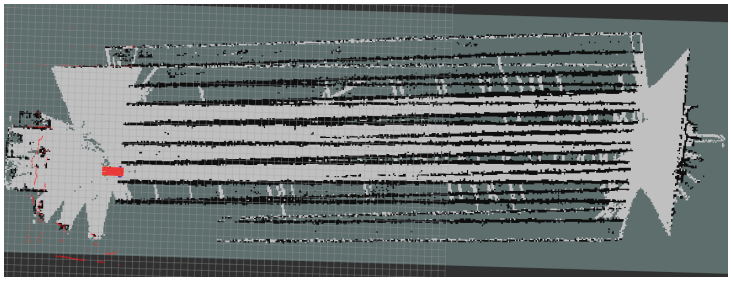
The data presented in Figure 4.5a demonstrates that the greater the distance from the vine, the larger the average error was. The change in distance of 0.6 m seems negligible to a sensor that can measure up to 100 m, but this accounts for an increase in beam width of 0.33 m (from 0.44 - 0.77 m). The increase in mean error from '0.2 m' to '0.8 m' was nearly 800%, but this only equates to 0.9 mm, which is minuscule compared to the system as a whole, with the largest mean error being 1 mm. Therefore looking at the spread of data may present more useful information on the effects of distance and or method of control. When looking at the standard deviation, it presents that the thought of an increase in beam width creating a greater variation in depth measurements is not present when looking at Figure 4.5b. The standard deviations demonstrated that the distance seems to have little to no effect on the distances used for navigating down the row. In contrast, it seems to present significant deviation when comparing the two control methods. When looking at this result, it seems that it is due to the PID system being tuned for tracking one side. When the 'two-side monitoring' is used, the error is effectively doubled by adding the errors from both sides. This increase in error associated with the two-side monitoring could be the reason for the larger and more drawn out oscillations

around the zero point when compared to the '0.5 m' experiment as seen in Figure 4.4d. It is also noticeable that the two-side monitoring 'Error 4' in Figure 4.4d demonstrate an unstable control system, with each peak getting exponentially larger. When removing the 'Error 4' data set from the prior calculations, the mean error dropped by 16% to 0.59 mm, while the standard deviation dropped 43% to 3 mm. This brought this method slightly closer but was not enough to compete with the other method for the current tuning of the PID controller.

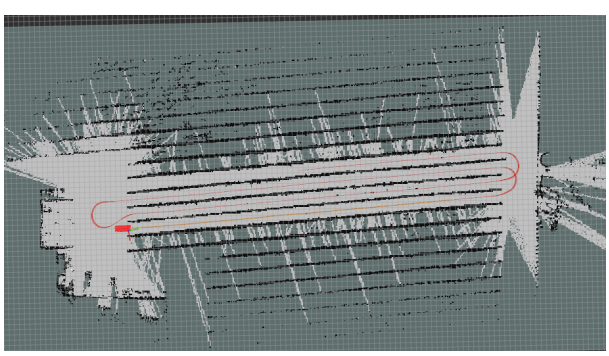
Comparing the centre following results from this experiment to other research such as [21], demonstrates the potential of the PID controller, producing 10% of the error compared to the paper. It should be noted that this can not be a direct comparison due to the row types being different and their research being done in a real-world experiment, but it can still provide a small validation for the results. This all leads to the next step, which is to test the PID controller in a real-world environment. While a physical test would solidify these results to fully determine the performance of this controller, it still produced informative results on the use of such a controller for the SAA.

4.4 Mapping an Environment

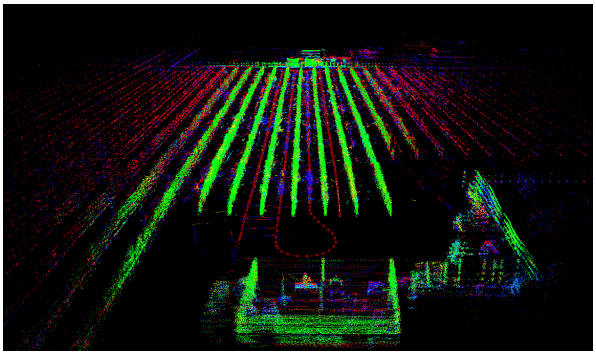
Mapping an environment accurately is significant to having safe autonomous navigation later on. The main packages used through development and integration were 'Gmapping' and 'RTAB-Map', with the 'Gmapping' solution being used for final testing. Therefore the purpose of this experiment is to compare the two packages to determine their suitability for a vineyard environment and to further test the 'Gmapping' package's ability to map large areas. The 'Gmapping' package is tested further because it is the final choice for SAA because of its consistency during development. The speeds used for this experiment ranged from 0.5 m/s to 2.0 m/s, which produced consistent results. Below are the tests performed on each package and the developments found through the research.



(a) RTAB-Map failed map, using system setup from field testing.



(b) RTAB-Map successful map, with the architecture setup used at the end of the project (latest).

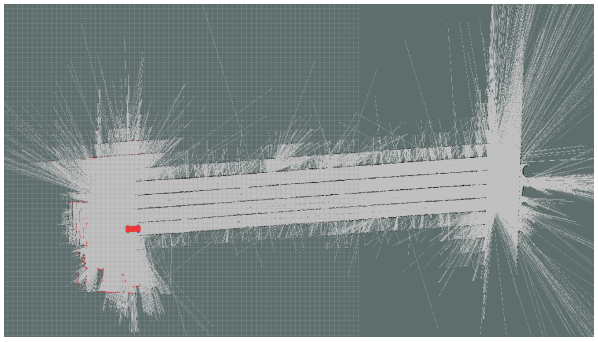


(c) RTAB-Map successful map using 3D point clouds.

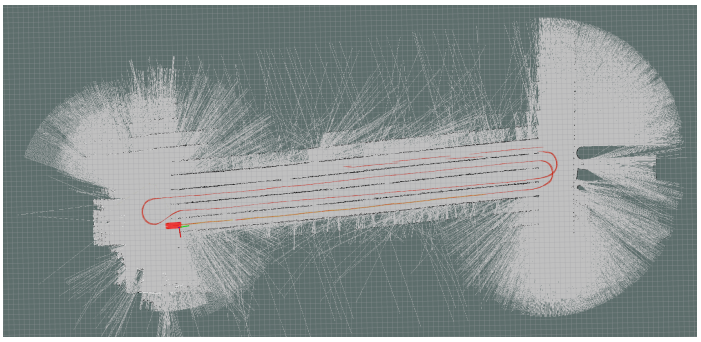
Figure 4.6: Comparison of two separate mapping packages with updated localisation configuration. Maps use the same data of the vineyard in Hastings New Zealand.

At the field testing in Hastings, the 'RTAB-Map' package failed during field testing with an ICP (Iterative Closest Point) error while going down the third row during the mapping process, as shown in Figure 4.6a. This type of failure is mentioned in the paper [68], where long corridor-like structures with low geometry complexity caused the package to not be able to localise the vehicle relative to the row. This was thought to be the only error involved with the mapping failure, but after all the current testing was done, the same data was tested again but with a better localisation configuration (latest version); this led to the vineyard being mapped correctly by 'RTAB-Map' and can be seen in Figures 4.6b and 4.6c. This likely means that during field testing, the localisation was not ideal for 'RTAB-Map' and assisted in the ICP error occurring caused the mapping failure.

Due to 'RTAB-Map' failing during field testing, a new solution was created in a short amount of time to allow testing to continue. This new solution involved the mapping package, 'Gmapping', which resulted in the maps like that presented in Figure 4.7b. The



(a) 'Gmapping', architecture setup used at field testing.



(b) 'Gmapping', architecture setup used at the end of the project.

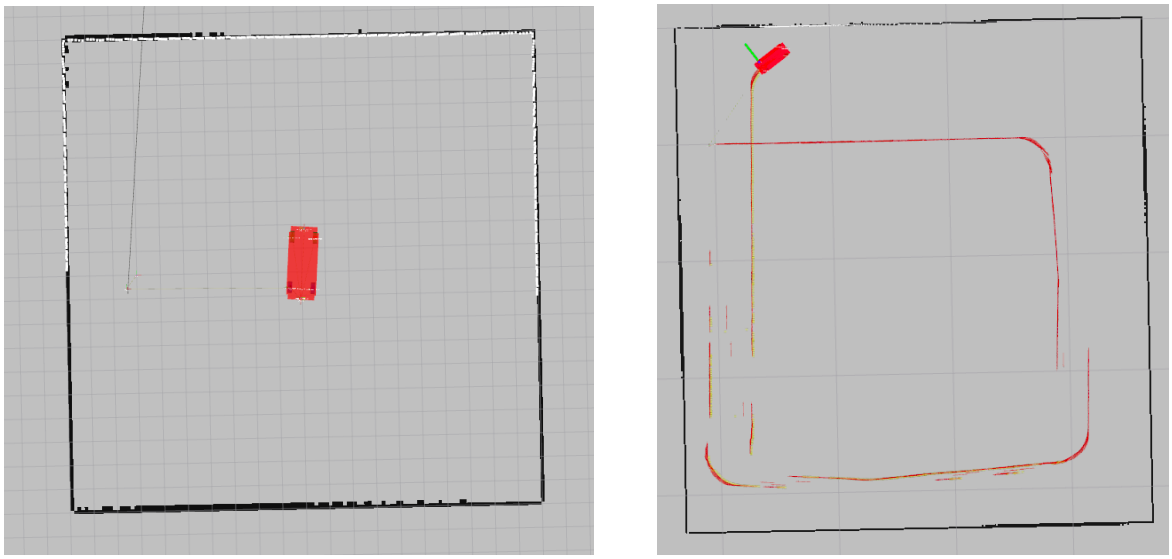
Figure 4.7: Comparison of two separate mapping packages with updated localisation configuration. Maps use the same data of the vineyard in Hastings New Zealand.

use of 'Gmapping' was later backed when it was mentioned in the research [40] that looked at mapping options for a vineyard. 'Gmapping' was part of the final choice from six different mapping solutions. It was noted within this paper that 'Gmapping' provides consistent maps even if they are not precisely aligned, demonstrating a slightly bowed shape. This bowing is not seen or is minor in Figure 4.7. These reasons led to the 'Gmapping' package being used in the final testing, where it was expected to work well for small Gazebo worlds and still provide good results for larger worlds similar to the real vineyard.

When comparing the occupancy maps created by both mapping packages with the current setups, it can be seen that they present very similar results, with 'RTAB-Map' in Figure 4.6b providing greater detection of the rows when compared to 'Gmapping' shown in Figure 4.7b. This is likely because the occupancy threshold parameter is different between the two packages. This was lowered for the 'Gmapping' package to help ignore long grass found in the rows, which can be seen in a couple of the rows for 'RTAB-Map' (as black dots in a row). When looking at the two Figures 4.6b and 4.7b, there is an odometry line that shows the path taken by the vehicle when mapping. The mapping package adjusts this path to align the map with prior points through their own stitch methods. It can be seen when 'Gmapping' was used that the odometry line jumps around a little and sways to the edge of the rows, which did not occur during the data collection. When looking at the odometry from the 'RTAB-Map' mapping, it produced steady and

consistent odometry. These results demonstrate that 'RTAB-Map' is a suitable package for mapping a vineyard, even though initial testing determined it not to be. This comes from the package performing just as well as 'Gmapping' if not slightly better by maintaining its localisation.

As 'Gmapping' relies on obstacles for reference (like most mapping packages), it was tested by making it map empty areas that were fenced, with sizes of 20 m^2 and 50 m^2 as demonstrated in Figure 4.8. The other areas that required mapping were the ones used for the other experiments, which involved a small vineyard world and a large 50 m^2 world with large rows breaking it up.



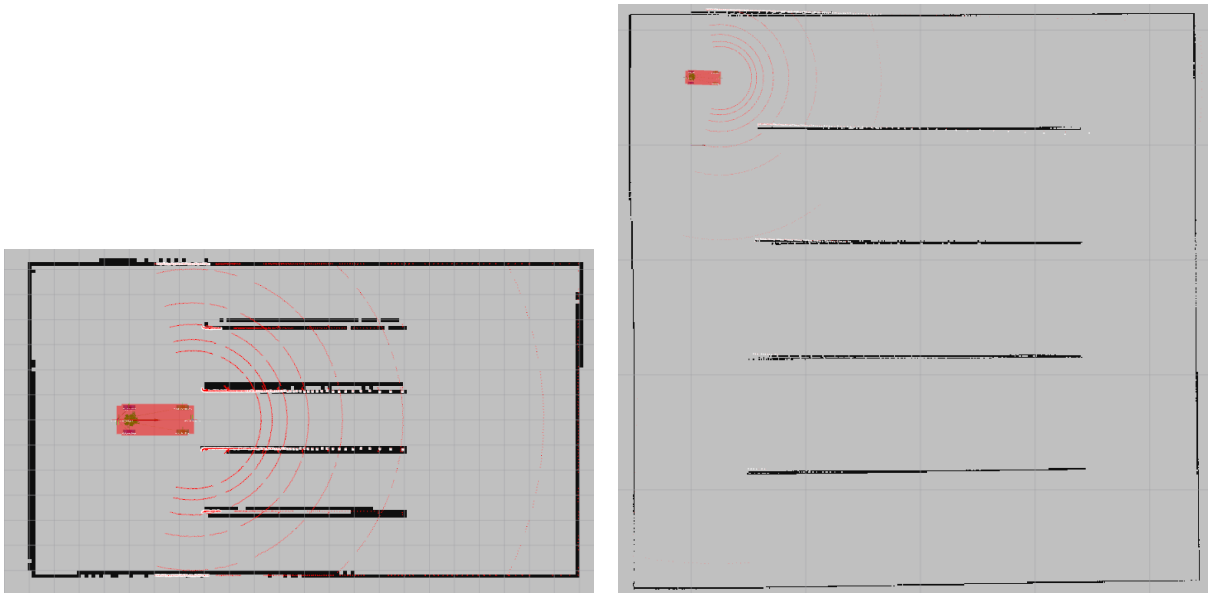
(a) 'Gmapping' package used to map 20 m^2 area for obstacle avoidance. Cell size is 1 m.

(b) 'Gmapping' package used to map 50 m^2 area. Cell size is 10 m.

Figure 4.8: The PID controllers ability to correct for large deviations from the target distance.

When the package is only able to see one reference (such as a wall), it begins to lose position, and once it is unable to see any reference points (empty space), it begins to lose orientation as well and, therefore, the mapping performance decreases. This can be seen in Figure 4.8b, where the size of the map is not 50 m^2 but closer to 40 m^2. This is not too significant of an issue due to the AMCL package being able to correct the vehicle's pose as it gets closer to more reference points, but it is not ideal. When the 'Gmapping' package get multiple references, it works well, showing a 20 m^2 area as seen in Figure 4.8a, due to it having vision on three out of the four walls.

The following Figure 4.9 demonstrates the occupancy maps created for the global planner experiments, which are similar in size to the prior empty maps but include rows to create a more complex environment for the planners.



(a) Vineyard occupancy map with vine rows, at 2.5 m spacing. Each cell is 1 m^2.

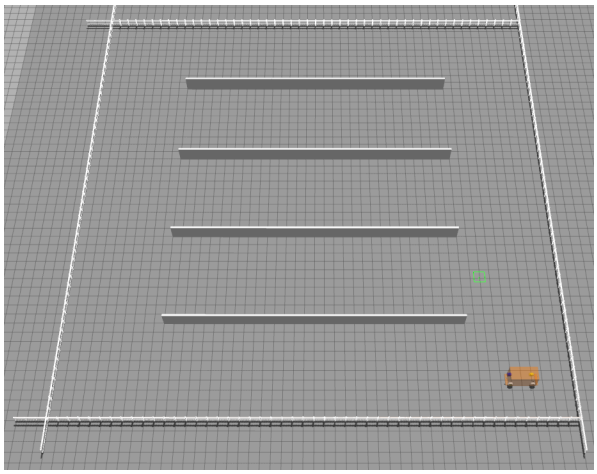
(b) 50 m^2 occupancy map with solid walls for rows, at 10 m spacing. Each cell is 10 m^2.

Figure 4.9: Maps used for global planner experiment.

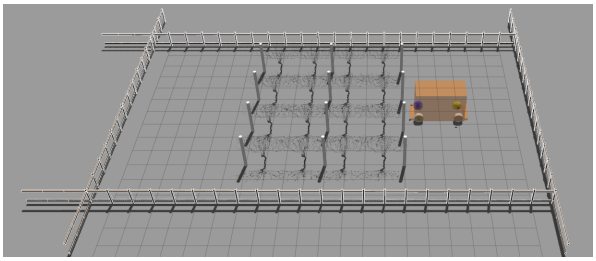
To test the planners in a near to real vineyard environment, the map in Figure 4.9a was created based on the vineyard world seen in Figure 4.10b. This map worked well for navigation despite the rows not being very defined but was able to retain the world size correctly, similar to Figure 4.8a. The map shown in Figure 4.9b was created to test the global planners in a large, relatively open area to detect any downfalls when there is a lot more space to search. This map had retained its size, unlike Figure 4.8b, but the map can be seen to be distorted slightly and not fully symmetric on its main axes as it should be. This caused some obstacles to be in a slightly different position than expected by the map. These issues did lead to some minor difficulties during the planner experiment by affecting the vehicle's pose, which will be looked into further in the planner experiment section in this chapter. These results show that the 'Gmapping' package is suitable for vineyard rows and small open areas, but any large open area will require a better mapping method or navigation method to traverse.

4.5 Path Planning Comparison

The three path planners used for testing involved Dijkstra, A*, and Bias-IRRT*. The experiment aims to determine the optimal planner to navigate a corridor crop environment. To compare each planner, a record was kept of the plans and paths taken when these were followed, along with the times taken to plan and complete the journey. To test the planners, they were used in three different scenarios, each attempting to represent an environment within a vineyard. The first was a straight-line (control) test at 10 m and 40 m, where each planner was used three times. The second was to plan diagonally through a large 50 m^2 map, which was split into 10 m intervals with walls, with each planner being used ten times. Thirdly was to plan through a small vineyard world with 2.5 m row spacing (centre to centre), with each planner being used ten times. These experiments were all done in simulation, with the worlds shown below in Figure 4.10. Once mapped, the starting position was kept consistent within the different scenarios but changed when the world had changed. This was done by using a goal publisher script to place the goal in the same position. It was also essential to select the starting position within an area with consistent cost values around it so that an uneven cost distribution would not skew the paths.



(a) 50 m^2 world created in Gazebo for planner testing.



(b) The vineyard world was created in Gazebo with the row length set to 8 m.

Figure 4.10: The two Gazebo worlds that were used for path planner testing.

Each planner had their parameters set as seen in Figure 4.11, with each one being

tuned to work best in the two simulated worlds.

```
GlobalPlanner_Astar:
  allow_unknown: false
  default_tolerance: 0.0
  publish_potential: false
  use_dijkstra: false
  use_quadratic: true
  use_grid_path: false
  old_navfn_behaviour: false
  orientation_mode: 1
  neutral_cost: 45
```

(a) A* parameters that were used for testing.

```
GlobalPlanner_Dijkstra:
  allow_unknown: false
  default_tolerance: 0.0
  publish_potential: false
  use_dijkstra: true
  use_quadratic: true
  use_grid_path: false
  old_navfn_behaviour: false
  orientation_mode: 1
```

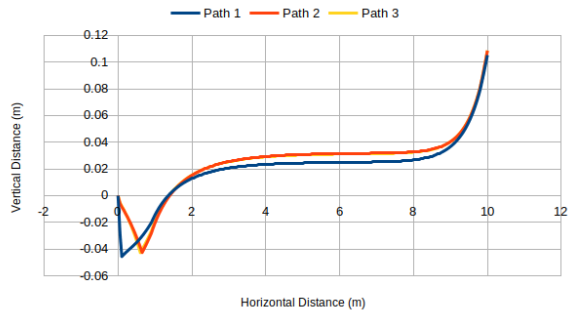
(b) Dijkstra parameters that were used for testing.

```
GlobalPlanner_InformedRRTStar:
  step_size: 4.0
  min_step_size: 0.4
  bezier_smoothing_value: 10
  max_cost: 150
  neighborhood_constant: 45
  path_check_amount: 65
  debug: false
  bias_value: 10  # 0 is off or no bias
```

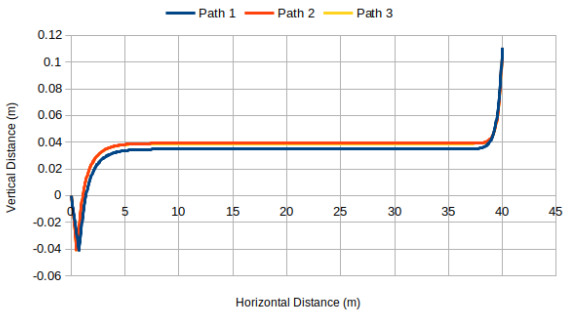
(c) Biased-IRRT* parameters that were used for testing.

Figure 4.11: The two Gazebo worlds are used for path planner testing.

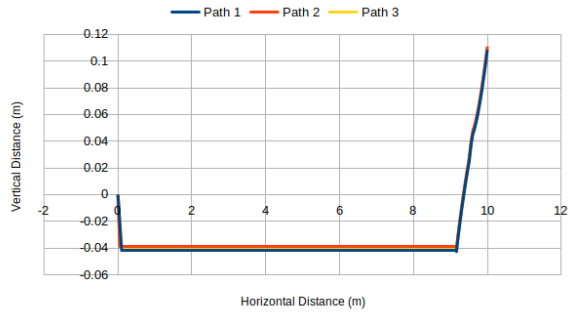
Because the planners use different cost functions, the time to plan and navigate was used to quantify the planner's results. Towards the end of testing, it was found that the ROS time was not equivalent to real-world time, with the ratio being determined by what Gazebo world was used. The times were scaled based on the ratio of simulation time to real-world time to ensure the data could be of use when comparing the different environments. The scaling provided improved results compared to using the original ROS times but should be remembered and changed to record the computer time for future testing.



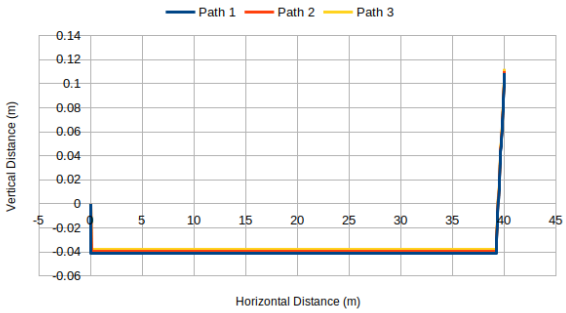
(a) Dijkstra plans for 10 m goal.



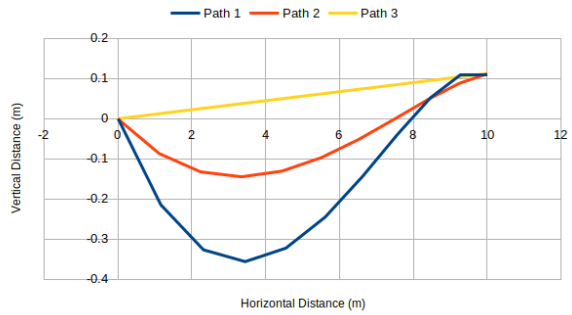
(b) Dijkstra plans for 40 m goal.



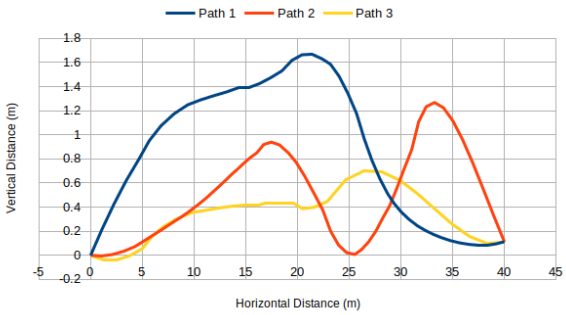
(c) A* plans for 10 m goal.



(d) A* plans for 40 m goal.



(e) Bias-IRRT* plans for 10 m goal.



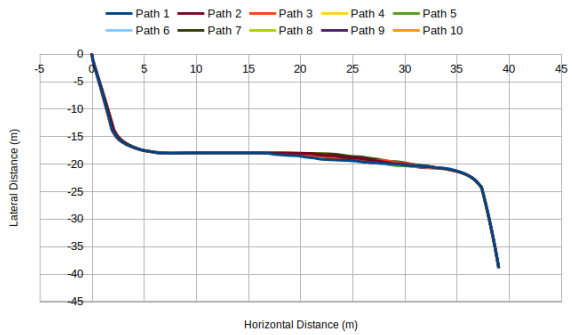
(f) Bias-IRRT* plans for 40 m goal.

Figure 4.12: Control experiments testing planners on a line of site goal.

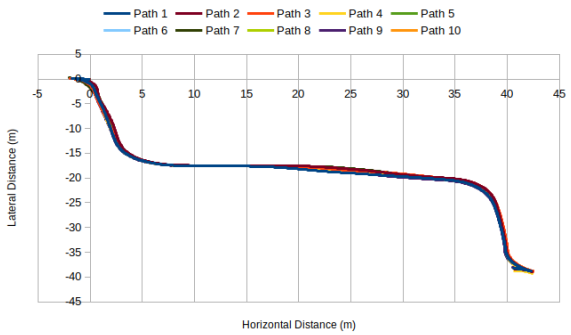
The outcome from the "control" (straight-line) experiments show that Dijkstra and A* formed relatively straight, optimal plans to traverse. It is interesting to compare the two plans created with Dijkstra creating curved paths and potentially better for a vehicle to follow, while A* produced consistent, sharp paths. A notable observation is that both planners had an initial point downward in the paths, which is a move that does not follow their cost functions. This is likely due to the 'base_link' position not being centred on a grid cell and because the planners are discrete, and using the grid cells to provide these fixed steps for the paths caused this slight offset.

The Bias-IRRT* planner can be seen to produce what seems to be the quickest path in the 10 m experiment with a straight line, but was still 20 s slower than the next planner A*. When comparing the path creation times to the paper [62], they produce similar results, with the time taken in the paper being around 4 s to get near-optimal solutions in a relatively non-complex solution. This may mean the planner required more time to produce a near-optimal path in more complex or larger areas.

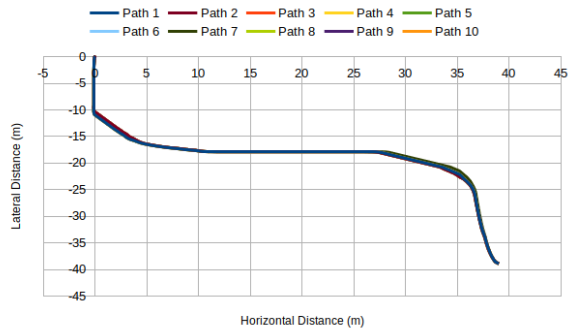
The following experiment covers the global planner's ability to plan consistent, optimal paths through a potentially multiple path environment. The following Figure 4.13, presents the plans and paths taken for the large 50 m^2 map seen in Figure 4.9b from the prior section.



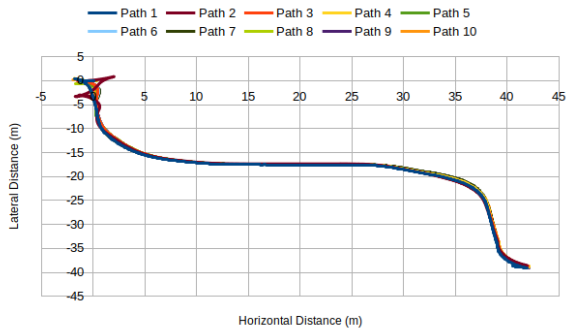
(a) Dijkstra plans.



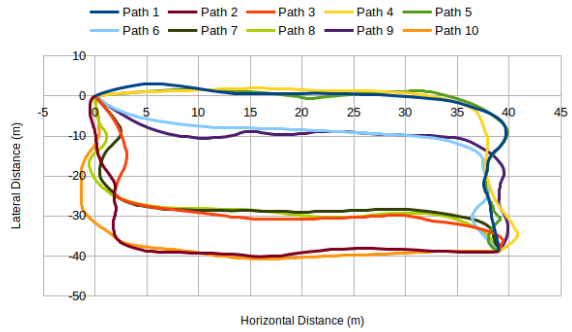
(b) The paths taken when using the Dijkstra plans.



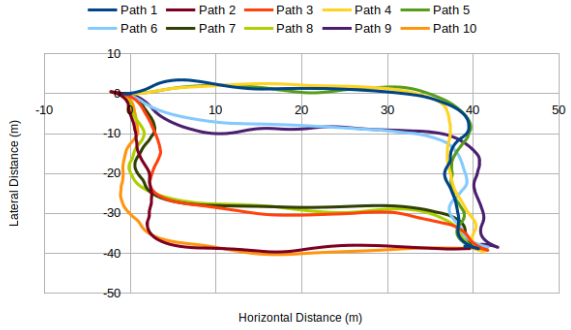
(c) A* plans.



(d) The paths taken when using the A* plans.



(e) Bias-IRRT* plans.

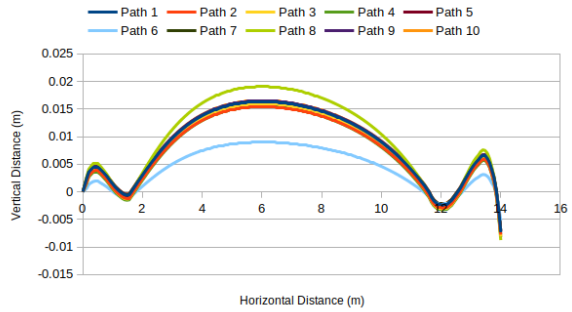


(f) The paths taken when using the Bias-IRRT* plans.

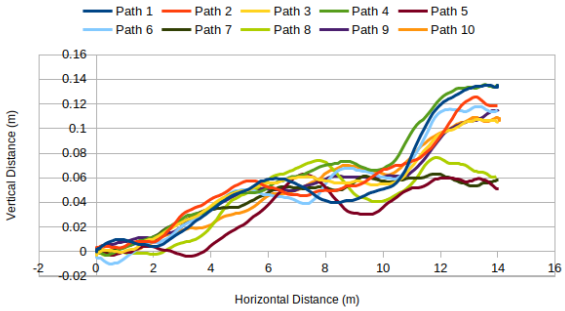
Figure 4.13: The plans created and paths travelled through the 'big world' environment.

As illustrated in Figures 4.13a and 4.13c the two planners that provided consistent and what looked to be optimal paths were the Dijkstra and A* planners. One main difference between these two planners is that the Dijkstra planner produced much smoother, rounded paths when compared to A*. This meant it was easier for the vehicle to follow and can be seen in Figures 4.13b and 4.14d, where the vehicle turns multiple times at the start with the A* planner. The Bias-IRRT* planner produced paths on almost every row,

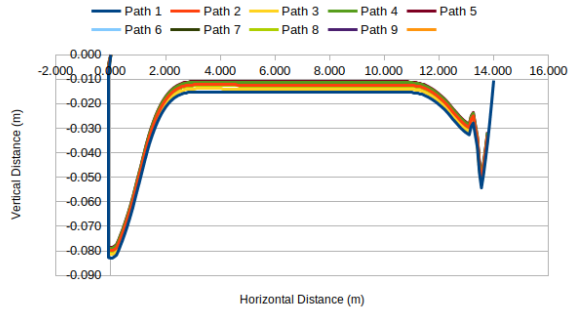
making it inconsistent, and surprisingly the one row it did not make a path down is the one both Dijkstra and A* had taken every time. When running the experiment, it was found that the planner connecting a new node to the closest branched node would make it progressively more difficult for branches to be created in the centre of the map as there will be established branches around it in other rows already. This would make any new node connect to the outside branches rather than potentially make a more optimal branch in the middle of the map. A limitation to the Bias-IRRT* was found where if the current plan is long due to weaving around and through rows, the 'informed' part of the planner does not help as the ellipse is too large to narrow down the search space. Therefore this made plans vary much more when compared to the straight-line experiment. Therefore with these two now known limitations of the Bias-IRRT*, it is understandable why it was difficult for the planner to produce a plan in the centre row. For it to do so, the planner would need many more iterations.



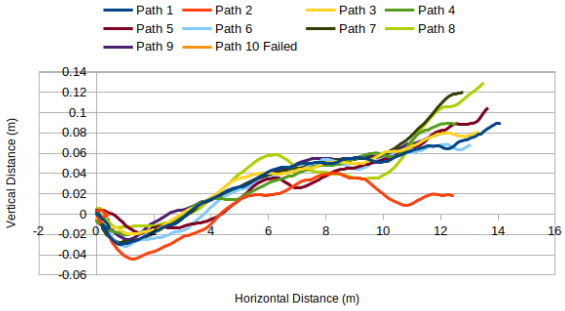
(a) Dijkstra plans.



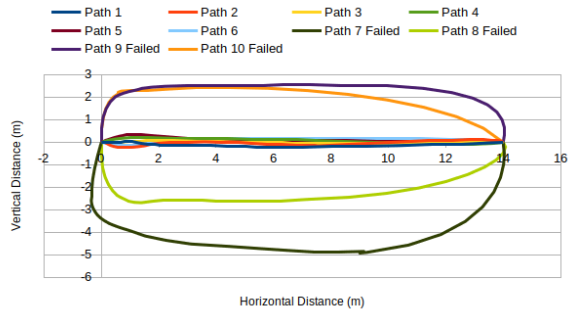
(b) Paths travelled using the Dijkstra plans.



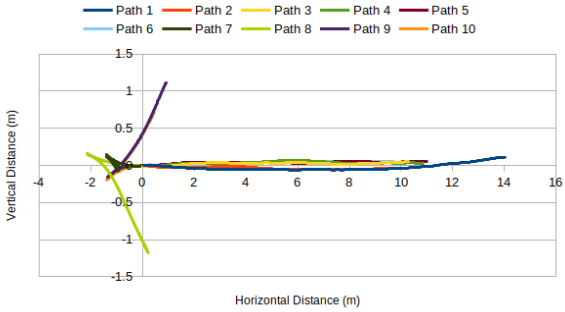
(c) A* plans.



(d) Paths travelled using the A* plans.



(e) Bias-IRRT* plans.



(f) Paths travelled using the Bias-IRRT* plans.

Figure 4.14: The plans created and plans travelled through the 'vineyard world' environment.

The 'vineyard' experiment produced similar results to the 'big world' experiment. Dijkstra and A* were the most consistent planners, producing paths down the row, which was the optimal path to take. The only concern was that the A* planner did fail to create the plan one time, with it showing a plan on RVIZ but was unable to retrace and publish it. This is likely a bug in the A* planner from the navigation stack that is caused when the cost values become larger with less room for the planner.

The Bias-RRT* planner was able to produce 60% of its paths down the row but failed any other time because of it trying to plan down other rows. This success rate is for

similar reasons to the 'big world' experiment, but due to the small search area to start with, it produced high chances of making shorter paths allowing the search area to shrink more until a path down the row was created.

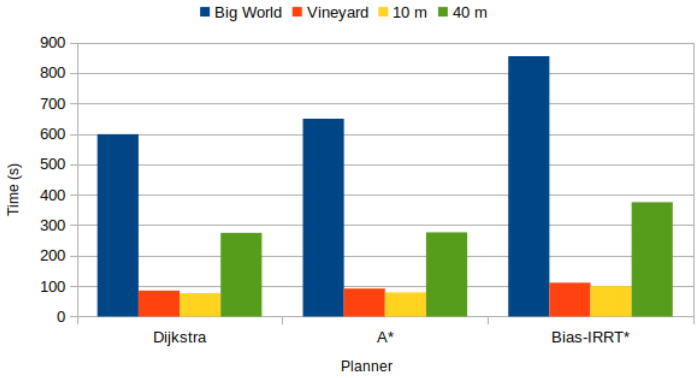
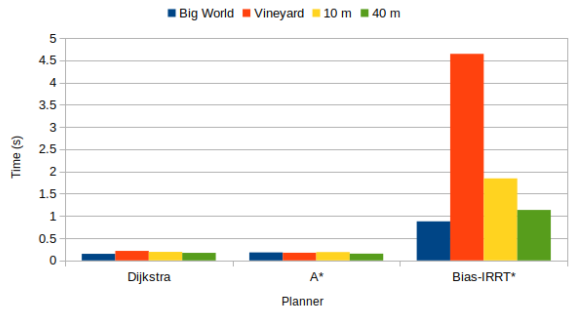


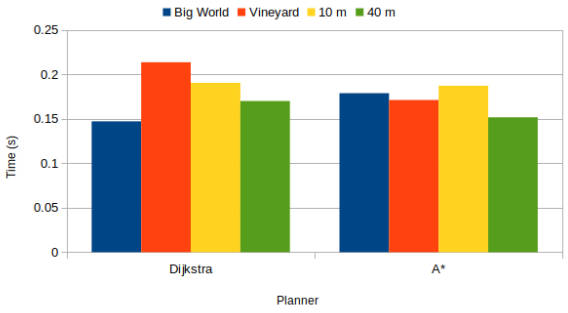
Figure 4.15: Time taken to navigate to the goal using the planed path for each planner.

When looking at the navigation times in Figure 4.15, it can be seen that the Bias-IRRT* is the slowest in every environment, and considering it took a significant amount more time when compared to the other two planners to create these plans. When comparing Dijkstra and A*, the navigation times are very close to each other, but it is demonstrated that Dijkstra's plans are slightly faster than A*'s. This would be because Dijkstra seemed to plan smoother paths, while A* is more ridged when turning. This, in turn, caused the vehicle to struggle when having to follow such sharp turns, which was illustrated in Figure 4.13d. A similar performance between the two grid-based planners is also seen in the paper [101] and showed that significant differences only became apparent in kilometre ranges.

The time taken to plan the paths were also recorded to see if there was any noticeable difference in planners and is seen in Figure 4.16.



(a) Time taken to plan a path to the goal for each planner.



(b) Zoom in on A* and Dijkstra planning times from Figure 'a'.

Figure 4.16: Effects of planner type and environment on planning time.

Figure 4.16, shows the experimental difference between the Dijkstra and A* planners better. Overall the differences are very minimal, but a few things can be noted, such as that A* found a path in the vineyard (a more complex) environment approximately 24% faster than Dijkstra. At the same time, Dijkstra performed approximately 21% better on the large across map goal. It is also interesting to see that Dijkstra seems to perform faster as the goal gets further away, which is entirely against the nature of the algorithm, with it needing to search every cell up to reaching the goal. This is in comparison to the A* times which stayed relatively consistent in time taken to form a path.

When reviewing all the planner results, the planners that showed the best performance for the vineyard environment were the grid-based planners due to them producing the best looking and most consistent paths. All of these planners are usable for the current vehicle, but in future research, there should be a focus on testing a kinematic planner such as the hybrid A* planner [32].

4.6 Obstacle Avoidance

Being able to avoid known obstacles is mainly tasked to the global planner while avoiding obstacles that the global planner was unaware of is given to the local planner. Both cases are tested in the following experiment, with the path taken when the obstacle was known being used as an ideal path for reference. This emphasises how well the local planner can adjust and travel safely to the goal. The plans and the paths were recorded ten times for

each obstacle to determine the efficacy of the local planner when following the global path and when having to avoid an obstacle in the way. The planner of choice to provide the global plan (ideal path) was the A* planner due to it finding the optimal or near-optimal path every time while still moving towards the goal.

The setup included creating a 20 m^2 mapped area as seen in Figure 4.17. Additionally, obstacles were created or sourced appropriately for use in the testing process, with these obstacles being presented in Figure 4.18.

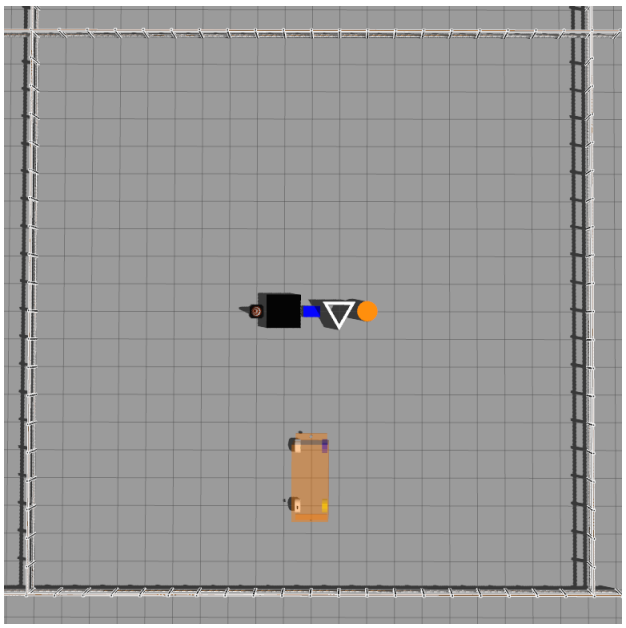


Figure 4.17: 20 m^2 Gazebo world used for obstacle avoidance testing.

The obstacles used included were meant to represent potential objects found on a vineyard and or represent generic shapes. The objects included the following; a construction cone (0.5 × 0.5 m), a large harvest bin (1.2 × 1.2 × 0.65 m), a small harvest bin (0.6 × 0.4 × 0.167 m), a triangle with each side being 1.15 m, and a pillar (0.36 m radius and 1 m tall). Each of these objects can be seen in Figure 4.18.

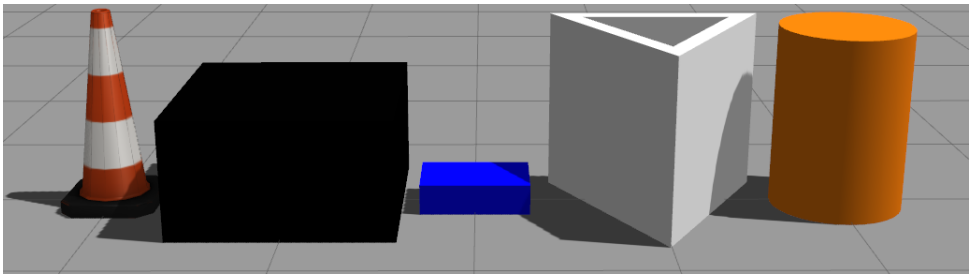
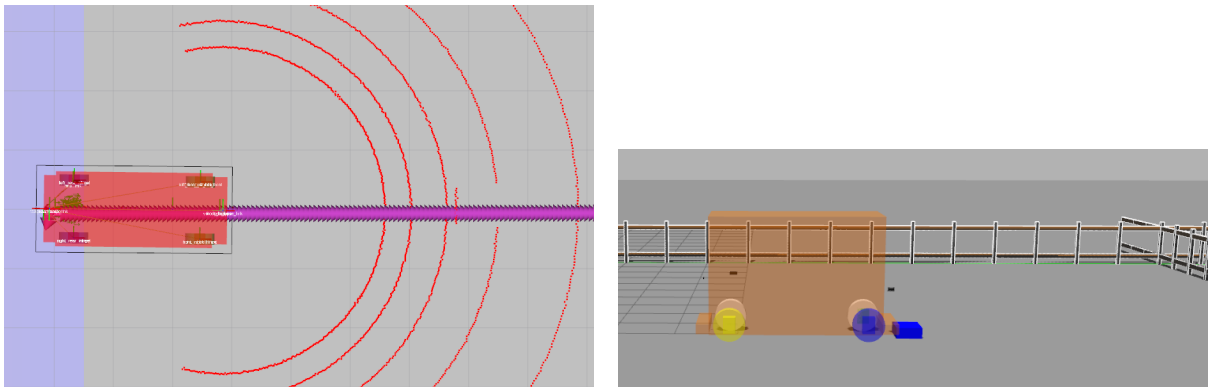


Figure 4.18: Four different obstacles were used to test the avoidance of the SAA.
Names of objects: cone, large harvest box (black), small harvest box (blue), triangle, and pillar.

The construction cone was sourced from the Gazebo models library [102], the boxes were made using standard URDF format but based on real harvest tools [103][104], and the triangle was built using CAD software. The experiment operation involved placing the centre of the obstacle 7 m from the rear axial (base_link) of the vehicle (approximately 4 m from the front of the vehicle) and the goal 13 m from the start location on the other side of the obstacles.

The first experiment looked at potentially one of the shortest and likely obstacles to come across in a vineyard environment, this being a small harvest bin used by workers when harvesting [104].



(a) Global planner plans right through the small harvest box.

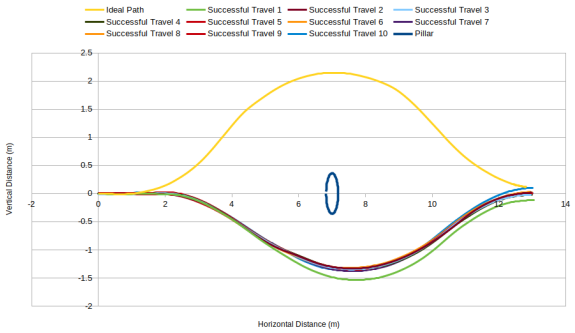
(b) Local planner was unaware of the small harvest box and travels straight into it.

Figure 4.19: Global and local planners did not take notice of small box and end up pushing it.

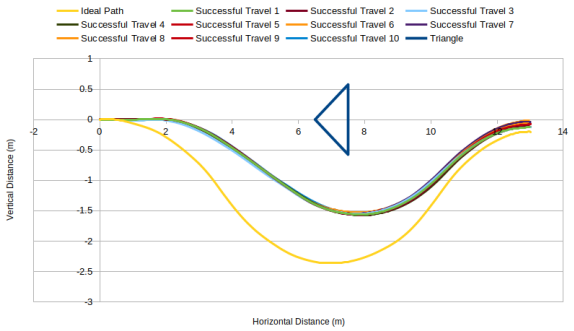
The small harvest box was hit in both cases of the experiment. This happened because neither map could see such a short box due to the minimum detection height of the LiDAR

being 0.4 m. This is a concern for the SAA as this item should be detected and avoided if possible.

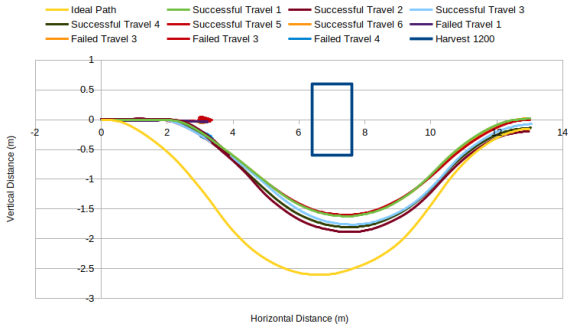
Covering general shapes was deemed necessary because the vehicle will come across similar objects and surfaces in a vineyard environment. Figure 4.20 demonstrates the results from the three experiments, with overall solid results for the pillar and triangle shapes, while providing intriguing information on the large harvest box.



(a) Local planners avoidance of a pillar.



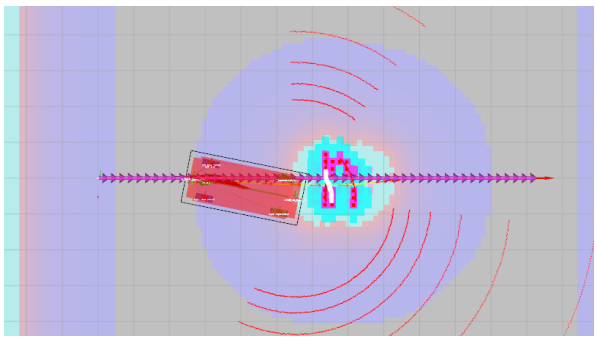
(b) Local planners avoidance of a triangle.



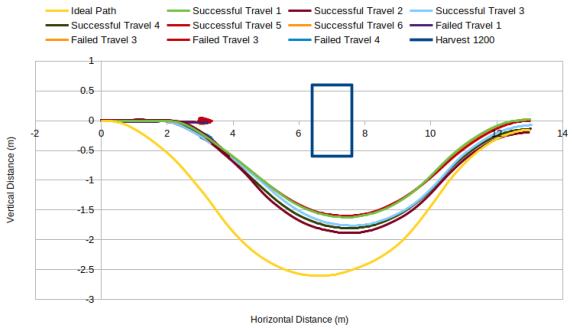
(c) Local planners avoidance of the large harvest box.

Figure 4.20: Comparison of travelled paths using A* planner and a straight plan through the generally objects.

The pillar and triangle experiments seen in Figure 4.20 demonstrated near ideal paths. The fact that the travelled paths are much closer to the object was expected due to the local planner having a smaller window of detection when compared to the global planner. The large harvest box was able to serve as a general shape as well as a real object that is used in the agriculture world for harvesting produce [103].



(a) Local planner failed by planning through large harvest box.

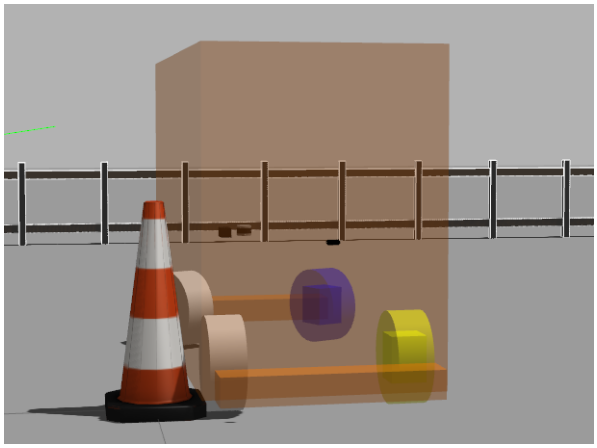


(b) Local planners avoidance of the large harvest box.

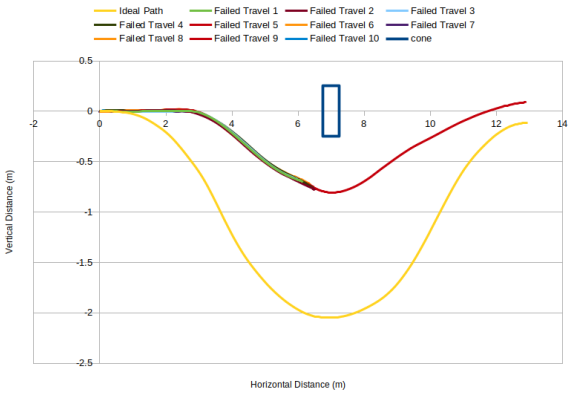
Figure 4.21: Comparison of a travelled path using A* planner and the paths taken relying solely on the local planner to get around the large harvest box.

F Interestingly this object was only evaded 60% of the time when relying only on the local costmap info and local planner, as illustrated in Figure 4.21b. Given the object's size, this result is unexpected, where the local planner should have been able to avoid this obstacle every time. This was due to the local planner trying to plan through the box where a voxel was not placed on the costmap, as seen in Figure 4.21a. This is a questionable action made by the local planner as it knows a rough idea of the vehicle size and is aware of the local costmap, meaning it would not be expected to plan through such a tiny gap. This was the only reason for colliding with the large box, with the successful attempts getting around the obstacle with no problem, which is demonstrated in Figure 4.21b. The issue could potentially be fixed by applying a local costmap plugin called "Costmap_conversion" that joins co-inside voxels to make a solid object on the costmap rather than individual points.

The worst performing experiment involved the construction cone, where it failed 100% of the time as seen in Figure 4.22.



(a) Vehicle colliding with cone base in Gazebo.



(b) Local planners avoidance of the cone.

Figure 4.22: Comparison of a travelled path using A* planner and the paths taken relying solely on the local planner to get around the construction cone.

While trying to navigate around the cone, the vehicle would begin to evade the cone, and once it thought it was clear, it would turn and collide into the base of the cone, as demonstrated in Figure 4.22a. This was due to the local costmap only showing the top of the cone as an obstacle. Because the top is significantly smaller than the base, the costmaps inflation layer was not large enough to make the vehicle take a wider turn. This is a significant result as cones are obstacles that will be present in an agricultural environment and need to be evaded appropriately.

When considering all of the obstacle avoidance experiments, the issue with the current SAA is the limit in height detection of obstacles, in combination with the small local costmap and the reasonably close path taken by the local planner. The issues with the large harvest box and the construction cone could be fixed by incorporating a periodic re-planning using the global planner as it looked to still produce a safe plan with minimal detection. This re-planning will still not help with obstacles below the detection height and will benefit from more complex solutions. The way to fix this problem is to lower the height at which LiDAR data is incorporated into the local costmap but filter out non-obstacles such as long grass. This could be done by increasing the detection threshold and or sensor fusion by incorporating a camera to detect and filter out irrelevant objects.

4.7 Ability to Transfer Autonomy

The main objective of this research is to find a modular SAA that can be transferred from one custom platform to another for an agricultural environment. For this architecture to be transferred, a few components will need to be changed and or tuned.

- URDF: This would need to be changed to represent the new vehicle and sensor layout.
- TEB Local Planner: This would likely take a reasonable amount of time to tune for any change in vehicle.
- Robot Localization Package: This may require a small amount of tuning due to a change sensor layout.
- Costmap 2D Package: The tuning of parameters may be required.

The number of changes needed during the transfer process will depend significantly on how much the new vehicle and tasks differ from those used in this research. The types of vehicles this architecture can be applied for are other four-wheeled vehicles that implement Ackermann or differential styled drive control and potential for vehicles that allow for omnidirectional driving.

4.8 Summary

This chapter covered the evaluation done on the SAA, with a golf cart as the base vehicle. Through field testing, it was found that the SAA required more testing within a vineyard environment and adjustments to the architecture to allow it to be used in all areas of agriculture. These conclusions came from the SAA's ability to first map a vineyard using 'RTAB-Map', which changed after field testing to a 'Gmapping' configuration. This was later found to be an issue with 'RTAB-Map' and the localisation package setup at the time, as it successfully ran at the end of this project. Field trials also provided a greater understanding of the SAA's navigation side, with it only struggling to navigate down the vineyard rows. A couple of components were thought to create this navigation difficulty.

The first is the golf cart's minimum speed and delayed braking at the time on sloping terrain. The second was the costmap parameters being too strict, making it difficult for the global planner to perform correctly and, therefore, to navigate. Using this information, an improved simulation was developed and used for testing as the vineyard was not easily accessible.

To allow a vehicle to help workers or potentially harvest produces, it was deemed essential to have the SAA involve a 'maintain row distance' mode. This was done using a 3D LiDAR and a PID controller, which showed excellent results by tracking a distance with a mean error and a standard deviation of a few millimetres. These results should be taken as a starting point, with field testing being a true test for this component of the SAA.

Determining the best type of global planner for a vineyard environment was done by comparing three planners, Dijkstra, A*, Bias-IRRT*. This compares two very similar grid-based planners from the ROS navigation stack (Dijkstra and A*) to a sample-based planner (Bias-IRRT*). The key findings from this comparison were that the grid-based planners outperform the sample-based planner, which is understandable due to a vineyards layout being ordered and grid-like structure.

When an obstacle is unknown to the global planner, it is expected that SAA is capable of avoiding it. This was tested and placed a focus on the local planner to perform intelligently based on its current situation. To test this component of the SAA, general shaped obstacles and obstacles that could be found on a vineyard were positioned approximately 4 m in front of the vehicle and tasked to get to a position on the other side. This provided findings where 40% of the obstacles were entirely avoided, 10% partially avoided, and the other 40% failed. This was determined by the representation of the obstacle within the local costmap, which was impacted by the shape and size of each obstacle. This concluded that periodically calling the global planner to ensure an optimal and obstacle-free path will assist this issue. However, additional filtering and detection of surroundings are required for greater performance in obstacle avoidance.

A vital purpose of the SAA was to make it a transferable system for agricultural vehicles. Even though this was not fully tested in the research, it is believed that the SAA created will be suitable for other four-wheeled vehicles. There will be an increase in

difficulty when the new vehicle begins to differ greatly from the one used in this research, but the central adaptations are in software. The final chapter concludes the many sections of this work while bringing forth areas of the SAA that require further work to solidify the SAA's functions.

Chapter 5

Conclusions and Future Work

This chapter provides a conclusion of the thesis and the outcomes found through this research. Additionally, the chapter also covers potential improvements for the SAA that were found through the development and evaluation stages.

5.1 Conclusion

The overall aim of this thesis was to present a transferable architecture that transforms a vehicle into a semi-autonomous platform, which can be implemented in any corridor crop environment. The SAA fulfilled this objective. The architecture was integrated into a golf cart used as a platform for real-world testing and development. Additionally, developed simulations provided the necessary environments to evaluate the SAA's performance in many scenarios. This helped determine its efficacy as a tool to create a semi-autonomous vehicle.

The final SAA was achieved with a combination of hardware and software architectures, which were integrated using a ROS framework. These architectures were created to provide the SAA and, therefore, the platform to localise, map, navigate and communicate within corridor crops. Multiple sensors and electronic components were used to build this perception of the vehicle and the world around it, providing the vehicle with the physical capability to be semi-autonomous. The development of software and use of ROS packages provided the necessary intelligence to retrieve, react and plan within the corridor crop

environments.

This thesis also presented several experiments to evaluate the SAA's performance. This included field trials which yielded useful information about the SAA, as these trials illustrated the downfalls of testing in an open concreted area. This was demonstrated when attempting to map and navigate a vineyard, as the SAA's software component faulted and had to be adapted at field trials to be used for continued testing. After this, it was found that testing the SAA in a non-corridor crop environment will not assist in further developments.

The simulation was an additional method used, which allowed for the construction of specific worlds to represent different environments found on agricultural sites. Using these simulations allowed for the evaluation of different components, such as maintaining a specific distance from a row, mapping, navigation, and obstacle avoidance.

Through the use of simulations, it was found that the row follower was able to track a specific distance without exceeding an error of 5 mm. It was also found that monitoring both sides of the row was less effective for the current configuration of the controller, compared to monitoring one side only.

The mapping solutions that were compared and evaluated were 'Gmapping' and 'RTAB-Map'. The final solution included the 'Gmapping' package and was chosen due to 'RTAB-Map' failing during field trials. However, with the current localisation configuration, 'RTAB-Map' provided an accurate map of the vineyard used in field trials. This allowed for a comparison to be made between the two packages, which demonstrated that both were suitable for mapping vineyard environments. Due to 'Gmapping' being part of the final SAA, it was tested further to determine when the mapping package would start to fail. The point of failure occurred when the map became larger and sparse of references, which was seen at an environment size of 50 m^2 for the SAA. These maps were used to test the global planners Dijkstra, A*, and bias-IRRT*. This revealed that the grid-based planners Dijkstra and A* significantly outperformed the sample-based planner, bias-IRRT*. Where

the grid-based planners produced consistent and near-optimal paths, the sample-based planner produced multiple paths in all environments and was only successful 60% of the time within the vineyard world.

Lastly, an experiment was designed to test how effective the local planner was at evading obstacles that the global planner was unaware of. The results showed the SAA's need for different types of sensors and software to provide additional obstacle detection, with smaller and or oddly shaped objects not being seen and therefore not evaded.

This thesis presented many positives to the SAA and highlighted areas where there are opportunities for further developments to be made. Overall, the SAA is an AV development that can provide autonomous technology to many agricultural areas. Its potential to alleviate stressors within the agricultural sector makes the SAA a promising technology.

5.2 Future Work

Although the SAA can allow the golf cart to drive autonomously through a vineyard environment, there were still shortcomings found in this thesis. To further the SAA's autonomy level, the architecture requires more development on top of the findings from this research. The first few steps to achieve this should be to integrate more sensors such as the stereo camera and potentially a RADAR. This will likely allow for better obstacle avoidance and recognition, overall allowing the architecture to reach higher levels of intelligence. As mentioned in the thesis, having these different detection methods would probably allow the vehicle to recognise small objects such as the small harvest box, thin metal fencing and an improved the row detection system used to start the row following the event. Improving the software section can be a near-endless development, but the main parts to focus on would be the following. A kinematic version of the global planners such as hybrid-A*, combined with a Reeds-Shepp or Dubins, would be worth investigating. This would be expected to improve the SAA for non-holonomic vehicles by providing more information for the TEB local planner and creating easier paths to follow. Improving and

adding to the current recovery methods would allow the vehicle to have a more robust system for any faults or anomalies.

References

[1] Tesla. Artificial intelligence & autopilot. Website, August 2021. Overview of current Tesla technology.

[2] Toyota. Toyota automated driving whitepaper. techreport, Toyota Motor Corporation, August 2020.

[3] Waymo. Waymo safety report. techreport, Waymo, February 2021. The report demonstrates the Waymo vehicle, its safety features and testing it goes through.

[4] K Morgan. A step towards an automatic tractor. *Farm mech*, 10(13):440–441, 1958.

[5] C. Thorpe, M.H. Hebert, T. Kanade, and S.A. Shafer. Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373, May 1988.

[6] John Deere. John deere reveals fully autonomous tractor at ces 2022. Article, January 2022. Fully autonomous tractor that's ready for large-scale production.

[7] Statista. Global production of fresh fruit from 1990 to 2020. Website, January 2022.

[8] Johannes Simon Cornelis. *Achieving sustainable urban agriculture*. Burleigh Dodds series in agricultural science ; 77. Burleigh Dodds Science Publishing, London, 1st edition, 2020.

[9] Stats NZ. Agricultural production statistics: June 2020 (final). Website, May 2021.

[10] Statista. Global fruit production in 2020, by selected variety (in million metric tons). Website, January 2022.

[11] Mahindra. Mahindra showcases its first ever driverless tractor in india. Mahindra technology showcase were they present their first ever driverless tractor., September 2017.

[12] Worksafe NZ. Major focus on reducing farm accidents involving vehicles. *News and Media*, March 2018. Article about the injurys and fatalities on NZ farms from vehicle and machinery. Made by Worksafe NZ (Government organisation).

[13] Open Source Robotics Foundation. Gazebo simulation. Webpage, January 2020. Simulation Software used for ROS system.

[14] Stanford Artificial Intelligence Laboratory. Ros melodic. Webpage, May 2018. Robotic Operating System.

[15] SAE. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles j3016. Webpage, April 2021.

[16] Tesla. Model 3. Webpage, January 2021. Tesla's partially autonomous electric vehicle.

[17] Murat Dikmen and Catherine Burns. Trust in autonomous vehicles: The case of tesla autopilot and summon. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1093–1098, Oct 2017.

[18] Erico Guizzo. How google's self-driving car works. *IEEE Spectrum*, October 2011.

[19] United States Department of Transport. Automation white paper. PDF on ITS website, 2015.

[20] Ryan Harrington, Carmine Senatore, John Scanlon, and Ryan Yee. The role of infrastructure in an automated vehicle future. *Bridge*, 40:48–55, 06 2018.

[21] Gokhan Bayar, Marcel Bergerman, A. Bugra Koku, and E. ilhan Konukseven. Localization and control of an autonomous orchard vehicle. *Computers and Electronics in Agriculture*, pages 118–128, 2015.

[22] Toyota. Statement regarding a collision between a pedestrian and a toyota e-palette vehicle at the tokyo 2020 olympic and paralympic athletes' village. Webpage, August 2021. Announcement by Toyota on 26th of August, 2021, about "e-Palette" incident.

[23] Roger L. McCarthy. Autonomous Vehicle Accident Data Analysis: California OL 316 Reports: 2015–2020. *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg*, 8(3), 09 2021. 034502.

[24] Koen Bussemaker. Sensing requirements for an automated vehicle for highway and rural environments. Master's thesis, Delft University of Technology, 2014.

[25] Paul Rau and Eric Thorn. A framework for automated driving system testable cases and scenarios. techreport, NHTSA, September 2018.

[26] Paolo Grisleri and Isabella Fedriga. The braive autonomous ground vehicle platform. *IFAC Proceed Volumes*, 43(16):497–502, 2010.

[27] Reinhold Behringer, S. Sundareswaran, Billy Gregory, R. Elsley, B. Addison, Wayne Guthmiller, R. Daily, and D. Bevly. The darpa grand challenge - development of an autonomous vehicle. In *IEEE Intelligent Vehicles Symposium*, pages 226 – 231, 07 2004.

[28] Reinhold Behringer, W. Travis, R. Daily, D. Bevly, Wilfried Kubinger, Wolfgang Herzner, and V. Fehlberg. Rascal - an autonomous ground vehicle for desert driving in the darpa grand challenge 2005. In *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.*, pages 644 – 649, 10 2005.

[29] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, and Ara Nefian. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23:661–692, 2006.

[30] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William "Red" Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[31] Mark Campbell, Magnus Egerstedt, Jonathan P. How, and Richard M. Murray. Autonomous driving in urban environments: approaches, lessons and challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368(1928):4649–4672, 2010.

[32] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Hähnel, Tim Hilden, Gabriel Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, and Sebastian Thrun. Junior: The stanford entry in the urban challenge. In *The DARPA Urban Challenge*, pages 91–123, 01 2009.

[33] Jonathan Bohren, Tully Foote, Jim Keller, Alex Kushleyev, Daniel Lee, Alex Stewart, Paul Vernaza, Jason Derenick, John Spletzer, and Brian Satterfield. Little ben: The ben franklin racing team's entry in the 2007 darpa urban challenge. *Journal of Field Robotics*, 25(9):598–614, 2008.

[34] Iris F.A. Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709, 2006.

[35] Pandu Sandi Pratama, Nguyen Trong Hai, Hak-Kyeong Kim, Dae Hwan Kim, and Sang Bong Kim. Positioning and obstacle avoidance of automatic guided vehicle

in partially known environment. *International Journal of Control Automation and Systems*, 14:1572–1581, October 2016.

[36] Alberto Vale, Rodrigo Ventura, Pedro Lopes, and Isabel Ribeiro. Assessment of navigation technologies for automated guided vehicle in nuclear fusion facilities. *Robotics and Autonomous Systems*, 97:153–170, 2017.

[37] Ji Zhang, George Kantor, Marcel Bergerman, and Sanjiv Singh. Monocular visual navigation of an autonomous vehicle in natural scene corridor-like environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3659–3666, Oct 2012.

[38] Marcel Bergerman, Silvio M. Maeta, Ji Zhang, Gustavo M, Freitas, Bradley Hamner, Sanjiv Singh, and George Kantor. Robot farmers autonomous orchard vehicles help tree fruit production. *IEEE ROBOTICS & AUTOMATION MAGAZINE*, 2015.

[39] Mark H. Jones, Jamie Bell, Daniel Dredge, Matthew Seabright, Alistair Scarfe, Mike Duke, and Bruce MacDonald. Design and testing of a heavy-duty platform for autonomous navigation in kiwifruit orchards. *Biosystems Engineering*, 187:129–146, November 2019. Received 14 December 2017, Revised 30 June 2019, Accepted 28 August 2019, Available online 18 September 2019.

[40] Ferran Roure, Germán Moreno, Marcel Soler, Davide Faconti, Daniel Serrano, Pietro Astolfi, Gianluca Bardaro, Alessandro Gabrielli, Luca Bascetta, and Matteo Matteucci. Grape: Ground robot for vineyard monitoring and protection. In Anibal Ollero, Alberto Sanfeliu, Luis Montano, Nuno Lau, and Carlos Cardeira, editors, *ROBOT 2017: Third Iberian Robotics Conference*, pages 249–260, Cham, 2018. Springer International Publishing.

[41] David Ball, Ben Upcroft, Gordon Wyeth, Peter Corke, Andrew English, Patrick Ross, Tim Patten, Robert Fitch, Salah Sukkarieh, and Andrew Bate. Vision-based obstacle detection and navigation for an agricultural robot. *Journal of Field Robotics*, 33(8):1107–1130, 2016.

[42] A. Medela, B. Cendon, L. Gonzalez, Raul Crespo Merino, and Ignacio Nevares. Iot multiplatform networking to monitor and control wineries and vineyards. In *2013 Future Network & Mobile Summit*, pages 1–10, 01 2013.

[43] Raivo Sell, Anton Rassõlkin, Ruxin Wang, and Tauno Otto. Integration of autonomous vehicles and industry 4.0. *Proceedings of the Estonian Academy of Sciences*, 68:389–394, 12 2019.

[44] Andrey Somov, Dmitry Shadrin, Ilia Fastovets, Artyom Nikitin, Sergey Matveev, Ivan seledets, and Oleksii Hrinchuk. Pervasive agriculture: Iot-enabled greenhouse for plant growth control. *IEEE Pervasive Computing*, 17(4):65–75, Oct 2018.

[45] Ronald Jurgen. *V2V/V2I communications for improved road safety and efficiency*. SAE International progress in technology series ; PT-154. SAE International, Warrendale, Pennsylvania, 1 edition, 2012.

[46] Subhadeep Patra, Peter Veelaert, Carlos T Calafate, Juan-Carlos Cano, Willian Zamora, Pietro Manzoni, and Fabio González. A forward collision warning system for smartphones using image processing and v2v communication. *Sensors (Basel, Switzerland)*, 18(8):2672–, 2018.

[47] Halter. Transform your farm with one system. Webpage, January 2021.

[48] Said Benaissa, Frank A.M Tuyttens, David Plets, Hannes Cattrysse, Luc Martens, Leen Vandaele, Wout Joseph, and Bart Sonck. Classification of ingestive-related cow behaviours using rumiwatch halter and neck-mounted accelerometers. *Applied animal behaviour science*, 211:9–16, 2019.

[49] P Sumathi, R Subramanian, VV Karthikeyan, and S Karthik. Soil monitoring and evaluation system using edl-asqe: Enhanced deep learning model for iot smart agriculture network. *International Journal of Communication Systems*, 34(11):e4859, 2021.

[50] Ana Paula Alves Torres, Claudio Bastos Da Silva, and Horácio Tertuliano Filho. An experimental study on the use of lora technology in vehicle communication. *IEEE Access*, 9:26633–26640, 2021.

[51] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, December 1959.

[52] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[53] Karl Kurzer. *Path Planning in Unstructured Environments: A Real-time Hybrid A* Implementation for Fast and Deterministic Path Generation for the KTH Research Concept Vehicle*. PhD thesis, KTH ROYAL INSTITUTE OF TECHNOLOGY SCHOOL OF ENGINEERING SCIENCES, 12 2016.

[54] Nak Yong Ko, Sung Woo Noh, and Yong Seon Moon. Implementing indoor navigation of a mobile robot. In *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, pages 198–200, Oct 2013.

[55] Sungkwan Kim, Hojun Jin, Minah Seo, and Dongsoo Har. Optimal path planning of automated guided vehicle using dijkstra algorithm under dynamic conditions. In *2019 7th International Conference on Robot Intelligence Technology and Applications (RiTA)*, pages 231–236, Nov 2019.

[56] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of a* pathfinding. *Procedia Computer Science*, 179:507–514, 2021. 5th International Conference on Computer Science and Computational Intelligence 2020.

[57] Mario Kusuma, Riyanto, and Carmadi Machbub. Humanoid robot path planning and rerouting using a-star search algorithm. In *2019 IEEE International Conference on Signals and Systems (ICSigSys)*, pages 110–115, July 2019.

[58] Lekshmi J Mohan and Joaquim Ignatious. Navigation of mobile robot in a warehouse environment. In *2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)*, pages 1–5, July 2018.

[59] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *AAAI Workshop - Technical Report*, 01 2008.

[60] Kwangjin Yang, Seng Keat Gan, and Salah Sukkarieh. A gaussian process-based rrt planner for the exploration of an unknown and cluttered environment with a uav. *Advanced Robotics*, 27(6):431–443, 2013.

[61] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[62] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014.

[63] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2, May 1993.

[64] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *ROBOTIK*, 2012.

[65] Christoph Rösmann, F. Hoffmann, and T. Bertram. Kinodynamic trajectory optimization and control for car-like robots. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5681–5686, 2017.

[66] Zhang Yongzhe, Benjamin Ma, and Chan Kit Wai. A practical study of time-elastic-band planning method for driverless vehicle for auto-parking. In *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, pages 196–200, March 2018.

[67] Xuexi Zhang, Jiajun Lai, Dongliang Xu, Huaijun Li, and Minyue Fu. 2d lidar-based slam and path planning for indoor rescue using mobile robots. *Journal of Advanced Transportation*, 2020:8867937, November 2020.

[68] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[69] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, May 2016.

[70] David Prokhorov, Dmitry Zhukov, Olga Barinova, Konushin Anton, and Anna Vorontsova. Measuring robustness of visual slam. In *2019 16th International Conference on Machine Vision Applications (MVA)*, pages 1–6, May 2019.

[71] Xiaomeng Li, Qiuzhan Zhou, Shaofang Lu, and Hao Lu. A new method of double electric compass for localization in automobile navigation. In *2006 International Conference on Mechatronics and Automation*, pages 514–519, June 2006.

[72] Yamaha. G30es golf cart. Website, 2015. Yamaha website for the G30Es golf cart.

[73] Peter R. Hedberg and Joy Raison. The effect of vine spacing and trellising on yield and fruit quality of shiraz grapevines. *American Journal of Enology and Viticulture*, 33(1):20–30, 1982.

[74] Richard E. Smart and Steve M. Smith. Canopy management: identifying the problems and practical solutions. In *Proceedings Second International Cool Climate Viticulture and Oenology Symposium.*, pages 316–321, 1988.

[75] Gabor Soter, Andrew Conn, Helmut Hauser, and Jonathan Rossiter. Bodily aware soft robots: Integration of proprioceptive and exteroceptive sensors. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2448–2453, May 2018.

[76] Velodyne LiDAR, Inc., Velodyne LiDAR, Inc. 5521 Hellyer Ave San Jose, CA 95138 U.S.A. *VLP-16 User Manual*, February 2019.

[77] Francisca Rosique, Pedro Navarro Lorente, Carlos Fernandez, and Antonio Padilla. A systematic review of perception system and simulators for autonomous vehicles research. *Sensors*, 19:648, 02 2019.

[78] Nancy Seckel and Adi Singh. Physics of 3d ultrasonic sensors. resreport, Toposens, 07 2019.

[79] Sebastian Dengler, Arthur Siebenhaar, Tobias Roth, and Dennis Maier. Toposens. ROS Wiki webpage, 2021.

[80] Inertial Labs, 39959 Catoctin Ridge Street, Paeonian Springs, VA 20129 U.S.A. *Single & Dual-Antenna GPS-Aided Inertial Navigation Systems INS USER MANUAL*, 2.44 edition, November 2020.

[81] LINZ. Positionz - real time service. Webpage, October 2020.

[82] LINZ. Positionz. Webpage, January 2019. The LINZ PositioNZ network consists of 39 continuously operating reference stations (CORS) located throughout New Zealand (including the Chatham Islands) and Antarctica.

[83] Philip Koopman. *Checksum and CRC Data Integrity Techniques for Aviation.* Carnegie Mellon University, May 2012.

[84] Truong Thanh. Hands-on introduction to robot operating system(ros). Web Article, November 2020.

[85] Autoware.AI. Ymc. GitHub, March 2020. The controller package for vehicles manufactured by Yamaha Motor Co., Ltd.

[86] Brian Gerkey. Amcl. Website, August 2020. created 2009.

[87] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *In Proceedings National Conference on Artificial Intelligence and Innovative Applications of Artificial Intelligence*, pages 343–349, 01 1999.

[88] Wallace Pereira Neves dos Reis, Orides Morandin, and Kelen Cristiane Teixeira Vivaldini. A quantitative study of tuning ros adaptive monte carlo localization parameters and their effect on an agv localization. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 302–307, 2019.

[89] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[90] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Machine Intelligence and Pattern Recognition*, volume 1, pages 435–461, 01 1986.

[91] Bin Liu, Zhiwei Guan, Bin Li, Guoqiang Wen, and Yu Zhao. Research on slam algorithm and navigation of mobile robot based on ros. In *2021 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 119–124, Aug 2021.

[92] Mathieu Labbe. rtabmap_ros. GitHub Webpage, 2021.

[93] David V. Lu, Dave Hershberger, and William D. Smart. Layered costmaps for context-sensitive navigation. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 709–715, Sep. 2014.

[94] Baoye Song, Guohui Tian, and F. Zhou. A comparison study on path smoothing algorithms for laser robot navigated mobile robot path planning in intelligent space. *Journal of Information & Computational Science*, 7:2943–2950, 12 2010.

[95] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[96] George Kouros. car-maneuver-recovery. GitHub, November 2016.

[97] Sebastian Pütz, Jorge Santos Simón, and Joachim Hertzberg. Move Base Flex: A highly flexible navigation framework for mobile robots. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2018. Software available at `https://github.com/magazino/move_base_flex`.

[98] Forrest Edwards. ackermansteer. GitHub Webpage, June 2019.

[99] Stefan Kohlbrecher, Johannes Meyer, and Andreas Flåten. hector_gazebo_plugins. GitHub, 2012.

[100] Open Source Robotics Foundation. Tutorial: Using gazebo plugins with ros. Webpage, 2014.

[101] Chen Wang, Jeng-Shyang Pan, Hua-Rong Xu, Jie Jia, and Zhen-Yu Meng. An improved a* algorithm for traffic navigation in real-time environment. In *2015 Third International Conference on Robot, Vision and Signal Processing (RVSP)*, pages 47–50, Nov 2015.

[102] Nate Koenig and Steve Peters. gazebo_models. Github, January 2020.

[103] TCI New Zealand. Surestore produce bins. Website, August 2021.

[104] Viscount Plastics. Horticultural industry. Website, 2021. Dual Height Crate.

Appendix A

Components

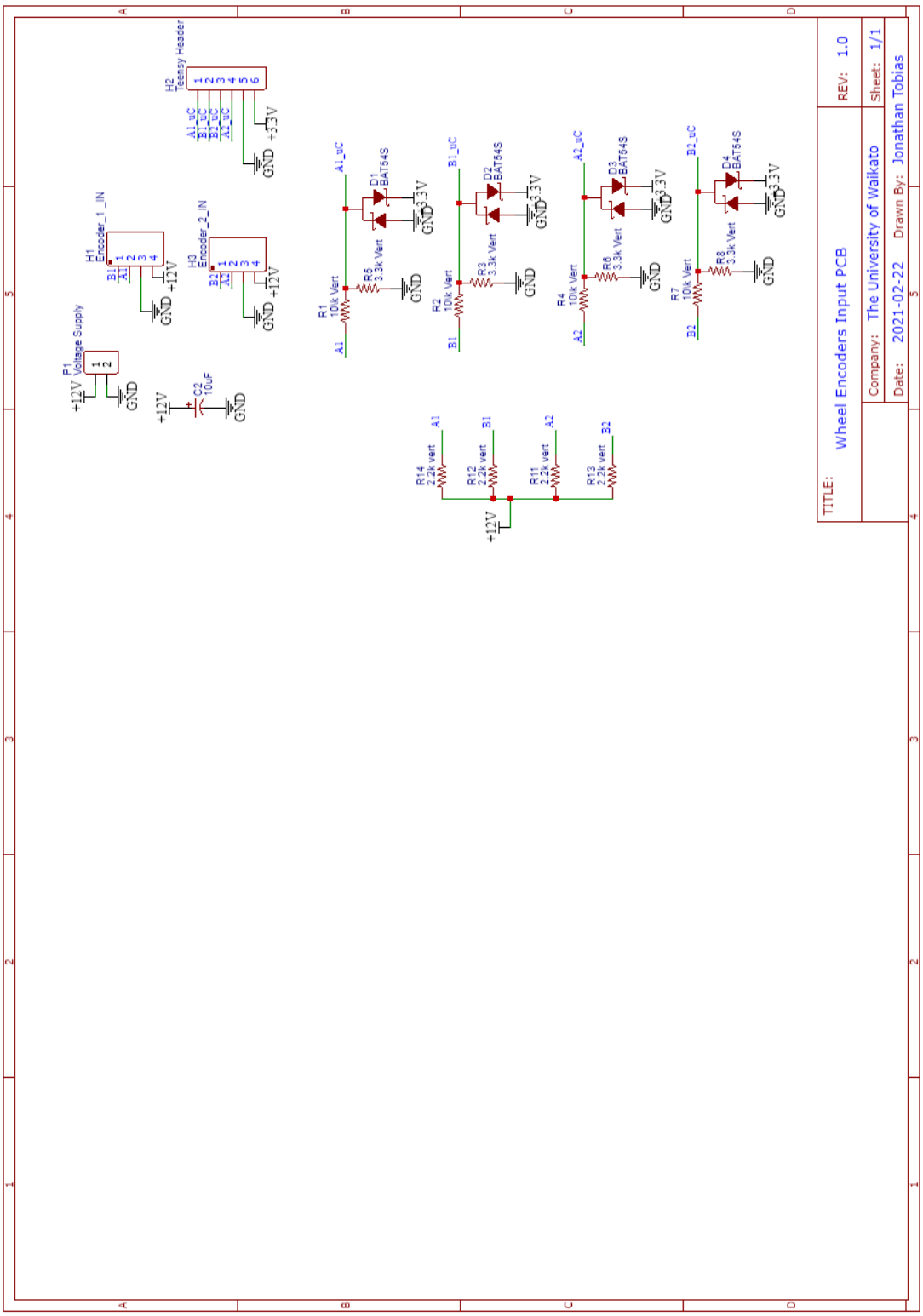


Figure A.1: Wheel encoders step-down circuit board.

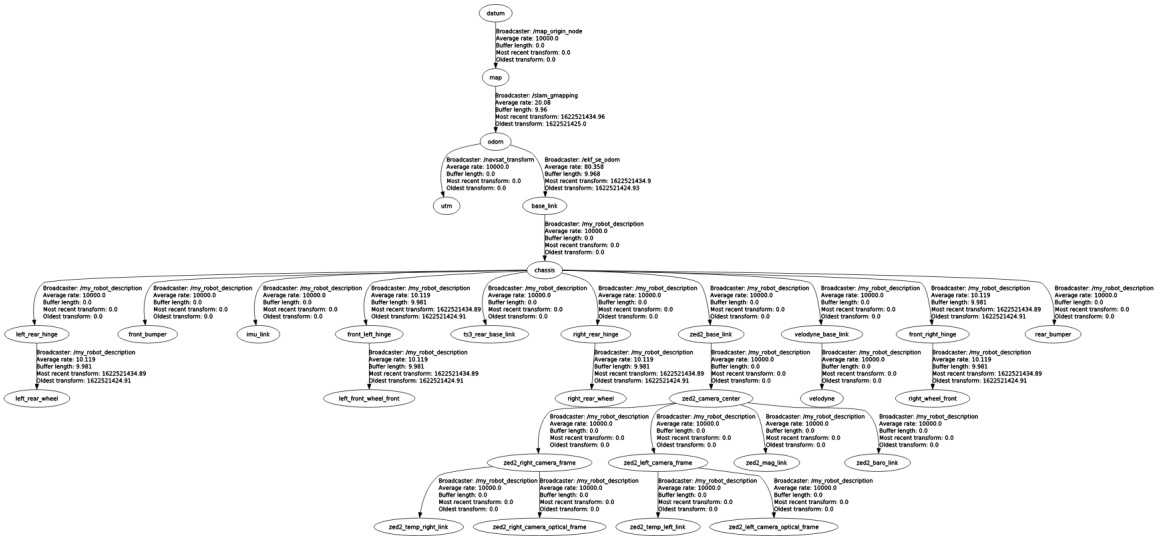


Figure A.2: Mapping systems TF tree

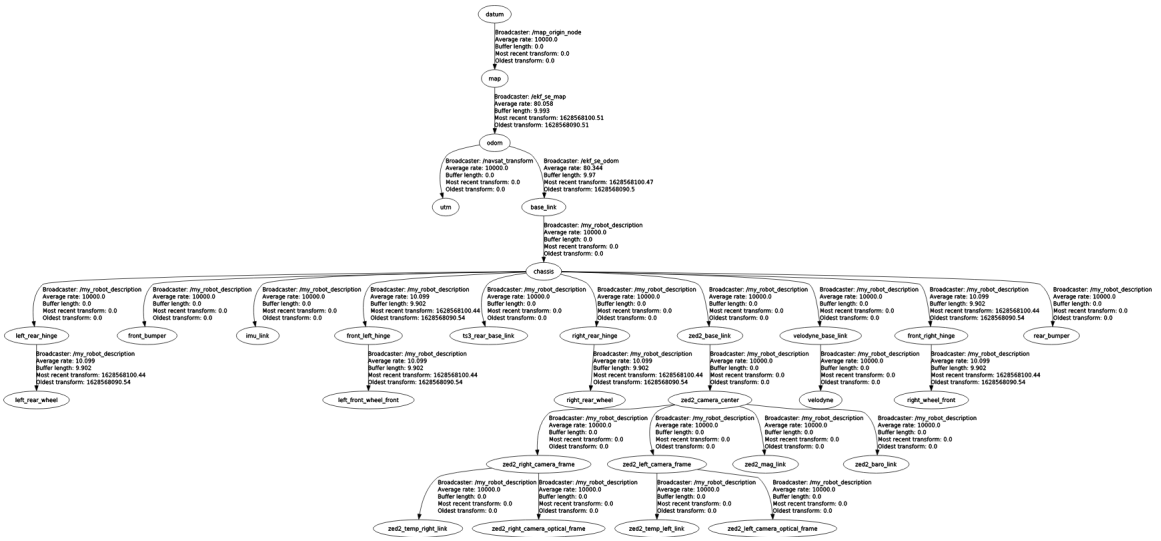


Figure A.3: Navigation systems TF tree