A Comparison Between Particle Filter-based and Graph-based SLAM Algorithms for Formula Student Artificial Intelligence

Craig Newlands

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2024

Abstract

Formula Student AI is an international autonomous racing competition where students from different universities create a driving system for an autonomous vehicle and compete against each other across a range of events, with no prior knowledge of the track. Simultaneous Localisation and Mapping, SLAM, is a method of localising a vehicle in an unknown environment while creating a map of this environment. SLAM is an ideal candidate for use in the Formula Student AI events to localise the vehicle while generating a map of each track. This thesis aims to compare particle filter-based and graph-based SLAM algorithms in the context of Formula Student AI. To achieve this goal, FastSLAM 2.0 and GraphSLAM have been implemented and tested on real-world track data and compared across a variety of metrics. Furthermore, additional cone colouring has also been implemented into the GraphSLAM algorithm. The results show the GraphSLAM algorithm outperforms the FastSLAM 2.0 algorithm and could be used to map the tracks during Formula Student AI events.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Craig Newlands*)

Acknowledgements

Many thanks to my supervisor James Garforth for providing invaluable support and guidance throughout this thesis.

My thanks also go to the previous and current members of the Edinburgh University Formula Student AI team, without their contribution to the team this thesis would not have been possible. A special mention to Angus Stuart who contributed massively to the infrastructure and previous SLAM algorithms implemented in the software stack, as well as the insightful SLAM-related discussions and suggestions.

Finally, I would like to thank my family and friends for their support in everything that I do.

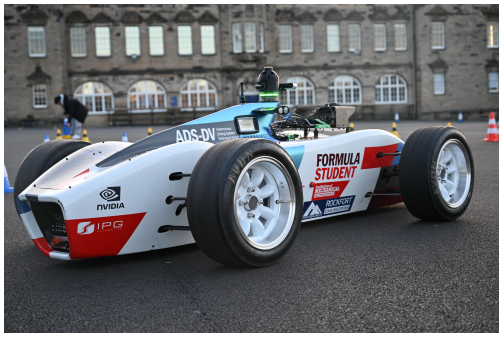
Table of Contents

Chapter 1

Introduction

Autonomous vehicles have been a topic of interest in recent years to improve road safety by making use of advancing technologies that are being made available, such as advanced lidar and camera systems. From June 2022 to June 2023 in Great Britain alone 29,429 people were killed or seriously injured in road accidents [39], where it is thought 90-95% of these fatalities were down to human error [31]. Autonomous systems have the potential to reduce these numbers and many manufacturers are already making use of autonomous systems, in the form of Advanced Driver Assistance Systems (ADAS) [34], in their cars to improve safety. ADAS makes use of lidar, radar, and camera systems to implement features such as pedestrian detection/avoidance, lane departure warning/correction, and automatic emergency braking. All of which improve road safety. Companies such as Oxa Technologies [30] and Waymo [42] are taking this a step further and are trying to implement fully autonomous systems where no human driver is required at all. To speed up the research and development of autonomous systems many autonomous car competitions have been created over the years, the first being DARPA in 2004 [8] and more recent competitions include Indy Autonomous Challenge [14] and Formula Student AI [15]. Pictures of an Indy Autonomous Challenge car and Formula Student AI Autonomous Driving Systems – Dedicated Vehicle (ADS-DV) can be seen in Figure 1.1.

(a) Indy Autonomous Challenge Car 2024 [14].



(b) IMechE Formula Student AI ADS-DV.

Figure 1.1: Example of vehicles in autonomous car competitions.

1.1 Formula Student AI

Formula Student AI is an annual autonomous vehicle competition organised by the Institution of Mechanical Engineers (IMechE), held at Silverstone Race Track. It was first introduced in 2017 and is a competition for university students to develop a system to drive a fully autonomous vehicle across a range of dynamic events, as well as compete in static events such as design, business, and real-world AI presentations. There are two competition classes within Formula Student AI that teams can compete in; the Dynamic Driving Task (DDT) category and the Automated Driving System (ADS) category. For the DDT category, the IMechE provides a vehicle, the ADS-DV, for the teams to fit their sensors onto and run their software stack on. For the ADS category, teams must build their own vehicle fitted with their sensors and run their software stack on it.

The dynamic events include Skid Pad, Acceleration, Autocross, and Trackdrive. Each has a different configuration specified in the FS2024 AI Rules [3]. Skid Pad is a figure of eight; the vehicle enters perpendicular to the figure of eight and must complete two loops of each side before exiting opposite the entrance point, where it must stop in a bounded box. Acceleration is a 75-metre straight with a bounded box at the end where the autonomous vehicle must stop. Autocross is a single lap of an unknown track where the autonomous vehicle must stop within 30 metres after the finish line. Trackdrive is similar to autocross except 10 laps must be completed before stopping within 30 metres after the finish line. This thesis will focus on the dynamic Trackdrive event, with more details following in Section 1.1.1.

The track boundaries for the dynamic events are determined by coloured cones, small blue cones to the left, and small yellow cones to the right. Small orange cones indicate the entry and exit boxes and big orange cones indicate the start and finish lines. The maximum distance between cones is 5 metres. The different cones used in Formula Student AI can be seen in Figure 1.2.

Figure 1.2: Big orange, small orange, small yellow, and small blue cones used for Formula Student AI dynamic events [32].

1.1.1 Trackdrive

As mentioned in Section 1.1, this thesis will focus on the dynamic Trackdrive event. Trackdrive involves autonomously navigating an unknown track for 10 laps. No prior knowledge of the track can be used by the autonomous system, meaning any knowledge of the track must be learned during each attempt. The system must autonomously count the number of laps and stop within 30 metres after the finish line.

The exact configuration for the Trackdrive event, specified in the Formula Student AI Rules [3], is as follows:

- No straights longer than 80 metres
- Constant turns can have a diameter of up to 50 metres
- Hairpin turns must have a minimum of 9 metres outside diameter
- Track may have chicanes, multiple turns, decreasing radius turns, etc.
- The minimum track width at all points is 3 metres
- The maximum distance between track boundary cones is 5 metres

Figure 1.3 gives a visual representation of this configuration around the start/finish line; showing the start position, the stop area, the distance between cones, the start/finish line, the track width, and the cone configuration.

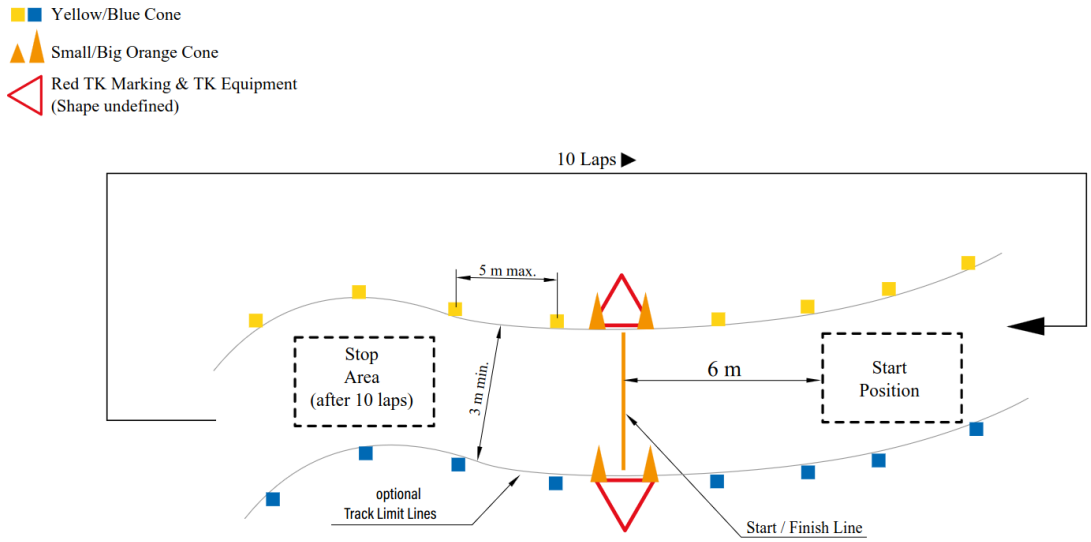


Figure 1.3: Trackdrive configuration. Figure from Formula Student Germany Competition Handbook 2024 [4].

The event is scored based on the time to complete each lap, including any penalties received. A 2-second penalty is awarded for knocking over cones or moving a cone enough that its base is outside the perimeter of its starting base position. A 10-second penalty is awarded for all four wheels of the vehicle exiting the track boundary. Points are awarded based on elapsed time, including penalties, and an additional 10 points are awarded for each successfully completed lap. Finally, a 50-point deduction is awarded for an unsafe stop, where the vehicle does not stop within the track boundary or it has not entered the 'AS Finished' state, which indicates the vehicle is safe to approach.

1.1.2 ADS-Dedicated Vehicle

Section 1.1 briefly mentions the IMechE-provided Formula Student AI ADS-DV, seen in Figure 1.1b. Teams must provide their own AI computer which communicates with

the Vehicle Control Unit (VCU) using the CANbus communication protocol through an umbilical connector which also provides power to the AI computer and sensors. The ADS-DV comes fitted with a stereo camera that teams can use for autonomous driving. Teams also have the option to fit additional sensors to the vehicle as long as they are approved by the IMechE and are mounted securely within the regions shown in Figure 1.4 or within 100mm of the bodywork.

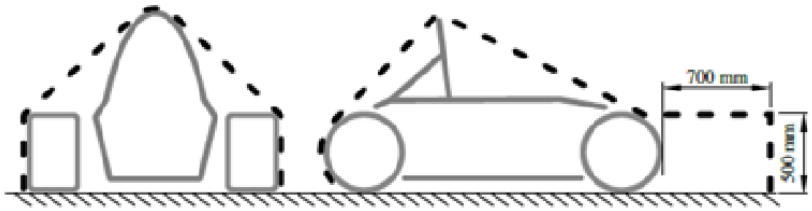


Figure 1.4: Sensor mounting locations [3].

The experiments conducted in this thesis were carried out on the ADS-DV with a similar sensor plate to that in Figure 1.5; where a stereo camera, inertial measurement unit (IMU) sensor, and lidar can be seen mounted in the triangular region in front of the ADS-DV 'shark fin' that can be seen in Figure 1.1b.

Figure 1.5: Edinburgh University Formula Student sensor plate.

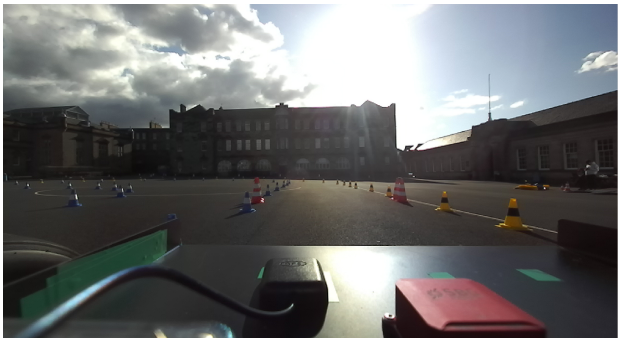


Figure 1.6: Edinburgh University Formula Student example camera image.

The restricted sensor mounting locations pose an additional challenge when receiving sensor data. Figure 1.6 shows the restricted view the camera has of the track. This is due to the low profile of the sensor plate, required to fit in the zones outlined in Figure 1.4, and the obstruction from the wheels. Due to the limited field of view, it can be hard to differentiate between different points on the track. As the track layout consists of

the same coloured cones on each side of the track, as mentioned in Section 1.1.1, there may be multiple points on the track that produce very similar images. For example, two different straights may look the same, only being able to see a few blue cones on the left and a few yellow cones on the right.

1.2 SLAM

To achieve the fastest lap time and the most number of points in the Trackdrive event, the autonomous system must navigate the vehicle accurately and quickly around the track following the optimal racing line. Racing at higher speeds, to achieve the optimal lap time, increases the noise error in the sensor measurement. This makes it harder to detect cones, giving the system less time to react to the twists and turns of the track layout. Therefore only using sensors to navigate the track is not sufficient. These limitations can be overcome by mapping the track on the first lap of the Trackdrive event and maintaining the location of the vehicle within this map. The trajectory can be used for optimising the racing line for the proceeding 9 laps of the Trackdrive event. Without the detection of the finish line, the first complete lap cannot be determined and the optimisation cannot be applied.

The problem of autonomously creating a map of an unknown environment has been a topic of interest in research for many years. SLAM is one solution to this problem and focuses on creating a map of an unknown environment while simultaneously localising within it. A basic mapping algorithm is not sufficient for the Trackdrive event due to the navigational precision that is required. Creating a map of the track by solely taking cone measurements from the sensors on the vehicle and transforming them to the global frame requires positional accuracy of the vehicle that cannot be achieved with a standard Global Positioning System (GPS). Standard GPS has an accuracy of 4.9 metres in open space [40], this is not sufficient for Formula Student AI events where the track width is 3 metres. This means a more complex solution is required, such as SLAM.

The application of Formula Student's Trackdrive event poses a unique opportunity to test SLAM in the context of autonomous racing. In most autonomous racing competitions, such as Indy Autonomous Challenge [14] and DARPA [8], the track is known prior to racing. This is not the case for Formula Student's Trackdrive event where the track is unknown, making SLAM more applicable.

1.3 Motivation

Edinburgh University Formula Student AI (EUFS-AI) has had great success in Formula Student AI since first competing in 2018, where we won the DDT class. Since then we have won Formula Student AI five years in a row and are hoping for another year of success in 2024. Over the years we have used various SLAM algorithms, but have not had much success with them at competition; until last year when our FastSLAM 2.0 algorithm played a vital role in our Trackdrive event win at the competition. Over the years Formula Student AI has been getting more popular and teams are gradually producing more competitive software, meaning at EUFS-AI we must continue to

develop our software stack and explore areas of improvement. There are different families of SLAM algorithms, that excel in different applications, FastSLAM 2.0 and the other SLAM algorithms we have previously implemented at EUFS-AI are all from the family of particle filter-based SLAM algorithms, so choosing a SLAM algorithm from a different family of algorithms, such as graph-based SLAM algorithms, may lead to improvement in performance. More details on particle filter-based and graph-based SLAM algorithms will follow in Section 2.3.2.

This thesis sets out to implement GraphSLAM, a graph-based SLAM algorithm, that will be compared with our current FastSLAM 2.0, a particle filter-based SLAM algorithm, implementation at EUFS-AI. An improved algorithm will output a more accurate map and trajectory. This is particularly useful for the Trackdrive event, where an accurate map allows the planning and control pipeline to produce an optimal racing line based on the track layout, achieving faster lap times and receiving minimal penalties.

1.4 Aims

The work discussed in this thesis explores how particle filter-based SLAM algorithms and graph-based SLAM algorithms behave under the requirements of Formula Student AI. Specifically, the main aims of this thesis can be summarised as follows:

1. To implement a graph-based SLAM algorithm into the EUFS-AI software stack.
2. To determine which SLAM algorithm, either particle filter-based or graph-based, consistently produces more accurate maps.
3. To identify which SLAM algorithm, either particle filter-based or graph-based, has a smaller computational latency.
4. To recognise any challenges for SLAM algorithms in Formula Student AI.

To explore these aims the graph-based SLAM algorithm that will be implemented will be GraphSLAM and the compared particle filter-based SLAM algorithm will be FastSLAM 2.0. The algorithms will be evaluated on real-world data sets similar to those that can be found at the Formula Student AI competition. All data sets were collected by the EUFS-AI team using the ADS-DV, seen in Figure 1.1b.

1.5 Thesis Outline

The rest of the thesis will be laid out as follows. Chapter 2 explores the theoretical background of SLAM and the two algorithms of focus. It also includes a literature review on SLAM evaluation methods. Chapter 3 describes the wider Formula Student AI software stack and the SLAM implementation details. Chapter 4 explains the data collection procedure and the evaluation approach. Chapter 5 presents and evaluates the results from the experiments. Chapter 6 concludes with a summary of the thesis and potential future work.

Chapter 2

Background and Related Work

This chapter will explore the theory behind particle filter-based SLAM algorithms and graph-based SLAM algorithms, highlighting the specific algorithms that will be explored in this thesis. Following this is a review of previous literature that discusses common evaluation methods for comparing SLAM algorithms. Finally, a review of literature specific to the application of SLAM in Formula Student will be conducted, highlighting different evaluation methods to compare different SLAM algorithms.

2.1 SLAM History

The idea of creating a map and simultaneously localising within that map was first introduced in 1991 by Leonard and Durrant-Whyte [20]. The term *SLAM* did not come around until 1996 in a paper about localising autonomous guided vehicles by Thrun and Montemerlo [9]. At this time all SLAM algorithms used Extended Kalman Filters or paticle methods for estimating position. A year later in 1997, Lu and Milios introduced a graph-like representation with links between different poses and formulated an optimisation algorithm to globally optimise the graph [24]. This eventually led to the development of GraphSLAM in 2006 by Thrun and Montemerlo [36]. GraphSLAM does not make use of the traditional probabilistic approach, instead introducing soft constraints and then optimising these constraints. The following year, in 2007, another breakthrough in SLAM development was made by Georg Klein and David Murray. They presented Parallel Tracking and Mapping (PTAM) [17] which explores the idea of splitting the tracking and mapping in Augmented Reality systems. This was a significant breakthrough, showing that the mapping does not need to be done live, simultaneously with the tracking. Meaning the optimisation of the map can be done separately, when computational resources are less sparse, resulting in better optimisations.

2.2 Mapping and Localisation

As mentioned in Section 1.2 achieving the optimal lap time for the Trackdrive event requires a map of the track and the position of the vehicle in this map. With this information, the optimal path around the track can be created and the fastest lap times

achieved. Mapping the track and localising within this map are the fundamentals behind SLAM.

Mapping aims to produce a map of the track using landmarks to represent the environment. Landmarks are features that are extracted from the environment using sensor measurements, such as lidar data and image data from cameras. Due to sensor limitations, unless the track is very small, not all landmarks can be identified in a single lidar point cloud or image frame. Landmarks can be artificial, where they are placed in the environment to help the robot localise, or natural, where they already exist in the environment. The quality of a landmark is determined by how easily it can be identified by the robot's sensors. If a landmark can be identified across multiple frames then it is likely that this same landmark will be easily detected if the traversing robot returns to the same area. In the context of the Trackdrive event, the map is static as the landmarks are the cones that are set out to represent the track boundary, as seen in Figure 2.1. These are artificial and not natural landmarks.

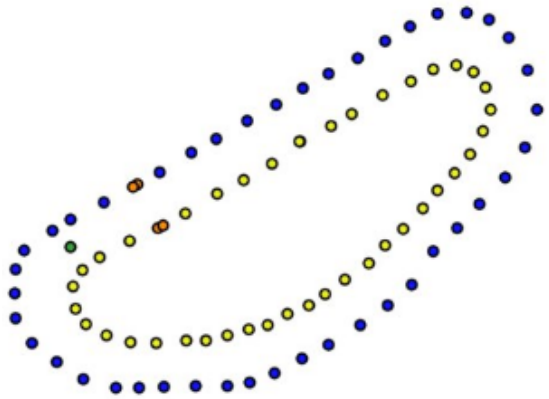


Figure 2.1: Example map with cones for landmarks.

Localisation aims to find the position of the vehicle in the environment. For this, we require a map of the environment. For localising within a map there are two frames of importance; the map frame and the vehicle frame, as seen in Figure 2.2. The map frame is the global coordinate system in which the landmarks and the pose of the vehicle are described. This is independent of the vehicle position. The vehicle frame is the coordinate system attached to the vehicle and is used to describe the detection of landmarks. There are different types of localisation; tracking, where the initial position is known, global localisation, where the initial position is unknown, and kidnapped robot problem, where the robot can teleport anywhere at any time. The Formula Student AI application focuses on tracking, as the initial position of the vehicle is known on the track. Localisation works by considering the robot's motion between timesteps, and how the perceived environment has changed in this time to identify its location in the map.

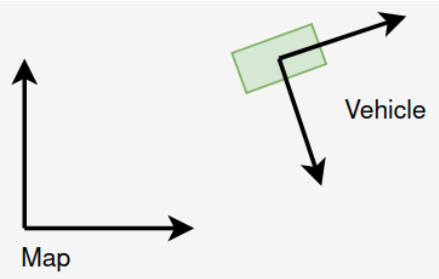


Figure 2.2: Frames of reference.

2.3 SLAM Properties

SLAM problems can be categorised based on different properties. This section will explore the main properties covered in this thesis.

2.3.1 Online SLAM and Full SLAM

Whether a SLAM solution has the property of online SLAM or full SLAM is determined by the approach used for handling data.

The online SLAM problem focuses on estimating the map and only the current pose of the vehicle. Some online SLAM algorithms discard older sensor data focusing solely on more recent data [38] to allow for quick adjustments to changing environments and improve computational efficiency. These characteristics make online SLAM a suitable candidate for the Trackdrive event where real-time responses are required to navigate the unknown track layout and only the current position of the vehicle in the track is desired. However, as the estimates are made based on recent observations and not all observations, any previous errors that have not been accounted for are propagated into future estimates, accumulating and potentially resulting in worse performance on longer or more complex tracks.

On the other hand, the full SLAM problem focuses on estimating the map and all vehicle poses. No previous data is discarded and all knowledge about the map and poses is used for making estimates [38]. This improves accuracy as all available data is considered, giving a global solution to the problem. This makes full SLAM a suitable candidate for the Trackdrive event where multiple laps are completed on the same static track meaning a global optimisation can be created and utilised to remove errors and improve the accuracy of navigation over the 10 laps. However, due to full SLAM making use of all knowledge about the map and the vehicles poses additional computation is required. This could lead to an increase in computational latency, introducing delays and reducing real-time responsiveness.

2.3.2 Particle Filter-Based SLAM and Graph-Based SLAM

Particle filter-based SLAM and graph-based SLAM are two of the most common approaches to solving the SLAM problem mentioned in Section 1.2. Both excel in different scenarios depending on the characteristics of the application.

Particle filter-based SLAM generates a collection of particles, each representing a different state, which includes the vehicle's position and the map of the track. As the vehicle is traversing around the track the particles get assigned weights based on the accuracy of the predicted state compared to the sensor measurements. After new weights are assigned to each particle, only the particles with the highest weights are maintained, this is known as resampling. This process is repeated allowing the vehicle to traverse the track following the most probable path, whilst creating a map of the track [38]. As the Trackdrive track can have sharp twists and turns as well as long straights, the vehicle can travel at varying speeds. These non-linear movements allow for particle filter-based SLAM algorithms to excel due to their probabilistic nature.

Graph-based SLAM constructs a graph of the track where nodes represent the vehicle pose or landmarks (cones) and edges represent sensor measurement encoding between two nodes, these are constraints. The graph is then optimised to find the best configuration of nodes that aligns with the real-world sensor measurements [11], resulting in the optimal map and vehicle trajectory. Graph-based SLAM can also utilise the properties of loop closure, this is when nodes that we have previously added to the graph are observed again, allowing any drift to be corrected. This makes graph-based SLAM particularly useful for the Trackdrive event where we complete 10 laps of the same track, seeing the same landmarks/cones on the track multiple times, allowing any drift to be corrected and a more accurate map and trajectory to be achieved.

2.4 SLAM Theory

As stated in Section 1.2, SLAM addresses the problem of estimating the map of the track whilst also estimating the position of the vehicle in this map. Probabilistically this can be formulated in two different ways; the first being the online SLAM problem and the second being the full SLAM problem, both outlined in Section 2.3.1. Before going into the details of these SLAM problems, some important variables need to be defined:

x_t represents the vehicle state at time t. The state is made up of the location, x and y coordinates, of the robot and the orientation θ. Both are in the map frame.

m_i represents the location of cone i in the map frame. The map m is all the landmarks that make up the track.

u_t represents the control data and the change of state during time $(t-1;t]$. The sequence of control data from time t_1 to t_2 is represented as $u_{t_1:t_2}$.

z_t represents the measurements from the sensors present on the vehicle. For notational simplicity, the rest of this section assumes one measurement per timestep but this is not the case for the implemented algorithms or the rest of this thesis. The sequence of measurements from time t_1 to t_2 is represented as $z_{t_1:t_2}$.

2.4.1 Online SLAM Formulation

The online SLAM problem can be defined as the joint posterior probability density of the current vehicle state and the map given the vehicle state at time $t = 0$, along with

the measurements and control inputs.

$$p(x_t, m | z_{1:t}, u_{1:t}, x_0) \tag{2.1}$$

The joint posterior of the online SLAM problem can be estimated using the Bayes Filter, as seen in Algorithm 1. The key idea of the Bayes Filter is to maintain a belief of the state, which is updated over time with new observations.

Algorithm 1 Pseudocode for the Bayes filter [38]

1: **function** BAYES_FILTER($bel(x_{t-1}, m), u_t, z_t$)
2: **for** all x_t **do**
3: $\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx$
4: $bel(x_t) = \eta p(z_t | x_t) \overline{bel}(x_t)$
5: **end for**
6: **return** $bel(x_t)$
7: **end function**

There are two main steps to the Bayes Filter; the prediction step and the update step. The prediction step aims to predict the next pose of the vehicle based on the control inputs (u_t), this is line 3 in Algorithm 1. The update step aims to correct the predicted pose based on the measurement models ($p(z_t|x_t)$), this is line 4 in Algorithm 1. Measurement models describe the probability of the measurement given the current state of the vehicle.

The Bayes Filter is underlined by the Markov assumption, which states that if the current state of the vehicle (x_t) and the map (m) are known, then the previous control inputs ($u_{1:t}$) and measurements ($z_{1:t}$) do not affect our belief about the state. The effect of this assumption can be seen in Figure 2.3. This assumption may seem extreme but works surprisingly well in real work applications as a large amount of information about the state at time t is held within the state and the recent control inputs.

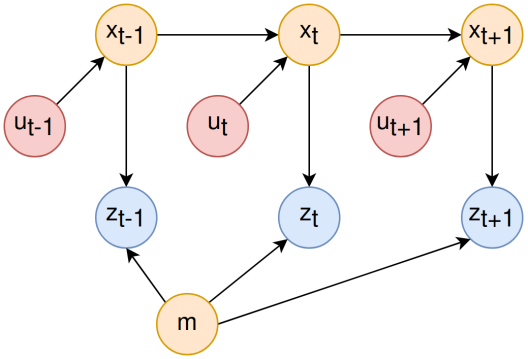


Figure 2.3: Hidden Markov model showing the relationship between the vehicles states x, the controls u and the measurements z over time. [38]

Due to the complexity and the dimensionality of the state space, it would be too computationally expensive to estimate the exact probability distribution given in Equation

2.1. Therefore, we must approximate the Bayes Filter and one approach to this is the Particle Filter. Particle filters estimate the pose distribution using many particles, each representing a 'guess' at the vehicle's pose. A location with more particles is more likely. There are three steps involved in the particle filter:

1. **Update step** involves updating each particle based on the motion model, this typically introduces noise to incorporate the uncertainty in the vehicle motion.
2. **Weighting step** assigns a weight to each particle based on the number of observations. Higher weights mean a better match between the perception systems and the measurement model.
3. **Resampling step** resamples the particles with replacement proportional to the weight of each particle. The weights are reset after sampling.

These steps are repeated updating the state of the vehicle. A particle filter-based SLAM algorithm will be explored in Section 2.5.

2.4.2 Full SLAM Formulation

The full SLAM problem can be defined as the joint posterior probability density of all vehicle states and the map given the vehicle state at time $t = 0$, along with the measurements and control inputs [38].

$$p(x_{1:t}, m | z_{1:t}, u_{1:t}, x_0) \tag{2.2}$$

It is also not feasible to compute the full joint posterior, given in Equation 2.2, in real-time due to the size of the state space. Therefore, approximation methods are required. One approach to this is a graph-based method, where nodes represent landmarks and vehicle positions and edges represent constraints between sensor measurements and vehicle motions, as discussed in Section 2.3.1. Typically these constraints are optimised using least-square to minimise the error introduced by the constraints [11]. A graph-based SLAM algorithm will be explored in Section 2.6.

2.5 FastSLAM 2.0

FastSLAM 2.0 was first introduced in 2003 by Montemerlo and Thrun [26] to improve on its predecessor, FastSLAM 1.0. FastSLAM 2.0 uses particle filters for estimating the optimal vehicle path. Each particle contains an estimated vehicle state and an Extended Kalman Filter (EKF) for each landmark in the map, where each EKF updates the landmark's position relative to the particle's pose. The steps involved in FastSLAM 2.0 can be seen in Algorithm 2.

FastSLAM 2.0 was chosen to represent the class of particle filter-based SLAM algorithms in this thesis as it has been proven by EUFS-AI to solve the SLAM problem efficiently in the context of Formula Student AI. Due to the use of particles, FastSLAM 2.0 does not have to explore the entire state space, improving its computational efficiency. In Formula Student AI, being computationally efficient is a key property to

Algorithm 2 Single time step of the FastSLAM 2.0 algorithm

for each particle **do**

Prediction: predict new state from motion model and control inputs

Data Association: associate the observed landmark with an existing landmark on the map or add a new landmark to the map

Update: update the local map with the measurement

Weight: compute the new weight of the particle based on the number of observations

Resampling: resample M particles based on their weights, where larger weights are assigned higher probabilities, keeping the most accurate particles

ensure the algorithm does not lag behind the real-time movements of the autonomous vehicle. The real-time updates of the vehicle state and map are also crucial for autonomous racing to make immediate decisions when navigating the track.

FastSLAM 2.0 is the probabilistic SLAM algorithm of choice in this thesis as it has already been implemented by EUFS-AI and has shown good results for this application.

2.6 GraphSLAM

GraphSLAM solves the full SLAM problem by using all the poses and landmarks in the map. It works by first generating a sparse graph of nodes and edges, where nodes represent the pose of the vehicle or a landmark and edges represent nonlinear quadratic constraints. This can be seen in figure 2.4. Through constraint optimisation, a maximum likelihood map and a corresponding set of vehicle poses are generated [36].

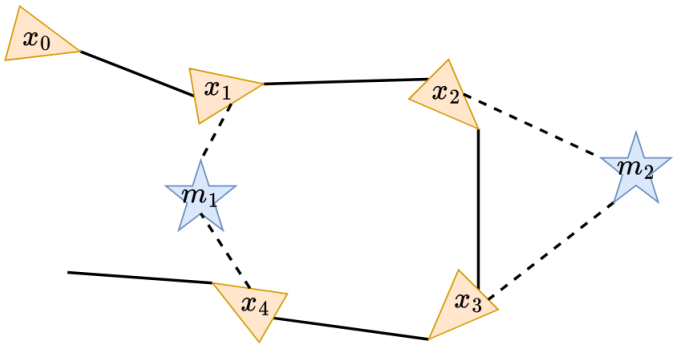


Figure 2.4: GraphSLAM graph showing 5 poses $(x_0, ..., x_4)$, 2 map features (m_1, m_2), motion constraints (solid lines) and measurement constraints (dashed lines). [36]

In GraphSLAM there are two types of constraints; motion edges and measurement edges. Motion edges join two consecutive vehicle poses and measurement edges join

vehicle poses to landmarks. These constraints are the negative log-likelihood of the measurement and motion models. Therefore, minimising the sum of the constraints gives the most likely map and the most likely vehicle path [38].

The sum of constraints, J, for the example graph in figure 2.4 would be:

$$\begin{aligned} J = & x_0^\top \Omega x_0 + \\ & \sum_t [x_t - g(u_t, x_{t-1})]^\top R^{-1} [x_t - g(u_t, x_{t-1})] + \\ & \sum_t [z_t - h(m_{c_t}, x_t)]^\top Q^{-1} [z_t - h(m_{c_t}, x_t)] \end{aligned} \tag{2.3}$$

Where the three terms represent the constraints of the initial pose x_0, the constraints of the motion model, and the constraints of the measurement model, respectively.

The steps involved in the GraphSLAM algorithm can be seen in Algorithm 3.

Algorithm 3 GraphSLAM algorithm

Initialisation: initialise an empty graph for the poses and constraints

Graph construction: add nodes for vehicle poses and landmarks to the graph, create edges for constraints between nodes based on sensor data, the motion model, and the measurement model

Optimisation: minimise the sum of constraints to find the optimal vehicle path and landmark positions

In this thesis, the g^2o library will be used for graph optimisation [18].

In the context of Formula Student AI, if loop closure is achieved we can obtain a complete map of the track after one full lap. Therefore, using a SLAM algorithm that solves the full SLAM problem, such as GraphSLAM, seems desirable. The environment is also static, with a fixed number of landmarks. This means after a landmark has been added to the map it is unlikely to change position, keeping our graph up to date.

2.7 Literature Evaluating SLAM

This section explores the literature encompassing SLAM evaluation methods, first identifying possible evaluation metrics and then moving on to comparisons between SLAM algorithms on baseline datasets.

Zhang *et al.* presents metrics to evaluate the quality of trajectory estimates in visual(-inertial) odometry systems [43]. The two error metrics explored are the Absolute Trajectory Error (ATE) and the Relative Error (RE). The ATE measures the difference between the ground truth trajectory and the estimated trajectory in the same reference frame. It is calculated by aligning the estimated trajectory to the ground truth trajectory and then computing the root mean squared error (RMSE) for all positions in the estimated trajectory and ground truth trajectory. ATE may be sensitive to outliers

due to the alignment process; if the two trajectories are not aligned properly then the RMSE between all positions will be much larger. The RE measures the accuracy of the trajectory estimate by splitting the trajectory into segments and examining the relative motion between these segments. It is calculated by picking pairs of points on the trajectory, either a certain distance apart or over a certain time interval, to create segments. These segments are extracted from both the estimated and true trajectory and a relative transformation is computed. The difference in the relative transformation across all segments is the error. This is more complex to compute than the ATE as it involves aligning multiple segments and then computing a distribution over these segments but it is more robust to outliers as there is no global alignment.

Campos *et al.* presents ORB-SLAM3, one of the latest cutting-edge SLAM algorithms, which at its core is a graph-based SLAM system [6]. The algorithm is compared across different configurations on metrics such as RMSE of ATE, scale error and running time. The RMSE of ATE metric is used as described above in the paper by Zhang *et al.* [43]. The scale error measures the difference between the true scale and the estimated scale. First, the scale factor needs to be computed by aligning the estimated trajectory with the ground truth trajectory and taking this transformation to determine the scale factor. The scale factor, s, is then used to calculate the difference between the estimated and true scale using this equation $\textit{Scale error} = |1 - s| \times 100\%$. The running time measures the time to complete different parts of the algorithm, for example, the time to complete feature extraction, pose estimation, local and global optimisation steps, and map merging.

For evaluating SLAM algorithms there are two main data sets used consistently across the field. Nieto *et al.* [28] introduced these benchmark data sets which give sensor data for two different outdoor environments navigated using a utility vehicle. The vehicle was fitted with additional sensors, such as GPS for evaluating ground truth, dead reckoning sensors, and a laser range sensor. The first data set was collected at the top of a parking lot for better satellite-to-GPS communication and used 60mm steel poles covered with reflective tape for landmarks. The second data set was collected in Victoria Park and made use of natural landmarks such as trees.

Thrun *et al.*[37] compared FastSLAM 1.0 with EKF SLAM and FastSLAM 2.0 on the benchmark data sets mentioned above. The root mean square (RMS) vehicle pose error is computed for different algorithms and is the difference between the vehicle pose from the SLAM output and the true ground truth pose. Convergence speed is the rate at which the algorithm processes the sensor measurements, updating its estimate of the map and vehicle pose. The accuracy of each algorithm was computed using 100 landmarks. For FastSLAM 1.0 the RMS vehicle pose error was calculated over a various number of particles ranging from 1 to 5000. The scaling performance of FastSLAM 1.0 is evaluated using the computation time for 500 sensor updates over a varying number of landmarks. Memory usage for FastSLAM 1.0 was also presented to show the effect of increasing the number of landmarks. Varying odometric noise was added to the data to compare its effect on the vehicle position. The RMS vehicle pose error, for FastSLAM 1.0 and EKF SLAM, was computed over 4 different noise values. The RMS vehicle pose error is computed for FastSLAM 1.0 and 2.0 on the simulated data set with varying levels of measurement noise and on the simulated data

set and Victoria Park data set with varying numbers of particles. Scaling performance for FastSLAM 1.0 and 2.0 is also computed. The RMS vehicle pose error is computed for FastSLAM 1.0 and 2.0 on varying-sized loops to evaluate the performance of loop closing. The convergence speed between FastSLAM 2.0 and EFK SLAM was also calculated and evaluated by calculating the RMS landmark error over time. FastSLAM 2.0 was evaluated on 1, 10, and 100 particles. These experiments were all repeated multiple times and error bars were computed where applicable.

Li *et al.* presents a comparison between different GraphSLAM optimisation algorithms [21]. The three data sets used are the City1000, M10000, and Manhattan3500; which are all commonly used SLAM data sets. Various metrics are used to evaluate convergence speed, sensitivity to noise, and robustness to outliers. Convergence speed is evaluated by recording the time taken for each algorithm to minimise the total error in the SLAM graph. Sensitivity to noise is evaluated by adding Gaussian noise to initial poses and constraint measurements before computing the RMSE between the ground truth trajectory and the estimated poses, the sum of the squared differences between the true measurements and the estimated measurements, and the convergence speed. Robustness to outliers is evaluated by adding false positive loop closures and measuring the RMSE between the ground truth trajectory and the estimated poses.

The papers above all shared different metrics for evaluating a range of SLAM algorithms, some more modern like ORB-SLAM 3 and some more traditional like FastSLAM and GraphSLAM. Although these SLAM algorithms are from different eras in SLAM history they still share common evaluation metrics such as root mean squared error of vehicle pose, levels of noise and algorithmic speed. The higher-quality papers repeat experiments to show reliability and validity and compute error bars to show variability.

2.8 SLAM in Formula Student

SLAM plays an influential role in the autonomous Formula Student competition for traversing and mapping the track. Many teams adopt SLAM to solve the localisation and mapping problem. One of the first teams to produce a paper on their autonomous system was AMZ, students of ETH Zurich [16]. For the 2018 competition, they chose to implement FastSLAM 1.0 over other SLAM algorithms as they believed it best suited the rules and requirements of Formula Student AI, as outlined in Section 1.1. They believed the chosen algorithm had to be real-time to compute the latest sensor data and easily tunable to reach its full potential quickly due to limited testing time. For AMZ, a particle-based approach ticked these boxes and they continued to follow this belief into the 2019 competition, upgrading from FastSLAM 1.0 to FastSLAM 2.0.

Due to the popularity of SLAM within the Formula Student community, finding the optimal SLAM algorithm can determine your success at the competition. Therefore, over recent years many teams have produced papers comparing SLAM algorithms on a variety of evaluation metrics.

2.8.1 Formula Student Literature Evaluating SLAM

Lopes presents a comparison between FastSLAM 1.0, FastSLAM 2.0 and GraphSLAM in the context of Formula Student [23]. The algorithms are tested on simulation and real-world data gathered by Lopes. On both data sets, the RMSE of the cone positions between the ground truth and the SLAM map are computed and compared across all three algorithms. Also, on the real-world data, the RMSE per cone class as a function of the distance travelled is computed for all three algorithms. Both real-world data experiments are conducted on two different tracks. Qualitative analysis is carried out to compare the localisation results by illustrating the estimated SLAM trajectory against the raw odometry data. Three different methods are used for data association; Maximum Likelihood + Individual Compatibility, Joint Compatibility Branch and Bound, and Maximum Likelihood + Tracking Information. The accuracy of each method is evaluated by first computing the distance between the observations and the associated landmarks and then verifying if the distance is within the expected radius of a cone. The computational time for each method was also evaluated.

Le Large *et al.* evaluates the accuracy and efficiency between EKF SLAM and Graph-SLAM [19]. Different metrics are used to evaluate the accuracy. The first being the number of blue, yellow, orange, and total cones mapped by each algorithm. The match ratio was also calculated, which is the number of cones that match between the SLAM-generated map and the ground truth map, divided by the number of cones on the SLAM-generated map. The next metric evaluated was the percentage of mapped cones with a positioning error above 30cm. Finally, the mean squared error (MSE) of the positions of all mapped cones was calculated. Efficiency was also evaluated on various metrics. The first being the time taken to compute the callback for odometry data and observations, and the second being the central processing unit (CPU) usage for one complete lap of the Autocross event. Finally, for the GraphSLAM algorithm only, a sliding window was implemented with varying window lengths (5s, 10s, 20s, and 30s), which limits the number of constraints used for optimisation. The MSE, run time, and CPU usage were evaluated for each window length and when no window was used.

The papers covered in this section all show quantitative comparisons between particle filter-based SLAM algorithms and graph-based SLAM algorithms for a variety of evaluation metrics. This is just a select few of many papers [41, 5] that explore SLAM comparisons in the context of Formula Student, all making use of similar evaluation metrics.

Within Edinburgh University Formula Student we have also evaluated FastSLAM 1.0 and FastSLAM 2.0 on various metrics. The metrics used were false positive and false negative rates for loop closure, the RMSE between the map produced by the SLAM algorithm and the map generated from the Real-Time Kinematic (RTK) receiver, the difference in the number of landmarks between the map produced by the SLAM algorithm and the map generated from the RTK receiver, and worst-case execution time for one iteration. Some of these metrics will be further explored in this thesis through the comparison with GraphSLAM.

2.9 Summary

In this chapter, we explored the development of the localisation and mapping problem and looked at how SLAM solves this problem. We went into detail on a particle filter-based SLAM algorithm, FastSLAM 2.0, and a graph-based SLAM algorithm, GraphSLAM. Finally, we carried out a literature review on different evaluation metrics used in the general SLAM field and then the specific use case of Formula Student AI.

I chose FastSLAM 2.0 as the baseline for comparison with GraphSLAM, as mentioned in Section 1.3, as it is our best-performing SLAM implementation at EUFS-AI. The choice to implement GraphSLAM, over other more modern graph-based SLAM algorithms such as ORB-SLAM3, was due to the simplicity of the Formula Student application, limitations of our hardware and current software architecture. As seen in 1.1, the environment in which the vehicle is traversing is relatively controlled and predictable. We only have four types of cones/landmarks, with each track layout having to comply with very specific rules. This means we would not be able to utilise the full potential of implementing a more modern advanced algorithm such as ORB-SLAM3, which can handle dynamically changing environments and multi-map merging [6]. The simplicity of GraphSLAM makes it more applicable to the Formula Student application. Furthermore, a combination of the ADS-DV's hardware constraints and the computational complexity of our current software stack, explained more in Section 3.1, limits the computational resources that are available for our SLAM implementation. Due to this limitation, choosing a simpler, less computationally expensive algorithm, such as GraphSLAM, reduces the computational latency and helps maintain the real-time performance required in a racing environment.

Chapter 3

Implementation

In this chapter, the high-level architecture of the EUFS-AI autonomous software system is explained, detailing the requirements for the implementation. Followed by a detailed explanation of the SLAM implementation.

3.1 EUFS-AI Software Architecture

This section outlines the high-level architecture of the EUFS-AI autonomous software system to determine the implementation requirements for the SLAM algorithms to fit into the wider system. The structure can be seen in Figure 3.1.

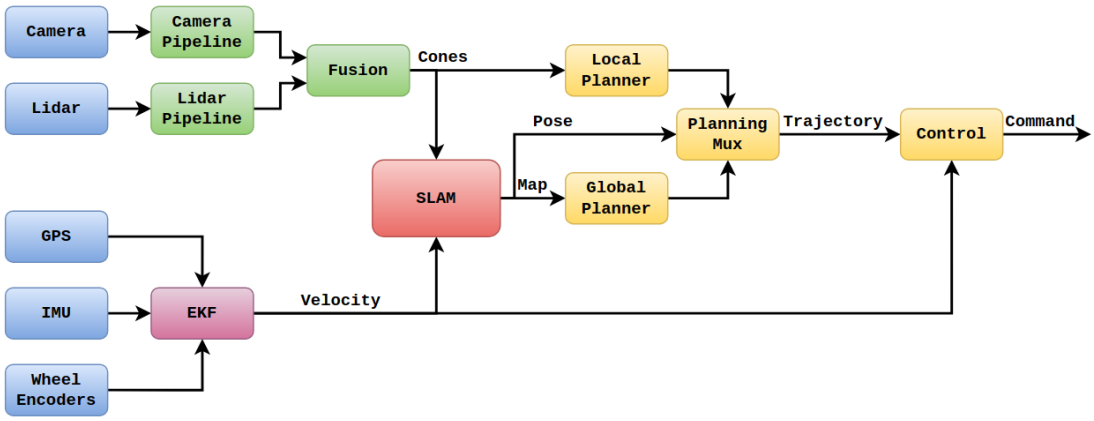


Figure 3.1: EUFS-AI software architecture.

The autonomous system has two modes; learning mode and racing mode. For the Trackdrive event, the system starts in learning mode. Learning mode navigates the vehicle around the track while the SLAM sub-system, seen in red in Figure 3.1, creates the map of the track. While in learning mode the perception sub-system, in green, outputs cone positions to the planning and control sub-system, in yellow, which calculates a trajectory and generates a set of commands using the controller to follow this trajectory. The controller also considers the velocity estimates from the EKF output, in pink,

when generating the output commands. The trajectory is restricted by the distance the perception pipeline can identify cones, limiting the speed of the vehicle, as discussed in Section 1.2. Note the global planner is not used at this time as there is no SLAM output.

After the first lap is complete, SLAM outputs a map of the track to the global planner, which generates an optimal trajectory. After this optimal trajectory has been generated the system then switches to racing mode. The planning multiplexer takes the optimal trajectory from the global planner, along with the current pose from the SLAM sub-system and outputs a new trajectory for the controller to generate a set of commands to follow this trajectory. As the track is now known, the vehicle is not limited by the perception sub-system and can travel at much faster speeds.

The focus of this thesis is on the SLAM sub-system, seen in red in Figure 3.1, which takes in velocity from the EKF and cones from the perception sub-system. Figure 3.2 shows the current perception cone extraction algorithm applied on top of a camera image. Due to the preprocessing of the images by the perception sub-system to extract the cone locations, implementing a visual-based SLAM algorithm, such as ORB-SLAM3, where images are taken as the input and landmarks are extracted directly from the images, would require drastic changes to the entire sub-system. The perception sub-system would no longer be required to extract the cones, as the images would be directly fed into the SLAM sub-system. Furthermore, the map output from the SLAM sub-system would contain more than just cones as landmarks, which would require additional changes to the planning and control sub-system to account for the additional landmarks. These changes to the perception and planning and control sub-systems are outside the scope of this thesis, which focuses on the SLAM sub-system only. Future work could be carried out to explore this, as better results could be achieved by extracting landmarks directly from images.

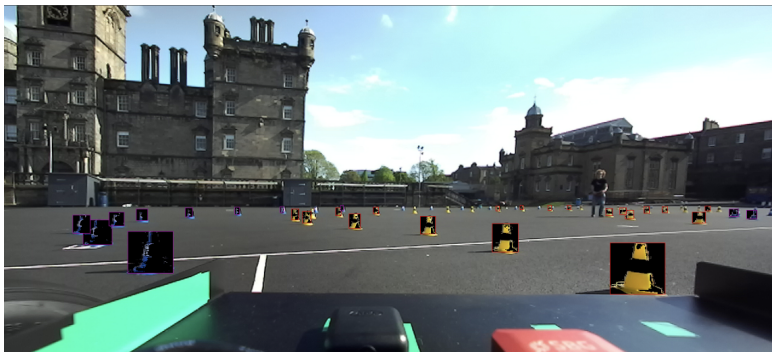


Figure 3.2: EUFS-AI camera image with cone extraction algorithm.

3.1.1 SLAM Implementation Requirements for EUFS-AI

Based on the architecture of the autonomous system mentioned in Section 3.1, the SLAM implementation in this thesis must be Robot Operating System (ROS) 2 compliant [35] to allow integration with the other sub-systems. It must take in velocity measurement, from the EKF sub-system, and an array of x and y-coordinates with the corresponding 2 by 2 covariance matrix, from the perception sub-system. Note

the x and y-coordinates are in the ROS `base_footprint` coordinate frame, where the origin is under the centre of mass of the vehicle. The implementation must publish each complete lap of the track, the vehicle pose continuously after the first lap, and the map of the track after the first lap (to allow racing mode). Based on these requirements we can now go into the implementation details.

3.2 Programming Language

Due to the requirement that the system must be ROS 2 compliant, the system can only be implemented in C/C++ or Python, as these are the only supported programming languages in ROS 2. C/C++ are faster at runtime compared to Python, therefore the GraphSLAM implementation is in C++. Furthermore, the current EUFS-AI SLAM implementations are in C++, allowing for common methods and classes to be shared between algorithms.

3.3 Odometry Buffer

As outlined in Section 3.1, the inputs to the SLAM sub-system are the velocity estimates from the EKF and the cone positions from the perception sub-system. The rates of these inputs are different; velocity estimates are received at a rate of 30Hz and cone positions around 10Hz. To handle the variations in input rates a buffer was implemented to store the most frequent inputs, in this case, the velocity estimates. As the motion model requires only the previous pose and the current pose, the velocity estimates are integrated into a sequence of poses and stored as poses instead. Each pose in the buffer is assigned a timestamp which comes from the timestamp of the velocity measurement that caused the vehicle to move from the previous pose to the current pose. Upon receiving cone estimates, the motion model takes as input the oldest pose in the buffer and the most recent pose in the buffer, before or at the timestamp on the cone estimates. The buffer is then cleared, except for the last pose which is kept as the current pose of the vehicle for use when the next set of cone estimates are received.

3.4 Motion Model

The motion model is used to predict the future pose of the vehicle before considering the cone estimates. The chosen motion model for this thesis was the odometry motion model outlined by Thrun *et al.* in Probabilistic Robotics [38]. This odometry motion model is simple with few parameters, making it easy to implement and a common choice for SLAM implementations [7, 25]. The motion model algorithm, along with the mathematics behind this model, can be found in Appendix A.

3.5 Measurement Model

The measurement model is used to interpret the cone positions based on the vehicle pose and the map. The measurement model described in [10] was chosen as it transforms

the cone positions from the `base_footprint` frame to the map frame using the vehicle pose in the map, which suits our system as we receive the cone positions in this frame. The underlying theory behind this measurement model can be found in Appendix B.

3.6 Data Association

Data association is used in SLAM to identify if a newly received cone already exists in the SLAM map or if it should be added to the map. One of the simplest data association techniques is gating, which is the technique used in this thesis. There are also more advanced techniques such as joint compatibility branch and bound (JCBB) [27] and multi-hypothesis association [28].

The gating technique implemented uses nearest neighbour with the Mahalanobis distance. Mahalanobis distance was chosen over the standard Euclidean distance as it considers the cone measurement uncertainty, giving more accurate data association. The cone estimates received from the perception sub-system have colours associated with each cone. When the perception sub-system cannot determine the colour of the cone with enough accuracy it marks the colour as unknown. For each cone received from perception, the Mahalanobis distance is calculated between this cone and each cone of the same colour or unknown colour in the map. The pair of cones with the smallest Mahalanobis distance and within the threshold distance are merged with the new position being the mean of the two positions, the colour of the pre-existing cone is adopted, if not unknown. If the distance threshold is not met and no cones are merged then the new cone is added to the map. Incorporating the cone colours into the data association prevents cones that we know are on different sides of the track from being merged. This addition makes the assumption that the cones assigned a colour by the perception sub-system are done so accurately.

3.7 Loop Closure

Loop closing identifies when the vehicle crosses the finish line and thus has completed a lap. There is no standard loop closure detection approach for SLAM as it is application-specific. The loop closure approach implemented in this thesis works as follows. Once the SLAM system receives large orange cones from the perception sub-system, the largest distance between these cones is calculated. If this distance exceeds 2.5 metres then a vector between these cones is created. The threshold distance of 2.5 metres was chosen based on the Trackdrive layout, where the start/finish line width must be at least 3 metres, as specified in Section 1.1.1. This threshold distance prevents a vector from being created between two orange cones on the same side of the track. Once we have the vector across the track a 3m by 3m square is created perpendicular to the vector. As we do not know the direction of the vector, i.e. it could be from the big orange cone on the left to the big orange cone on the right or vice versa, a square is created on each side of the vector. When the vehicle enters this square, known by calculating the distance to the orange cones, a countdown is started to estimate when the vehicle will cross the start/finish line. This estimate is calculated based on the distance to the line

and the current velocity of the vehicle, assuming the vehicle will continue at constant velocity. To reduce the impact of this assumption, the countdown value is repeatedly re-computed until the vehicle is no longer in the square or the large orange cones are no longer reported by the perception sub-system. This countdown is required due to the sensor's limited field of view, first mentioned in Section 1.1.2, meaning the large orange cones go out of sight of the sensors before the vehicle crosses the line, as seen in the right-most diagram of Figure 3.3. Figure 3.3 helps to visualise this loop closure approach.

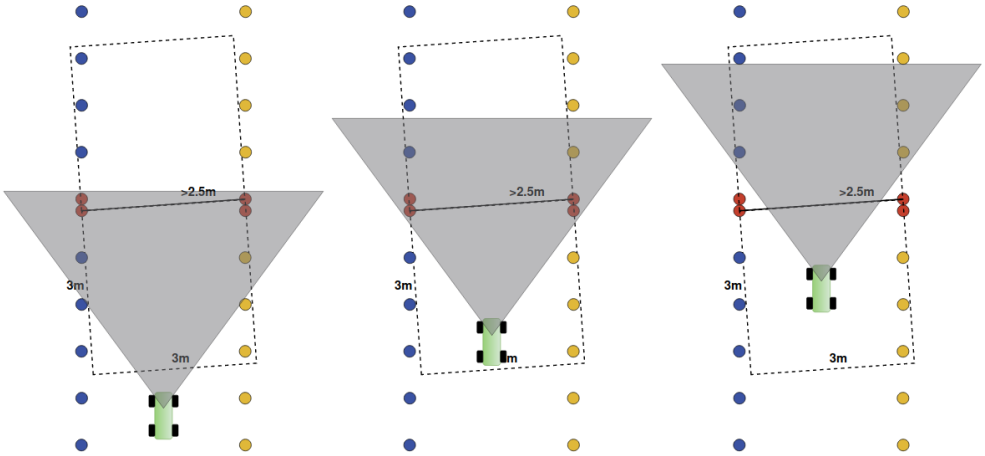


Figure 3.3: Loop closure detection.

3.8 Unknown Cone Colouring

After running my GraphSLAM algorithm and looking at the results I noticed many of the cones in the SLAM map were left uncoloured, despite updating colours during data association. I decided to further advance the colour correction in the data association process to also account for the side of the vehicle the cones are positioned on. We know the colour of the cones to the left of the vehicle are blue and to the right of the vehicle are yellow, as highlighted in Section 1.1, so this information can be utilised to colour cones with unknown colour. The orientation of the vehicle is taken and a line is projected in the direction of travel. If the cone of interest is within a distance threshold from the vehicle and is to the left of this line, it is assigned the colour blue, and to the right, it is assigned the colour yellow. The distance threshold is to restrict the distance of colouring unknown cones, as it is common for the perception sub-system to mark far away cones as unknown before the vehicle is close enough to be confident in their colour. It also limits the negative effect of colouring cones on the wrong side of the track at sharp corners, where it may appear that cones on the left side of the track are on the right side of the vehicle, and vice versa, due to the vehicle not yet changing direction for the corner. This addition may help improve data association as more cones around the vehicle will be assigned a colour, allowing them to be merged if they are within the Mahalanobis distance of another cone already in the map. Furthermore, outputting a more accurate map from the SLAM sub-system will allow the planning and control sub-system to plan a better trajectory around the track.

Chapter 4

Methodology

In this chapter, the data collection procedure is outlined, along with the evaluation metrics, and evaluation procedure used to analyse the collected data and answer the aims outlined in Section 1.4. The results of the evaluation procedure are presented in Chapter 5.

4.1 Track Layouts

This section details the three track layouts used in this thesis to evaluate the two SLAM algorithms of choice. These tracks follow the Trackdrive event rules, outlined in Section 1.1.1, but have very different layouts to test various aspects of the SLAM algorithms.

For each track, an RTK Global Navigation Satellite System (GNSS) receiver was used to map the cones with centimetre accuracy. The RKT GNSS was placed on the centre of each cone and its position was recorded. To obtain more accurate position data, readings were recorded for 2 seconds, collecting 20+ GPS messages, and an average was calculated. As well as the position of each cone, the colour of each cone and the rough starting position of the vehicle (around 3 metres behind the start line) were also manually recorded. The complete RTK maps of the tracks can be seen in Figures 4.1, 4.2, and 4.3.

4.1.1 Rectangle Track

The rectangle track, as seen in Figure 4.1, has 90°turns at each corner of the track and long straights between. Based on the author's experience, this is unlikely to be a track layout at the Formula Student AI competition; however, it tests the SLAM algorithm's capability to handle long straights and sharp turns. Being able to do this accurately would allow for increased speeds along the straights, beyond the capability of solely using perception data, and faster navigation through tight corners, such as hairpins or chicanes. This being the longest of the three tracks maximises the distance before the vehicle revisits the same point on the track, testing the performance of loop closure and data association.

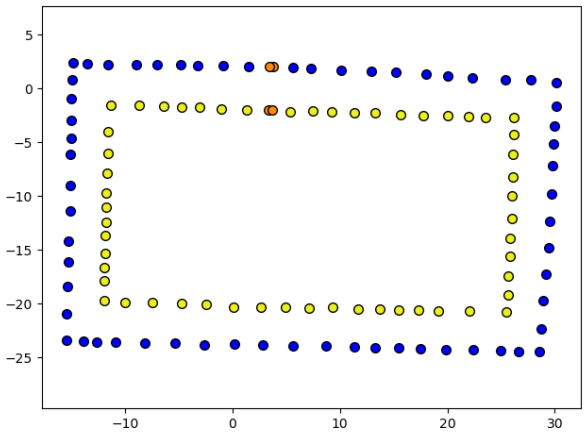


Figure 4.1: RTK map of the rectangle track.

4.1.2 Peanut Track

The peanut track, as seen in Figure 4.2, has a relatively simple shape without any long straights or sharp turns. The complexity of this track comes from the near intersection in the middle of the track where the perception sub-system will output cones from the other side of the track as well as the side of the track the vehicle is currently on. This poses additional challenges for the algorithm's data association. Unlike the rectangle track, the peanut track is non-linear as it is made up of continuous left and right turns. This tests the algorithm's ability to deal with larger uncertainty in the vehicle orientation.

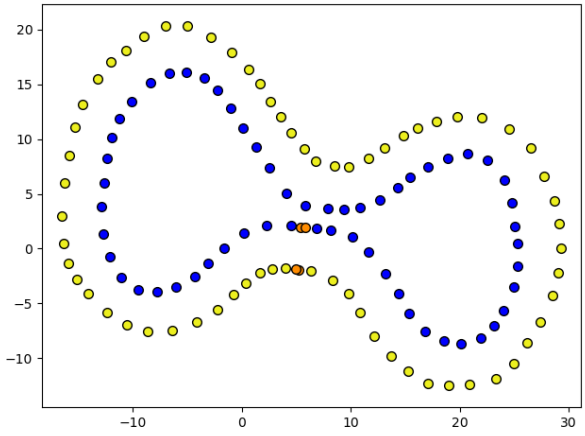


Figure 4.2: RTK map of the peanut track.

4.1.3 Hairpin Track

Finally, the hairpin track, as seen in Figure 4.3, is the most complex of the three tracks and, based on the author's experience, is the most representative of what a Trackdrive track layout might look like at the Formula Student AI competition. Similar to the peanut track, this track is non-linear with few straights. The cone density is also a lot higher compared to the other two tracks, which tests the algorithm's ability to deal with

increased computational complexity and the ability of the algorithm's data association to handle cones closer together.

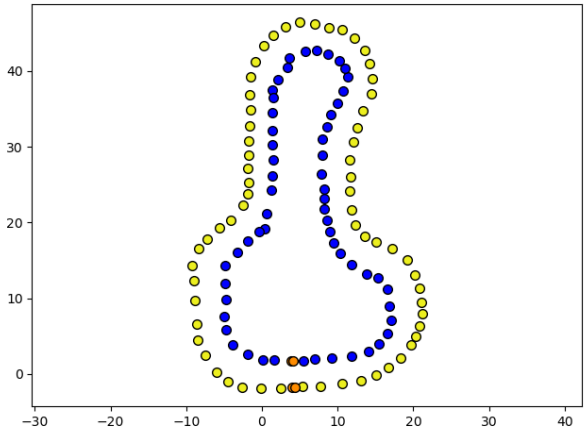


Figure 4.3: RTK map of the hairpin track.

4.2 Autonomous Data Collection

The ADS-DV, outlined in Section 1.1.2, equipped with a sensor suite and the EUFS-AI software stack, was the chosen platform to obtain the data required to evaluate the SLAM algorithms of interest in this thesis. Each track was mapped using the RTK GNSS, as described in Section 4.1, and then the ADS-DV was driven around each track autonomously in learning mode (defined in Section 3.1). As the vehicle was traversing the track, the EKF output and the perception output were recorded using `rosbag2` to allow the SLAM algorithms to be evaluated offline.

4.3 Evaluation Metrics

This section outlines the evaluation metrics used to analyse the two SLAM algorithms of interest in this thesis. The initial metrics used to evaluate the performance of the algorithms were; map RMSE, cone counts, and worst-case execution time. These metrics were chosen, over other metrics (see Sections 2.7 and 2.8.1), as they most effectively answer the aims defined in Section 1.4. Additionally, the chi-squared metric was used to evaluate the accuracy of different GraphSLAM optimisation algorithms.

4.3.1 Map Root Mean Squared Error (RMSE)

The first metric is the map root mean square error. This metric is a measure of the error in the distances between the cones in the map generated by the SLAM algorithms after the first lap of the track and the map created using the RTK receiver, mentioned in Section 4.1. The RMSE is a metric used across most SLAM evaluation literature to compute the accuracy of the SLAM map, as outlined in Sections 2.7 and 2.8.1. The SLAM algorithm that consistently produces a lower RMSE can be determined as the

algorithm that more accurately maps the track, which is the second aim of this thesis. Furthermore, as this is a common metric for measuring accuracy, performance can be easily compared with other literature.

4.3.2 Cone Counts

The second metric is the difference between the number of cones in the map generated by the SLAM algorithms after the first lap of the track and the map created using the RTK receiver. Using this metric alongside the RMSE gives a truer representation of the accuracy, as the SLAM map could have additional cones nearer to the cones in the RTK map, skewing the RMSE and accuracy of the algorithm. The SLAM algorithm that consistently produces a lower cone count difference, alongside a lower RMSE, can be determined as the algorithm that more accurately maps the track, which is the second aim of this thesis. These counts can also be refined by colour to assess the ability of the algorithm to classify cones.

4.3.3 Worst-Case Execution Time

The third metric is the worst-case execution time for one iteration in learning mode. One iteration is the time taken for a message from the perception subsystem to be integrated into the state. Lower computation time is desirable to guarantee the SLAM algorithms process the inputs fast enough to keep up with the real-time responsiveness required when racing. Also, lower computation time within the SLAM sub-system allows the limited computational resources of the ADS-DV to be used by other sub-systems. The worst-case execution time was chosen as this shows the maximum delay between consecutive outputs, which is crucial for understanding the sub-system's reliability and safety. The SLAM algorithm with the lower worst-case execution time across different tracks can be determined as the algorithm with the smaller computational latency, which is the third aim of this thesis.

4.3.4 Chi-Squared

The additional metric is the chi-squared value, which is the sum of the squared residuals between the estimated values and measurements during the GraphSLAM optimisation process. A lower chi-squared value indicates the robot pose and cone position estimates are close to the measurements, resulting in a better-fitting graph. The graph optimisation process is an iterative procedure and computing the chi-squared value after each iteration allows for convergence to be evaluated and different optimisation algorithms to be compared. The optimisation algorithm that converges to a lower Chi-value is more desirable as it shows a better fit to the measurements, suggesting a more accurate solution to the real-world environment.

4.4 Evaluation Procedure

This section describes the evaluation procedure that was carried out on the collected data, as described in Section 4.2, to obtain the results in Chapter 5.

4.4.1 Parameter Tuning

Prior to conducting experiments, the parameters for the SLAM algorithms were tuned. As the FastSLAM 2.0 algorithm was used by EUFS-AI at the Formula Student AI competition, the parameters for this algorithm were already tuned. The optimisation techniques used for this tuning were a Bayesian Optimisation technique, implemented by Nogueira *et al.* [29], and the Covariance Matrix Adaptation Evolution Strategy [12], which is a black box optimisation (BBO) technique. BBO is the process of minimising an objective function without knowing the underlying function itself, simply choosing inputs and observing the outputs. The objective function used by both techniques was the RMSE value between the SLAM-generated map and the RTK-created map after alignment using Iterative Closest Point (IPC). As the GraphSLAM algorithm implemented in this thesis has different parameters from the FastSLAM 2.0 algorithm, parameter tuning had to be carried out. Due to limited time constraints and surprisingly accurate results from hand-tuning the parameters, the optimisation techniques used for FastSLAM 2.0 were not used for GraphSLAM. The parameters were hand-tuned across the different tracks until a sufficient map was generated. Future work should be conducted to tune these parameters using optimisation techniques.

4.4.2 Map RMSE

The GPS coordinates collected using the RTK receiver were converted to Universal Transverse Mercator coordinates to transform them onto a 2D flat plane. The coordinates were then shifted so the estimated vehicle start position became the origin of the map frame. The cone coordinates were then rotated so that the positive x-axis lies in the vehicle's direction of travel, determined by a vector from the vehicle start position to the centre of the start line. IPC is then used to align the cone in the SLAM map with the cones in the RTK map to create cone pairs. The RMSE of these pairs is then computed.

4.4.3 Worst-Case Execution Time

The worst-case execution time was calculated by recording the execution time of the function that integrates the perception input. This time was computed for every perception input received during the first lap of the track and the largest time reported. The computation time for integrating the velocity inputs are ignored as they are insignificant compared to the computation time of integrating the perception input.

This metric was collected offline on a laptop running Ubuntu 22.04 LTS with an Intel Core i7-8565U CPU with 16GB RAM. See Appendix C for the extended specifications.

4.4.4 Chi-Squared

Seeking further improvement to the accuracy of the GraphSLAM algorithm an experiment to compare different optimisation algorithms was conducted. The chi-squared value for each iteration was computed during the optimisation procedure by the g^2o library [18], where the optimisation algorithm and the number of iterations were specified as parameters.

Chapter 5

Results & Discussion

This chapter explores the results of the FastSLAM 2.0 and GraphSLAM algorithms across various metrics, followed by a discussion regarding these results.

5.1 Results

This section presents the results from the experiments described in Chapter 4. Initially, FastSLAM 2.0 is compared to GraphSLAM across a variety of metrics, followed by an additional experiment to seek further improvement to the GraphSLAM algorithm. Note for all experiments involving a comparison between FastSLAM 2.0 and GraphSLAM, FastSLAM 2.0 was run with 100 particles and 10 treads, and GraphSLAM was run using the Levenberg-Marquardt (LM) optimisation algorithm with a maximum of 50 iterations for optimisation. These experiments were repeated 20 times and averages were computed, except for the worst-case execution time where the largest result was reported. The asterisk (*) indicates the enabling of unknown cone colouring for GraphSLAM.

5.1.1 Map RMSE

Table 5.1 presents the RMSE of the cone positions between the map generated from the SLAM algorithms and the map created using the RTK receiver. Note that the number of cones is not considered in these experiments, allowing a perfect score of 0 to be obtained if there were no cones in the SLAM map. The cone counts are considered in section 5.1.2.

Algorithm	Rectangle	Peanut	Hairpin
FastSLAM 2.0	0.86	1.54	1.42
GraphSLAM	0.38	0.93	0.73
GraphSLAM*	0.38	0.93	0.71

Table 5.1: Comparison of RMSE values for GraphSLAM and FastSLAM 2.0 across different tracks.

5.1.2 Cone Counts

Table 5.2, 5.3 and 5.4 show the average difference in the number of cones between the map generated from the SLAM algorithms and the map created using the RTK receiver. Positive values indicate more cones in the map generated from the SLAM algorithms and negative values indicate more cones in the map created using the RTK receiver.

Algorithm	Blue	Yellow	Orange	Unknown	Total
FastSLAM 2.0	+1.40	-4.20	-1.00	+3.10	-0.70
GraphSLAM	+1.00	-6.00	-2.00	+7.00	0.00
GraphSLAM*	+0.95	-1.00	-2.00	+2.00	-0.05

Table 5.2: Cone count differences for the rectangle track.

Algorithm	Blue	Yellow	Orange	Unknown	Total
FastSLAM 2.0	+5.20	+20.10	+2.36	+0.32	+27.98
GraphSLAM	-16.84	-1.00	-2.00	+18.32	-1.52
GraphSLAM*	-1.60	+0.10	-2.00	+3.00	-0.50

Table 5.3: Cone count differences for the peanut track.

Algorithm	Blue	Yellow	Orange	Unknown	Total
FastSLAM 2.0	-6.30	+7.90	-2.10	+1.00	+0.50
GraphSLAM	-2.05	+17.00	-2.00	+9.00	+21.95
GraphSLAM*	-0.05	+16.9	-2.00	+7.05	+21.9

Table 5.4: Cone count differences for the hairpin track.

5.1.3 Worst-Case Execution Time

Table 5.5 shows the worst-case execution time, in seconds, across the three different tracks.

Algorithm	Rectangle	Peanut	Hairpin
FastSLAM 2.0	0.0279	0.0115	0.0174
GraphSLAM	0.0028	0.0031	0.0022
GraphSLAM*	0.0029	0.0034	0.0022

Table 5.5: Comparison of worst-case execution time, in seconds, for each SLAM algorithm across each track.

5.1.4 Chi-Squared

Figure 5.1 shows the chi-squared error over 10 iterations of the Powell's Dogleg and the Levenberg-Marquardt optimisation algorithms on the rectangle track. Both algorithms were run 5 times and all results were reported. Note the maximum number of iterations was set to 10 and unknown cone colouring was enabled.

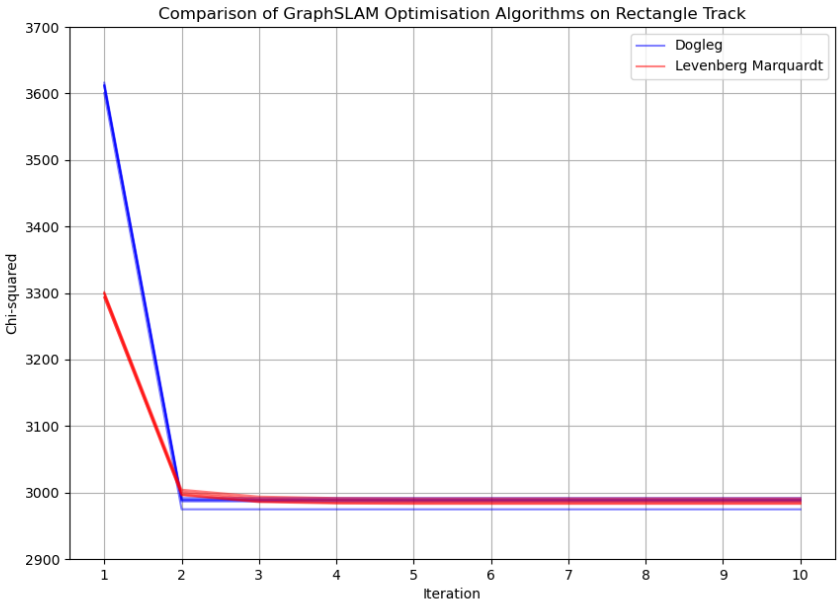


Figure 5.1: Chi-squared error over 10 iterations across 5 runs of the rectangle track for the Powell's Dogleg and the Levenberg-Marquardt optimisation algorithms.

Table 5.6 presents the average RMSE of the cone positions between the map generated from the GraphSLAM algorithm and the map created using the RTK receiver for both the optimisation algorithms. Each experiment was conducted 5 times and an average was computed for all tracks.

Optimisation Algorithm	Rectangle	Peanut	Hairpin
Levenberg-Marquardt	0.383	0.924	0.699
Dogleg	0.383	0.935	0.705

Table 5.6: Average RMSE across different tracks for the Powell's Dogleg and the Levenberg-Marquardt GraphSLAM optimisation algorithms.

5.2 Discussion

In this section, the results from Section 5 and the maps shown in Figures 5.2, 5.3, and 5.4 are analysed based on the thesis aims, outlined in Section 1.4. Including a discussion about the challenges and limitations of these results.

5.2.1 Implementation

The first aim of this thesis was to implement a graph-based SLAM algorithm into the EUFS-AI software stack. This has been successfully achieved, obtaining results for all metrics (see Section 5.1) and producing SLAM maps for each track (see Figures 5.2, 5.3, and 5.4).

5.2.2 Accuracy

The second aim of this thesis was to determine which type of SLAM algorithm, either particle filter-based or graph-based, consistently produced more accurate maps. To determine the accuracy of FastSLAM 2.0 maps and GraphSLAM maps two metrics were used; the RMSE between the SLAM map and the RTK map and the cone counts.

From Table 5.1 we can see that the RMSE across all tracks was smaller using the GraphSLAM algorithm compared to the FastSLAM 2.0 algorithm. The SLAM maps in Figures 5.2, 5.3, and 5.4 agree with this, showing GraphSLAM's SLAM map having more cones closer to the cones in the RTK maps than FastSLAM 2.0's SLAM map. Furthermore, the addition of the unknown cone colouring, during data association, further improved the accuracy of the hairpin track. As previously mentioned, in Section 4.1.3, the cone density of the hairpin track is higher than that of the other tracks. This potentially explains the improvement; an increase in cone density means more cones will be within the distance threshold for merging. Colouring more of these cones before merging prevents cones that previously would have had unknown colour, if unknown cone colouring was disabled, from being incorrectly merged with a cone where their true colours differ, thus resulting in more accurate maps, as seen in Figure 5.4c.

The cone counts show varying results. Table 5.2 shows that on the rectangle track, the difference in the total number of cones is similar for all algorithms, around 0. This may indicate that the RMSE for the rectangle track is a reliable metric to determine accuracy. Table 5.3 shows that on the peanut track, for FastSLAM 2.0, there are on average around 28 more cones in the SLAM map than the RTK map. This could indicate that the RMSE of the FastSLAM 2.0 SLAM map is skewed by the additional cones, making the use of the RMSE on the peanut track, as a measure of accuracy, less desirable. We can see from Figure 5.3a that the additional cones in the FastSLAM 2.0 SLAM map are mostly around the centre of the track, where the perception sub-system will report cones on both sides of the track. This is potentially a weak point in the FastSLAM 2.0 data association algorithm as it struggled to determine which cones to merge on which side of the track. Both the GraphSLAM implementations have similar totals, around 1, for the peanut track. Conversely, on the hairpin track, the FastSLAM 2.0 map has a total of +0.50 and the GraphSLAM maps both have much larger totals, around 22. This suggests that the GraphSLAM algorithms may have skewed RMSE results for this track. Looking at the maps of the hairpin track in Figure 5.4 we can see that both GraphSLAM maps have a surplus of cones around the start/finish line. This could suggest a weakness in the GraphSLAM data association algorithm, particularly when trying the merge big orange cones and the surrounding cones at the start/finish line.

The fluctuations in the cone counts across the peanut and hairpin tracks may indicate

that the threshold distance used in data association is nonoptimal and a better value could be chosen to improve performance across all tracks. Furthermore, a completely different data association algorithm, such as JCBB, may produce better SLAM maps with similar cone counts to the RTK maps.

Taking into consideration the RMSE values and the cone counts, potentially the most reliable set of results to determine the accuracy of the SLAM maps are the RMSE values from the rectangle track in Table 5.1. Based on these results the GraphSLAM algorithm is more accurate than the FastSLAM 2.0 algorithm; thus, meaning a graph-based approach produce more accurate SLAM maps than a particle filter-based approach.

The additional experiment conducted to further improve the accuracy of the Graph-SLAM algorithm by changing the optimisation algorithm showed minimal difference in the results. From Figure 5.1 we can see that on the rectangle track both optimisation algorithms converge to a chi-squared value just below 3000 across nearly all 5 runs of each algorithm. One Dogleg run converges to a slightly smaller chi-squared value, indicating that this run found a different minimum. This suggests that neither algorithm is consistently identifying the global minimum and further parameter tuning or the use of different optimisation algorithms should be considered. Figure 5.1 also shows that the Dogleg algorithm seems to converge faster, indicated by the steep drop down to the minimum chi-squared value after 2 iterations. The LM algorithm has a less steep drop after the first iteration and remains above the minimum value before plateauing out after 4 iterations. This suggests the Dogleg algorithm is slightly more efficient at finding the optimal value; however, this does not mean it is more accurate as they both converge to similar chi-squared values. Lin *et al.*[22] explores the same optimisation algorithms as this thesis, also on a SLAM problem, and reports a figure that follows the same trend as Figure 5.1. The plots for the peanut track and the hairpin track also follow the same trend as the rectangle track, thus were not presented.

If one algorithm was to converge to a smaller chi-squared value then it could be determined as a more accurate algorithm, as the error between the measurements and the estimates is smaller, however, this was not the case for either the Dogleg or the LM algorithms. Table 5.6 reinforces this as the RMSE between the SLAM map and the RTK map are similar across both algorithms on all tracks. There is potentially a slight reduction in accuracy on the peanut track when switching from the LM algorithm to the Dogleg algorithm as the RMSE increases by 0.011. Similarly, for the hairpin track the RMSE increases by 0.006 when switching from the LM algorithm to the Dogleg algorithm. However, these differences in RMSE are not significant enough to claim one optimisation algorithm produces more accurate maps than the other.

5.2.3 Computational Latency

The third aim of this thesis was to identify which type of SLAM algorithm, either particle filter-based or graph-based, has a smaller computational latency. The computational latency must be smaller than 100ms due to the 10Hz perception input. If this is exceeded then we will begin to lose messages and performance will start to deteriorate. To determine the computational latency of FastSLAM 2.0 and GraphSLAM the worst-case execution time metric was used.

From Table 5.5 we can see that the worst-case execution time for all algorithms across all tracks is below the 0.1 seconds (100ms) upper limit. However, the worst-case execution time for FastSLAM 2.0 is an order of magnitude larger than GraphSLAM across all tracks. This can be justified by FastSLAM 2.0's additional computational complexity when sampling the vehicle's pose before adding the cone to the map, which is not required by the GraphSLAM algorithm. The GraphSLAM algorithm with additional unknown cone colouring is very slightly slower than the standard GraphSLAM algorithm on the rectangle and peanut tracks. This is expected due to the additional computation required to determine the colour of any cone marked as having an unknown colour. The extra magnitude of difference in the GraphSLAM results compared to the FastSLAM 2.0 results makes GraphSLAM favourable, allowing for additional scope in the latency without worrying about crossing over the 100ms upper bound and losing perception data.

It must also be noted that these results were obtained on the author's laptop which has an Intel Core i7-8565U CPU [2], as mentioned in Section 4.4.3. This differs from the target CPU, used on the ADS-DV at the Formula Student AI competition, which is an Intel Core i7-6700TE [1]. However, both have the same number of cores and threads, meaning the results are unlikely to greatly differ. Note the memory and operating system are also the same across both platforms, as outlined in Appendix C.

Based on this analysis we can conclude that the algorithm with the smaller computational latency is GraphSLAM, the graph-based SLAM algorithm, and due to the additional magnitude of difference in results comes out more favourable when compared to FastSLAM 2.0.

5.2.4 Challenges & Limitations

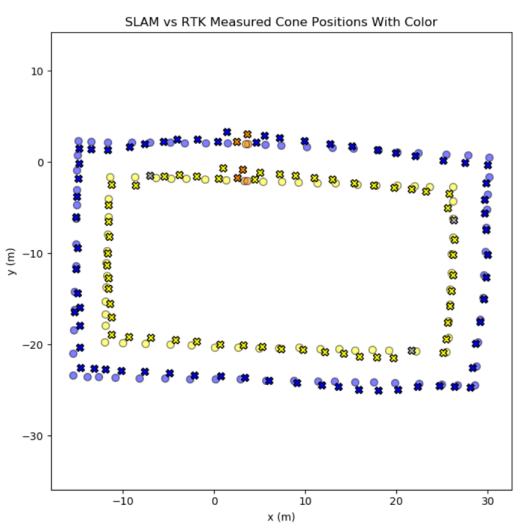
The final aim of this thesis was to identify any challenges for SLAM algorithms in the context of Formula Student AI. This section will cover some challenges and limitations experienced when collecting and analysing these results.

Firstly, due to limited access to the ADS-DV, the RTK maps and the associated EKF and perception data used in these results were collected in 2022, using an older sensor suite than the current suite used by the EUFS-AI team. Our current sensor suite has an improved Lidar which can see cones further away and can measure intensity with greater accuracy. This allows for not just distance to the cones to be measured but also cone colour. This improves perception output and would further improve the SLAM results presented in this thesis.

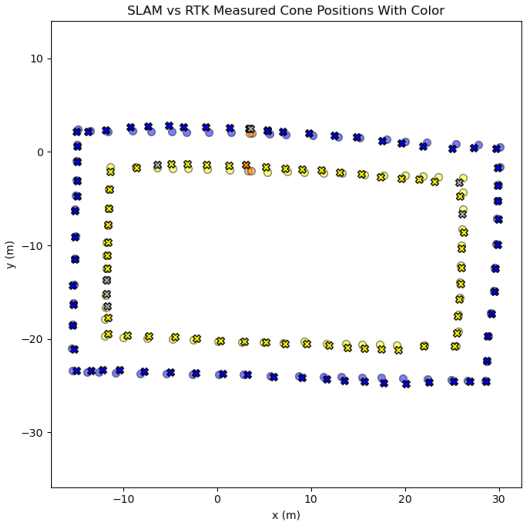
Secondly, based on the author's experience at Formula Student AI competitions, the tracks used in this thesis were shorter than most tracks used at the competition. This was down to the limited space available at our test location. Due to the accumulation of drift, longer tracks are inherently more difficult, meaning there is no guarantee that the SLAM maps created will be as accurate.

Finally, the parameter optimisation process was conducted only on the FastSLAM 2.0 algorithm and not the GraphSLAM algorithm, which was hand-tuned. This potentially means the GraphSLAM algorithm is using sub-optimal parameters, limiting its per-

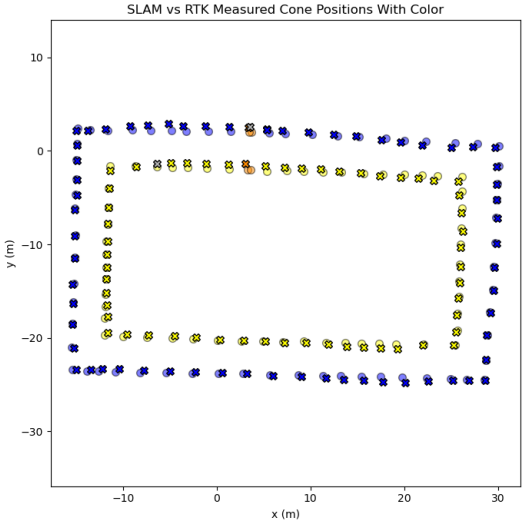
formance and reducing the accuracy of the SLAM maps. Furthermore, the parameter optimisation process takes many hours, depending on the number of parameters and track layout. At the competition the track is unknown prior to the event and the time between runs of the same track is not long enough to carry out the full parameter optimisation process. This poses an additional challenge of obtaining enough data on competition tracks prior to the event to generalise the parameters enough that they perform well on all tracks or come up with a more efficient parameter tuning approach that can be conducted either between runs or during runs at the competition. This should be investigated more.



(a) FastSLAM 2.0

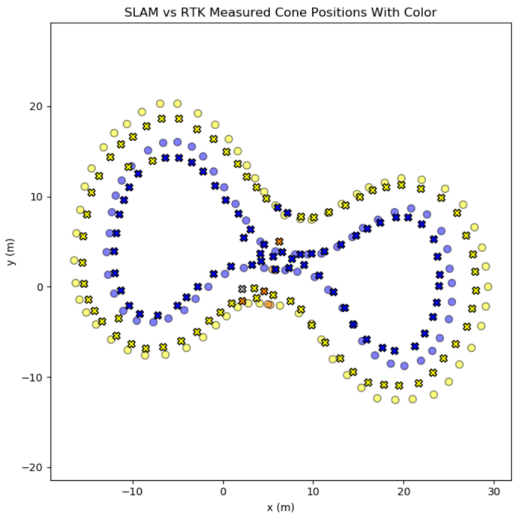


(b) GraphSLAM

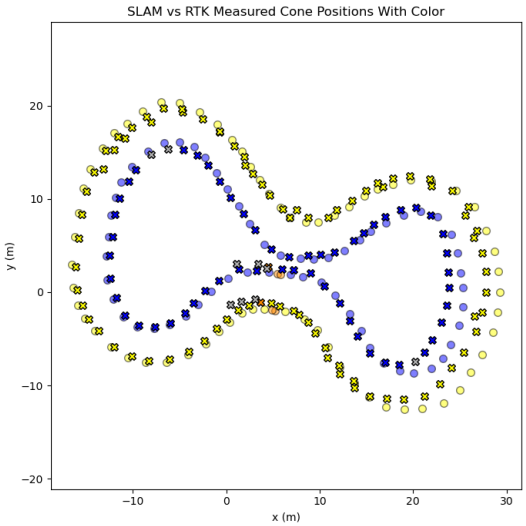


(c) GraphSLAM with unknown cone colouring enabled

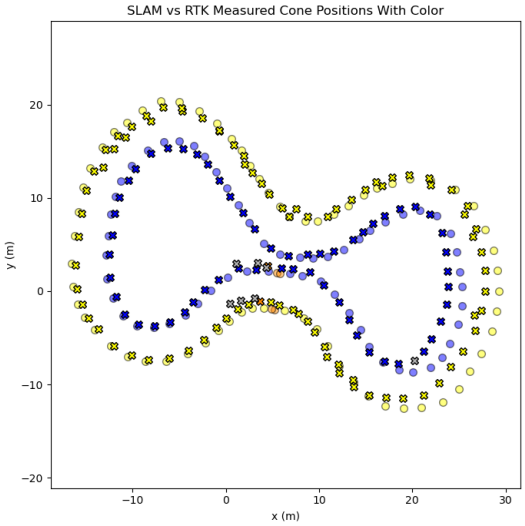
Figure 5.2: Rectangle track: The lowest RMSE SLAM maps (crosses) with RTK maps (faded circles) for each algorithm. The marker colours match the true cone colours and the grey markers indicate unknown cone colour.



(a) FastSLAM 2.0

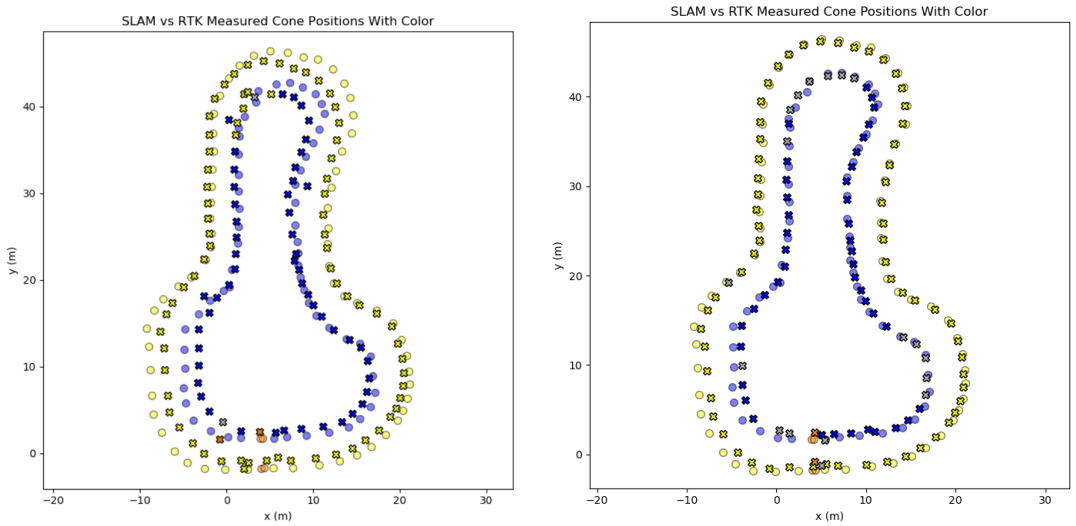


(b) GraphSLAM



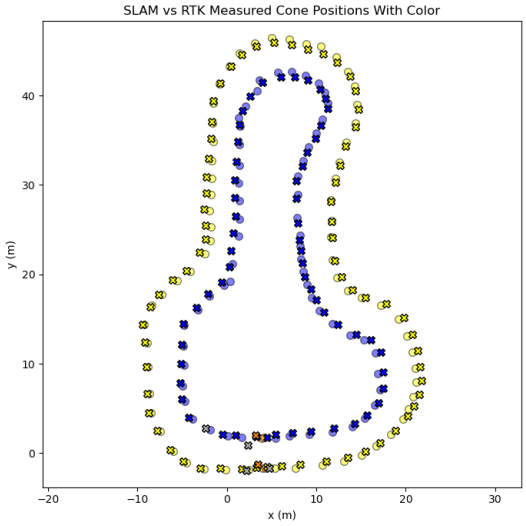
(c) GraphSLAM with unknown cone colouring enabled

Figure 5.3: Peanut track: The lowest RMSE SLAM maps (crosses) with RTK maps (faded circles) for each algorithm. The marker colours match the true cone colours and the grey markers indicate unknown cone colour.



(a) FastSLAM 2.0

(b) GraphSLAM



(c) GraphSLAM with unknown cone colouring enabled

Figure 5.4: Hairpin track: The lowest RMSE SLAM maps (crosses) with RTK maps (faded circles) for each algorithm. The marker colours match the true cone colours and the grey markers indicate unknown cone colour.

Chapter 6

Conclusion

In this chapter, the thesis is concluded with a summary of the achieved work, followed by an outline of potential future work.

6.1 Summary

This thesis investigated the use of particle filter-based and graph-based SLAM algorithms in the context of the Formula Student AI Trackdrive event. This event consists of navigating 10 laps of an unknown track in the fastest time possible. It was highlighted that the speed of the vehicle was limited by the perception range and to overcome this a map of the track, along with the position of the vehicle in this map, was required. SLAM was presented as the solution to this problem; creating a map of the track on the first lap and maintaining the vehicle pose within this map, allowing for optimal path planning and increased maximum speed thereafter.

The background was presented explaining the theory behind SLAM, introducing particle filter-based SLAM as a method for solving the online SLAM problem and graph-based SLAM as a method for solving the full SLAM problem. FastSLAM 2.0 and GraphSLAM were presented as the chosen SLAM algorithms for comparison in this thesis. FastSLAM 2.0 was previously implemented in the EUFS-AI software stack and GraphSLAM was implemented as part of this thesis. The background was concluded by exploring SLAM literature to identify potential evaluation metrics. The architecture of the EUFS-AI software stack was presented and the SLAM implementation details were discussed.

The data collection approach and evaluation approach were then presented, including the three track layouts and the chosen evaluation metrics. The maps of the three tracks were collected by the author and the EUFS-AI team using an RTK receiver. The EFK and perception output were also recorded as the autonomous vehicles navigated the track. The procedure used to obtain the chosen metrics of map RMSE, cone counts and worst-case execution time, alongside the additional chi-squared metric, was outlined.

The results showed that the GraphSLAM algorithm was able to create more accurate maps of the tracks than the FastSLAM 2.0 algorithm, along with a vastly smaller

computational latency, making it the superior SLAM algorithm. This agrees with the literature [23], where Lopes also found GraphSLAM outperformed FastSLAM 2.0 in an autonomous racing context. However, the pose accuracy was not considered in this thesis and should be evaluated before disregarding either of the algorithms and running on the EUFS-AI software stack at the Formula Student AI competition. Furthermore, the additional unknown cone colouring showed slight improvements in accuracy on the hairpin track but largely performed similarly to the standard GraphSLAM algorithm. Finally, the different GraphSLAM optimisation algorithms explored all resulted in similar performance across all tracks, and did not increase the accuracy of the GraphSLAM algorithm.

6.2 Future work

This section outlines potential future work that could be carried out to further improve the mapping of the Formula Student AI tracks and the localisation of the vehicle within these maps.

6.2.1 Pose Evaluation

The accuracy of the pose estimates was not tested in this thesis but is fundamental to the evaluation of SLAM algorithms in the context of Formula Student AI. Similar to the map evaluation procedure outlined in Section 4.4.2, the RTK GNSS receiver could be used to track the vehicle's ”true” position on the track and the RMSE error between this and the pose estimates generated from the SLAM algorithm could be computed. This would give a value for the accuracy of the pose estimates, providing the missing metric to conclude with strong evidence which of the SLAM algorithms is superior for this application.

6.2.2 Data Association

As described in Section 3.6, the current data association technique is gated Mahalanobis distance. This is a simple and computationally efficient technique that can work well in environments where processing data quickly is required. However, more advanced techniques such as JCBB [27] may work better in the context of Formula Student AI. Unlike gated Mahalanobis distance, JCBB considers multiple observations at once, comparing groups of observations to groups of landmarks. This would improve the data association of cones that are clustered together in areas of the track such as around the start/finish line and in tight corners.

6.2.3 Parameter Tuning

As mentioned in Section 5.2.4, the current parameter tuning process that involves running the Bayesian Optimisation technique and the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is not sufficient for use at the competition. Exploring a more efficient method for parameter optimisation or limiting the number of parameters that need to be tuned at the competition by identifying the parameters that are not

dependent on the track layout and tuning these prior to the competition could help overcome this challenge. Furthermore, CMA-ES can be modified to work in real-time [13], tuning the parameters as the vehicle navigates the track, which may also be another viable solution.

6.2.4 Visual-based slam

Due to the current EUFS-AI software stack architecture, as defined in Section 3.1, the input to both SLAM algorithms explored in this thesis comes from the perception sub-system where we received cone locations. This makes these SLAM algorithms feature-based. In more recent years the development of SLAM algorithms has taken a shift towards visual-based SLAM, such as ORB-SLAM3 [6] and OpenVSLAM [33], where the raw sensor measurements are directly used as inputs to the SLAM algorithm. Visual-based SLAM algorithms could be implemented and evaluated in the context of Formula Student AI. The computational complexity of these algorithms would need to be considered due to the limited computational resources available on the ADS-DV; however, if raw camera images are too computationally expensive to process then there is the potential to use Lidar point clouds to extract landmarks instead.

6.2.5 GraphSLAM Optimisation procedure

This thesis explored different optimisation algorithms that the g^2o library [18] had to offer. However, the g^2o library also allows the customisation of the optimisation algorithms through parameters such as maximum number of iterations, convergence threshold, robust kernel parameters, edge weighting and many more. This opens up another opportunity to carry out hyperparameter tuning to find the optimal parameter configuration for Formula Student AI.

Furthermore, the GraphSLAM algorithm implemented in this thesis optimises the graph after a full lap of the track is complete. Optimising the graph at different points throughout the first lap, instead of solely at the end of the lap, may allow for early correction of any errors in the vehicle pose or cone position estimates. This could improve the overall graph by preventing the accumulation of errors.

Finally, unlike FastSLAM 2.0, where there is no computational overhead after the first lap is complete, GraphSLAM has to optimise the graph after the completion of the first lap. The EUFS-AI software system cannot enter racing mode until the SLAM sub-system publishes the map, this means when the GraphSLAM algorithm is used the system will be in learning mode for longer than when the FastSLAM 2.0 algorithm is used. The length of this delay is determined by the graph optimisation time, which depends on the complexity of the graph and the values chosen for the parameters mentioned above. By tuning the parameters to reduce the optimisation time, for example, by reducing the number of optimisation iterations, the SLAM map could be published earlier but may be less accurate as it has undergone fewer optimisation iterations. An investigation could be carried out to compare the trade-off between the accuracy of the SLAM map and the optimisation time, to determine the effect of this delay in regard to the overall completion time of the 10 lap Trackdrive event.

Bibliography

[1] Intel core i7-6700te processor (8m cache, up to 3.40 ghz). https://ark.intel.com/content/www/us/en/ark/products/88201/intel-core-i7-6700te-processor-8m-cache-up-to-3-40-ghz.html, 2023. Accessed: 2024-04-01.

[2] Intel core i7-8565u processor (8m cache, up to 4.60 ghz). https://ark.intel.com/content/www/us/en/ark/products/149091/intel-core-i7-8565u-processor-8m-cache-up-to-4-60-ghz.html, 2023. Accessed: 2024-04-01.

[3] Formula student ai 2024 rules, 2024. URL https://www.imeche.org/events/formula-student/team-information/rules. Accessed: 2023-11-07.

[4] *FSG Competition Handbook 2024*. Formula Student Germany, Germany, 2024.

[5] Andres Alvarez, Nico Denner, Zhe Feng, David Fischer, Yang Gao, Lukas Harsch, Sebastian Herz, Nick Le Large, Bach Nguyen, Carlos Rosero, et al. The software stack that won the formula student driverless competition. *arXiv preprint arXiv:2210.10933*, 2022.

[6] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[7] Jose Luis Blanco Claraco and Machine Perception. Development of scientific applications with the mobile robot programming toolkit. *The MRPT reference book. Machine Perception and Intelligent Robotics Laboratory, University of Málaga, Málaga, Spain*, 40, 2008.

[8] Defense Advanced Research Projects Agency (DARPA). Grand challenge for autonomous vehicles. URL https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles. Accessed: 2023-10-27.

[9] Hugh Durrant-Whyte, David Rye, and Eduardo Nebot. Localization of autonomous guided vehicles. In *Robotics Research: The Seventh International Symposium*, pages 613–625. Springer, 1996.

[10] Udo Frese. A discussion of simultaneous localization and mapping. *Autonomous Robots*, 20:25–42, 2006.

[11] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[12] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[13] Ashley Hill, Eric Lucet, and Roland Lenain. Neuroevolution with cma-es for real-time gain tuning of a car-like robot controller. In *ICINCO (1)*, pages 311–319, 2019.

[14] Indy Autonomous Challenge. Indy autonomous challenge. URL `https://www.indyautonomouschallenge.com/`. Accessed: 2023-10-27.

[15] Institution of Mechanical Engineers. Formula student ai. URL `https://www.imeche.org/events/formula-student/team-information/fs-ai`. Accessed: 2023-11-07.

[16] Juraj Kabzan, Miguel I Valls, Victor JF Reijgwart, Hubertus FC Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, et al. Amz driverless: The full autonomous racing system. *Journal of Field Robotics*, 37(7):1267–1294, 2020.

[17] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234. IEEE, 2007.

[18] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011.

[19] Nick Le Large, Frank Bieder, and Martin Lauer. Comparison of different slam approaches for a driverless race car. *tm-Technisches Messen*, 88(4):227–236, 2021.

[20] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IROS*, volume 3, pages 1442–1447, 1991.

[21] Haoran Li, Qichao Zhang, and Dongbin Zhao. Comparison of methods to efficient graph slam under general optimization framework. In *2017 32nd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 321–326. IEEE, 2017.

[22] Tianxiang Lin, Jiayin Xia, Qishun Yu, Kerou Zhang, and Ben Zhou. Analysis for graph-based slam algorithms under g2o framework. 2021.

[23] Luís Afonso Nazaré Correia Lopes. A slam method for the formula student driverless competition. 2021.

[24] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4:333–349, 1997.

[25] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2718–2725. IEEE, 2020.

[26] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, volume 3, pages 1151–1156, 2003.

[27] José Neira and Juan D Tardós. Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on robotics and automation*, 17(6): 890–897, 2001.

[28] Juan Nieto, Jose Guivant, Eduardo Nebot, and Sebastian Thrun. Real time data association for fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 1, pages 412–418. IEEE, 2003.

[29] Fernando Nogueira et al. Bayesian optimization: Open source constrained global optimization tool for python. 2014.

[30] Oxa Technologies. Oxa technologies. URL `https://oxa.tech/`. Accessed: 2024-02-19.

[31] Royal Society for the Prevention of Accidents. Road crashes: An overview. `https://www.rospa.com/rospaweb/docs/advice-services/road-safety/road-crashes-overview.pdf`, 2023. Accessed: 2024-03-10.

[32] IMecheE Formula Student. Fs-ai dynamic events setup and cones specification, 2021. URL `https://www.imeche.org/docs/default-source/1-oscar/formula-student/2021/forms/ai/fs-ai-dynamic-events-setup-and-cones-specification.pdf?sfvrsn=2`. Accessed: 2024-02-22.

[33] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. Openvslam: A versatile visual slam framework. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2292–2295, 2019.

[34] Synopsys Automotive. What is adas? URL `https://www.synopsys.com/automotive/what-is-adas.html`. Accessed: 2024-03-10.

[35] Dirk Thomas, William Woodall, and Esteve Fernandez. Next-generation ros: Building on dds. *ROSCon Chicago*, 2014, 2014.

[36] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[37] Sebastian Thrun, Michael Montemerlo, Daphne Koller, Ben Wegbreit, Juan Nieto, and Eduardo Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 4(3):380–407, 2004.

[38] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. ACM New York, NY, USA, 2005.

[39] UK Government. Reported road casualties in great britain: Provisional estimates year ending june 2023. URL https://www.gov.uk/government/statistics/reported-road-casualties-in-great-britain-provisional-estimates-year-ending-june-2023/reported-road-casualties-in-great-britain-provisional-estimates-year-ending-june-2023. Accessed: 2024-03-10.

[40] U.S. Space Force and National Coordination Office for Space-Based Positioning, Navigation, and Timing. GPS Accuracy. https://www.gps.gov/systems/gps/performance/accuracy/, 2024. Accessed: 2024-03-16.

[41] Miguel I Valls, Hubertus FC Hendrikx, Victor JF Reijgwart, Fabio V Meier, Inkyu Sa, Renaud Dubé, Abel Gawel, Mathias Bürki, and Roland Siegwart. Design of an autonomous racecar: Perception, state estimation and system integration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 2048–2055. IEEE, 2018.

[42] Waymo LLC. Waymo. URL https://waymo.com/. Accessed: 2024-02-19.

[43] Zichao Zhang and Davide Scaramuzza. A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7244–7251. IEEE, 2018.

Appendix A

Motion Model

The following odometry motion model was presented in Probabilistic Robotics by Thrun *et al.* [38]. It samples a new vehicle pose based on the previous vehicle pose x_{t-1} as $x_{t-1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$ and the odometry u_t as $u_t = \begin{bmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{bmatrix}$ between times $t-1$ and t.

The algorithm transforms the change in odometry into a sequence made up of a rotation, translation, and another rotation (lines 2 to 4). Control noise is then added to this sequence (lines 5 to 7). The resulting values are then used to update the vehicle pose for the current time t (lines 8 to 10) and the new vehicle pose is returned (line 11).

Algorithm 4 Pseudocode for the Odometry Motion Model

1: **function** ODOMETRY_MOTION_MODEL(x_{t-1}, u_t)
2: $\quad \delta_{rot1} = \text{atan2}(y_t - y_{t-1}, x_t - x_{t-1}) - \theta_{t-1}$
3: $\quad \delta_{trans} = \sqrt{(y_t - y_{t-1})^2 + (x_t - x_{t-1})^2}$
4: $\quad \delta_{rot2} = \theta_t - \theta_{t-1} - \delta_{rot1}$

5: $\quad \hat{\delta}_{rot1} = \delta_{rot1} - \text{GAUSSIAN}(\alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2)$
6: $\quad \hat{\delta}_{trans} = \delta_{trans} - \text{GAUSSIAN}(\alpha_3 \delta_{trans}^2 + \alpha_4 (\delta_{rot1}^2 + \delta_{rot2}^2))$
7: $\quad \hat{\delta}_{rot2} = \delta_{rot2} - \text{GAUSSIAN}(\alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2)$

8: $\quad x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
9: $\quad y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
10: $\quad \theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

11: $\quad$ **return** $\bar{X}_t$
12: **end function**

Mathematically, the motion model can be defined as the noise-free motion model g plus some Gaussian noise:

$$\hat{x}_t = g(x_{t-1}, u_t) + \mathcal{N}(0, R_t) \tag{A.1}$$

The noise-free motion model g is defined as:

$$g(x_{t-1}, u_t) = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + \delta_{trans}\cos(\theta + \delta_{rot1}) \\ y + \delta_{trans}\sin(\theta + \delta_{rot1}) \\ \theta + \delta_{rot1} + \delta_{rot2} \end{bmatrix} \tag{A.2}$$

where the control parameters $\delta = [\delta_{rot1}\ \delta_{trans}\ \delta_{rot2}]$ are defined as:

$$\delta_{rot1} = \text{atan2}(y_t - y_{t-1}, x_t - x_{t-1}) - \theta_{t-1} \tag{A.3}$$

$$\delta_{trans} = \sqrt{(y_t - y_{t-1})^2 + (x_t - x_{t-1})^2} \tag{A.4}$$

$$\delta_{rot2} = \theta_t - \theta_{t-1} - \delta_{rot1} \tag{A.5}$$

The Jacobian for the motion model with respect to the above control parameters δ is:

$$G_\delta = \frac{\partial g(x_{t-1}, u_t)}{\partial \delta} = \begin{bmatrix} -\delta_{trans}\sin(\theta + \delta_{rot1}) & \cos(\theta + \delta_{rot1}) & 0 \\ \delta_{trans}\cos(\theta + \delta_{rot1}) & \sin(\theta + \delta_{rot1}) & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{A.6}$$

The Jacobian for the motion model with respect to the vehicle pose x_{t-1} is:

$$G_x = \frac{\partial g(x_{t-1}, u_t)}{\partial x_{t-1}} = \begin{bmatrix} 1 & 0 & -\delta_{trans}\sin(\theta + \delta_{rot1}) \\ 0 & 1 & \delta_{trans}\cos(\theta + \delta_{rot1}) \\ 0 & 0 & 1 \end{bmatrix} \tag{A.7}$$

The covariance matrix R_t is:

$$R_t = G_\delta M_t G_\delta^T \tag{A.8}$$

where M_t is defined as the covariance matrix of the noise that is added to the control parameters:

$$M_t = \begin{bmatrix} \alpha_1\delta_{rot1}^2 + \alpha_2\delta_{trans}^2 & 0 & 0 \\ 0 & \alpha_3\delta_{trans}^2 + \alpha_4(\delta_{rot1}^2 + \delta_{rot2}^2) & 0 \\ 0 & 0 & \alpha_1\delta_{rot2}^2 + \alpha_2\delta_{trans}^2 \end{bmatrix} \tag{A.9}$$

where α_1, α_2, α_3 and α_4 are the tunable hyperparameters of the motion model.

Appendix B

Measurement Model

Mathematically, the measurement model can be defined as the noise-free measurement model h plus some white Gaussian noise $\mathcal{N}(0, Q_t)$ with covariance Q_t:

$$\hat{z} = h(\mu_t, m_i) + \mathcal{N}(0, Q_t) \tag{B.1}$$

where $\hat{z} = \begin{bmatrix} \hat{z}_x \\ \hat{z}_y \end{bmatrix}$ is the predicted measurement.

The noise-free measurement model h is defined as:

$$h(\mu_t, m_i) = \begin{bmatrix} \cos\mu_{t,\theta} & \sin\mu_{t,\theta} \\ -\sin\mu_{t,\theta} & \cos\mu_{t,\theta} \end{bmatrix} \begin{bmatrix} m_{i,x} - \mu_{t,x} \\ m_{i,y} - \mu_{t,y} \end{bmatrix} \tag{B.2}$$

where $\mu_t = \begin{bmatrix} \mu_{t,x} \\ \mu_{t,y} \\ \mu_{t,\theta} \end{bmatrix}$ is the vehicle pose at time t and $m_i = \begin{bmatrix} m_{i,x} \\ m_{i,y} \end{bmatrix}$ is the ith landmark in the map m.

The Jacobian for the measurement model with respect to the vehicle pose μ_t is:

$$H_\mu = \frac{\partial h(\mu_t, m_i)}{\partial \mu_t} = \begin{bmatrix} -\cos\mu_{t,\theta} & -\sin\mu_{t,\theta} & -d_x \sin\mu_{t,\theta} + d_y \cos\mu_{t,\theta} \\ -\sin\mu_{t,\theta} & \cos\mu_{t,\theta} & -d_x \cos\mu_{t,\theta} - d_y \sin\mu_{t,\theta} \end{bmatrix} \tag{B.3}$$

where $d_x = m_{i,x} - \mu_{t,x}$ and $d_y = m_{i,y} - \mu_{t,y}$.

The Jacobian for the measurement model with respect to the landmark m_i is:

$$H_m = \frac{\partial h(\mu_t, m_i)}{\partial m_i} = \begin{bmatrix} \cos\mu_{t,\theta} & \sin\mu_{t,\theta} \\ -\sin\mu_{t,\theta} & \cos\mu_{t,\theta} \end{bmatrix} \tag{B.4}$$

The inverse measurement model h^{-1} is defined as:

$$h^{-1}(\mu_t, z_t) = \begin{bmatrix} \mu_{t,x} \\ \mu_{t,y} \end{bmatrix} + \begin{bmatrix} \cos\mu_{t,\theta} & -\sin\mu_{t,\theta} \\ \sin\mu_{t,\theta} & \cos\mu_{t,\theta} \end{bmatrix} \begin{bmatrix} z_{t,x} \\ z_{t,y} \end{bmatrix} \tag{B.5}$$

The Jacobian for the inverse measurement model with respect to the predicted measurement z_t is:

$$G_m = \frac{\partial h^{-1}(\mu_t, z_t)}{\partial z_t} = \begin{bmatrix} \cos\mu_{t,\theta} & -\sin\mu_{t,\theta} \\ \sin\mu_{t,\theta} & \cos\mu_{t,\theta} \end{bmatrix} \tag{B.6}$$

The Jacobian for the inverse measurement model with respect to the vehicle pose μ_t is:

$$G_\mu = \frac{\partial h^{-1}(\mu_t, z_t)}{\partial \mu_t} = \begin{bmatrix} 1 & 0 & -z_{t,x}\sin\mu_{t,\theta} + z_{t,y}\cos\mu_{t,\theta} \\ 0 & 1 & -z_{t,x}\cos\mu_{t,\theta} - z_{t,y}\sin\mu_{t,\theta} \end{bmatrix} \tag{B.7}$$

Appendix C

Hardware & Software Specifications

	ADS-DV Computer	Author's Laptop
Model	CQ67G fanless PC	Dell Inspiron 7580
Processor	Intel Core i7-6700TE	Intel Core i7-8565U
Processor cores	4	4
Processor threads	8	8
RAM	16GB	16GB
GPU	Nvidia GTX 1050 Ti	Nvidia MX150
Operating System	Ubuntu 22.04 LTS	Ubuntu 22.04 LTS

Table C.1: Hardware and software specifications for the ADS-DV computer used at the competition and during data collection, and the author's laptop used for obtaining the results.