A SLAM Method for the Formula Student Driverless Competition

Luís Afonso Nazaré Correia Lopes

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Dr. Pedro Daniel dos Santos Miraldo
Prof. Pedro Manuel Urbano de Almeida Lima

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Dr. Pedro Daniel dos Santos Miraldo
Member of the Committee: Prof. José António Da Cruz Pinto Gaspar

October, 2021

A SLAM Method for the Formula Student Driverless Competition

Luís Afonso Nazaré Correia Lopes

October, 2021

Dedicated to my grandparents.

Abstract

The world of autonomous driving has received much attention in recent years, propelled by the constant pressure from governments and society for safer vehicles and roads, allied with the technological advances in the fields of computer vision and motion planning.

In a constant effort to follow the trends of the automotive world, Formula Student Germany introduced the *Driverless* class, where prototypes must be able to compete in a number of different events fully autonomously, with no prior knowledge concerning the layout of the track.

Simultaneous Localization and Mapping (SLAM) addresses the problem of localizing a moving agent while simultaneously constructing a map of the environment. SLAM is widely considered in the literature as one of the most challenging problems regarding autonomous applications, even more when applied in racing conditions.

This thesis aims to provide a comprehensive review of SLAM algorithms in the Formula Student Driverless context. Three different algorithms were implemented from scratch and tested both in simulation and real scenarios. The algorithms consist of two particle filter approaches and a graph-based one. Furthermore, a new approach to the data association problem which combines tracking information with traditional methods is proposed and compared against others commonly used.

The results show that the proposed graph-based pipeline held considerably better results when compared to the filtering approaches whilst being significantly more efficient. Moreover, regarding the data association problem, the proposed method also produced the best results among the other algorithms in study.

Keywords: SLAM, Data Association, Autonomous Driving, Formula Student

Resumo

A área da condução autónoma tem sido alvo de bastante atenção nos últimos anos devido à constante pressão, por parte das autoridades e sociedade em geral, para veículos e estradas mais seguros, aliado aos avanços nas áreas da visão por computador e controlo.

Num esforço constante para seguir as tendências do mundo automóvel, a *Formula Student Germany* introduziu a classe *Driverless*, onde os protótipos devem completar um conjunto de eventos de forma totalmente autónoma sem qualquer conhecimento prévio da pista.

Simultaneous Localization and Mapping (SLAM) aborda o problema de localizar um agente em movimento enquanto, simultaneamente, constrói um mapa do ambiente que o rodeia. SLAM é considerado pela comunidade científica como um dos problemas mais complexos relativos à condução autónoma, especialmente quando aplicado a ambientes de corrida.

Esta tese tem por objectivo realizar uma análise comparativa dos métodos de SLAM mais utilizados no ambiente de *Formula Student.* Três algoritmos distintos foram implementados de raiz e testados tanto em simulação como em dados reais. Os algoritmos em estudo consistem em duas abordagens baseadas em filtros de partículas e outra baseada em grafos. Para além disso, é ainda proposto um novo método para o problema de *data association*, que combina informação de rastreamento com métodos tradicionais.

Os resultados mostram que a implementação baseada em grafos obteve resultados consideravelmente melhores do que as implementações baseadas em filtros de partículas, sendo ao mesmo tempo bastante mais eficiente computacionalmente. Relativamente ao problema de *data association*, o método proposto, quando comparado com os restantes métodos em estudo, apresentou também os melhores resultados.

Palavras-chave: SLAM, Data Association, Condução Autónoma, Formula Student

Acknowledgements

First, I have to thank my supervisor Dr. Pedro Miraldo for giving me the opportunity to develop this work and for all the continuous support. Also a word of appreciation to André and Gonçalo for all the knowledge and help.

Secondly, I have to thank my family. To my Mother and Father, for giving me all the conditions and support, not only during this thesis, but also throughout this journey in IST, and more recently FST. To my Sister and Brother, for always being by my side, even when my mood or patience was not at the highest levels. Finally, to my grandparents, for teaching me with so many great values, and to whom this thesis is dedicated. I will be forever thankful.

I also would like to thank my dear Inês, for the love, the kind words, and above everything, for the patience that always kept me going. Thank you for all the support in the past six years, without you this journey would have been so much more difficult.

To all my friends that helped me in the long nights of studying and never ending projects. We had a great time together.

To the entire FST10 team, it was a pleasure working with you all during these tough and strange times. I could not have asked for a better ending to this journey. I have learned so much during the past two years, thank you all for putting up with me.

Finally, to the Autonomous Systems department, what a group of incredible guys: João, Pedro, Francisco, Diogo, Bernardo, Miguel and Ivo. What an incredible adventure we have experienced together, I am so proud of what we accomplished together. Thank you for all the good arguments, late night sprints, endless workshop sifts and long testing days.

A special word of appreciation to Regimento de Artilharia N°5, in the person of Col. Vasco António for the testing site, and Sgt. Valente for all the help with the ground truth maps.

Contents

List of Figures

List of Tables

Acronyms

ADAS Advanced Driver Assistance Systems. 1

AHRS Attitude and Heading Reference System. 36

EBS Emergency Brake System. 3, 37

EKF Extended Kalman Filter. 15

FSAE Formula Society of Automotive Engineers. 3

FSG Formula Student Germany. 2, 35

FST Formula Student Lisboa. 5

GPS Global Position System. 6, 36

HV High Voltage. 35

IC Individual Compatibility. 9, 30

ICP Iterative Closest Point. 51

IMU Inertial Monitoring Unit. 36

JC Joint Compatibility. 9, 31

LiDAR Light Detection and Ranging. 35

MCL Monte Carlo Localization. 19, 47

ML Maximum Likelihood. 9, 29

MPC Model Predictive Control. 6

RANSAC Random Sample Consensus. 38

RES Remote Emergency System. 3, 37

RMSE Root Mean Squared Error. 58

ROS Robot Operating System. 37

SCNN Sequential Compatibility Nearest Neighbor. 9, 30

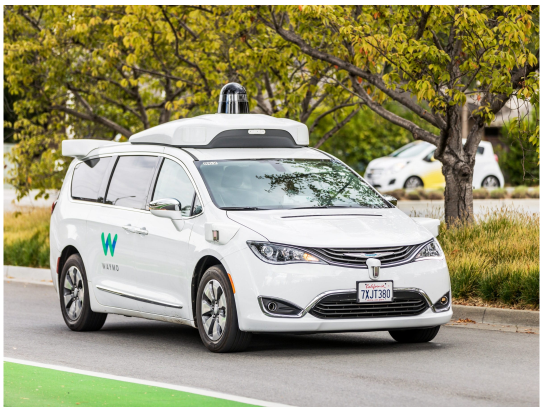
SLAM Simultaneous Localization and Mapping. 5, 6

Chapter 1

Introduction

The world of autonomous driving has received much attention in recent years, propelled by the constant pressure from governments and society for safer vehicles and roads and the ever-growing availability of sensors, like cameras and radars, in the current days' vehicles [1].

Although in a decreasing trend, it is estimated that 40 000 people lose their lives in road accidents each year on European roads, with the cause of 90% of those accidents being attributed to human error. To counteract this, and since higher levels of autonomy have the potential to drastically reduce dangerous driving behaviors, vehicle manufacturers are introducing at a rapid pace in-vehicle safety features and Advanced Driving Assistance Systems (ADAS). The latter combined with the technological advances in the fields of computer vision and motion planning, leads us to the state of the art regarding autonomous driving, with several manufacturers already testing their prototypes in real-life scenarios without any human interference, as presented in Figure 1.1. Examples include the self-driving taxi in Figure 1.1(a) developed by Waymo, and Cruise in Figure 1.1(b), that use primary LiDAR's to reach Level 4 autonomy, meaning that they do not require any human intervention during the vehicle's operation nor require the vehicles to be equipped with a steering wheel or pedals. On the other hand, a crucial aspect of any autonomous application is the ability to operate in the limits of handling, *e.g.* by performing emergency/avoidance maneuvers, and that is where racing comes in hand. Autonomous racing competitions such as Roborace [2] and Formula Student, in Figures 1.1(c) and 1.1(d), present a unique opportunity to develop, test and validate new technologies under challenging conditions.



(a) Waymo (Google Company) self-driving taxi in the streets of Phoenix, Arizona;

(b) Cruise autonomous prototype was recently tested in the streets of San Francisco without backup driver;

(c) Roborace a worldwide competition for autonomous driving electrical prototypes;

(d) Formula Student Driverless, a worldwide competition for engineering students;

Figure 1.1: Current state of the art regarding autonomous driving in the social context in (a) and (b) and in the competition context (c) and (d)

1.1 Formula Student Competition

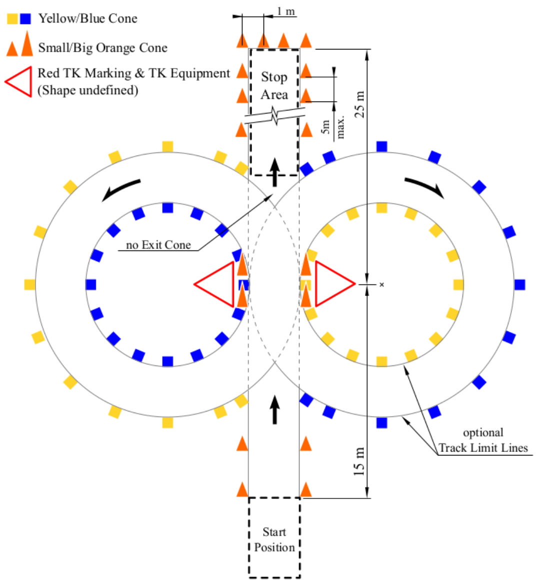
The Formula Student competition is Europe's most established educational engineering competition that challenges engineering students from the best universities around the world to design, build and test electric or combustion race cars according to a strict set of rules [3] and then compete against other teams in competitions organized all over the world. Much more than solely racing, Formula Student being an engineering competition, means that the fastest car does not necessarily win, but rather the team with the best overall package, both in terms of design, construction, performance and finances.

In a constant effort to follow the trends of the automotive world, in 2017, one of the most important competition organizers, Formula Student Germany (FSG), introduced a new class of vehicles, the *Driverless* class. In this class, the prototypes must be able to compete in a number of different events fully autonomously, with no human intervention and with no prior knowledge concerning the layout of the track.

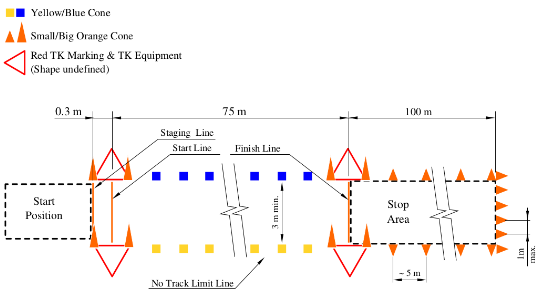
The competitions take place in famous racetracks around the world and follow a strict set of regulations established by the Formula Society of Automotive Engineers (FSAE) that must be met by every participating prototype before it can race, in an event called *technical inspection*. In this event, all the safety aspects related to the mechanical, electrical and autonomous characteristics of the prototype are evaluated and determine whether it is safe for competing. Having passed the technical inspection, the competitions are divided into two sets of events: static events and dynamic events. The static events evaluate the technical aspects related to the design, conception and sustainability of the prototype, and include an Engineering Design event, Cost and Manufacturing and a Business Plan Presentation where the car concept inserted in a profitable company project is presented to a board of judges/investors. In the dynamic events is where the prototypes show their actual racing capabilities and comprise a total of 5 events, as shown in Figure 1.2:

- **Skid Pad (1.2(a)):** the car must follow a track delimited by two pairs of concentric circles in an eight pattern figure;
- **Acceleration (1.2(b)):** the car must follow a straight course of 75 meters in length;
- **Autocross (1.2(c)):** the car must complete one lap to an unknown 200 to 500 meters long track composed of several corners and straights;
- **Trackdrive (1.2(d)):** the car must complete ten consecutive laps around the same track used in autocross;
- **Efficiency:** a ponderation between the energy spent in the trackdrive event and the elapsed time;

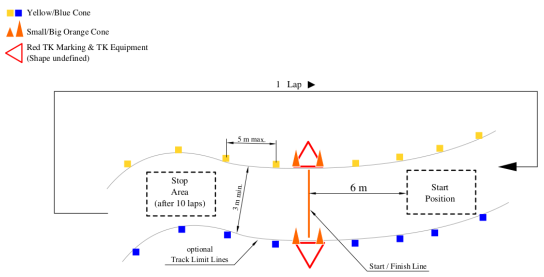
As shown in Figure 1.2, for the DV class vehicles the track boundaries are delimited by blue cones on the left and yellow cones on the right, small orange cones delimit

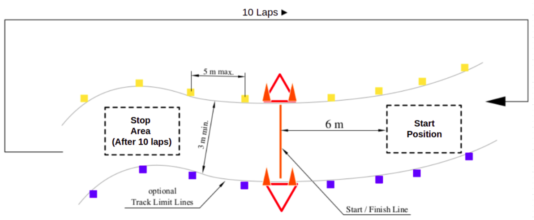


(a) Skid Pad Event;



(b) Acceleration Event;


Yellow/Blue Cone
Small/Big Orange Cone
Red TK Marking & TK Equipment (Shape undefined)
1 Lap
5 m max.
Stop Area (after 10 laps)
3 m min.
6 m
Start Position
optional Track Limit Lines
Start / Finish Line
(c) Autocross Event;



(d) Trackdrive Event;

Figure 1.2: Formula Student Dynamic Events for the Driverless Class vehicles. Figures obtained from [4].

stopping zones and big orange cones mark timekeeping zones. The shape, dimensions and color pattern of these cones must always be the same, as shown in Figure 1.3, and are regulated by the FSG rules [3]. Since the layout of the track is unknown, in order to safely navigate through the unknown environment the prototype must identify the cones' position and color using the data retrieved from the available perception sensors. Having detected the cones that delimit the track, the prototype then needs to compute a valid path between the track boundaries, comprising the path planning pipeline. Finally, in order to navigate through the track using the computed path one needs to determine a speed target and a steering input, which is performed by the control pipeline. During dynamic events, the only way of communication with the vehicle is via the Remote Emergency System (RES), responsible for starting by sending the *Go Signal* or stopping in case of emergency by deploying the Emergency Brake System (EBS).

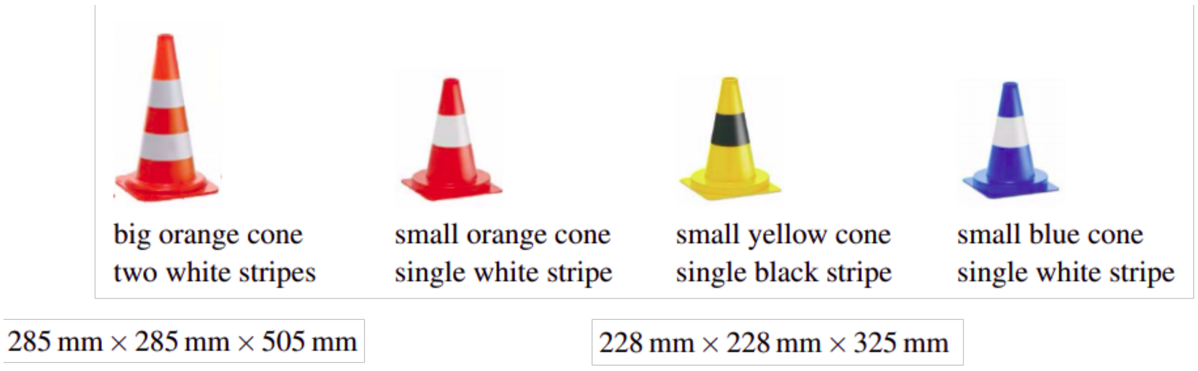


Figure 1.3: Specifications of the cones used in Formula Student Driverless competitions. Figure taken from [4].

1.2 Motivation

The Formula Student Lisboa team (FST), from Instituto Superior Técnico, has been developing prototypes for this competition since 2001, with ten prototypes developed so far and proven results in the most prestigious competitions of Europe. The team decided to embrace this new challenge and adapt the previous prototype (FST09e) to be fully autonomous and compete in the *Driverless* class in the summer of 2020. Initially composed of only eight students from various engineering courses, we developed a complete autonomous pipeline [5], comprising perception, estimation and control pipelines.

Due to the COVID-19 pandemic that plagued the world, the competitions did not take place, but an opportunity for extensive testing emerged highlighting some problems, which is to be expected since this was the team's first year developing autonomous prototypes, even though the pipelines were thoroughly tested in simulation and using synthetic data.

A key part when dealing with any kind of autonomous driving problem is being able to localize the vehicle on a given map, or, in the order hand, in case no map is available, construct a map of the environment given the data retrieved from the sensors and location of the vehicle. This being said, a *chicken-or-egg* like problem arises when neither a map is available or the localization of the vehicle is known, since both vehicle's motion and observation models are subject to noise, the mapping problem necessarily induces a localization problem. This problem is formally known as Simultaneous Localization and Mapping (SLAM), and tackles the problem of constructing a map whilst simultaneously localizing the agent within it [6]. This problem is widely considered in the literature as one of the most difficult [7] and at the same time essential for truly autonomous vehicles, and can become particularly overwhelming in racing conditions due to the higher speeds requiring very efficient algorithms running on low-level architectures with limited power consumption.

While the prototype is navigating through the unknown track, in parallel with the pipeline [5] described at the end of Section 1.1, a SLAM algorithm is also running in order to construct a map of the environment. This map is particularly useful for the trackdrive event since having an accurate representation of the track is a key requirement for algorithms like Model Predictive Control (MPC) [8] which exploit both the limits of the car and the track by optimizing the speed and trajectory, rather than depending only on the current observations for computing the best path to follow. Additionally, several SLAM techniques can easily be converted to localization-only algorithms, a feature that can be particularly useful in some events that will be discussed in Section 2.1.

The previous SLAM implementation within the team did not held satisfactory results both in terms of mapping and localization capabilities. To overcome this limitation an extensive research and testing of different SLAM algorithms was conducted, which resulted in the implementation from scratch of two types of SLAM approaches. The first one being an improved version of the previous pipeline and the second a graph-based approach where a new method for localization was proposed along with a different method for data association.

1.3 Topic Overview

Simultaneous Localization and Mapping (SLAM) is an essential capability for any mobile robot exploring an unknown environment. SLAM addresses the problem of a robot moving through an environment of which no *a priori* map is available. The aim of SLAM is to acquire a map of the environment, using a moving robot, while simultaneously localizing the robot relative to this map [9].

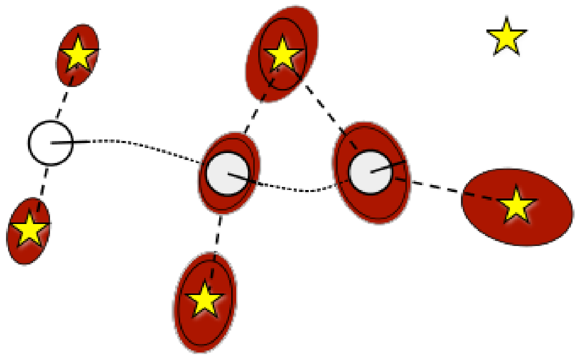


Figure 1.4: The duality of the SLAM problem, both the path (white circles) and the position of the landmarks (yellow stars) are unknown and both are subject to uncertainty (red ellipses) that grows over time. This means that the mapping problem necessarily induces a localization problem because observations are corrupted by error in the pose estimate.

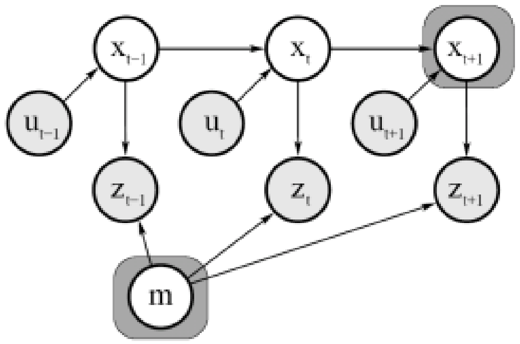
As the robot moves, it collects odometry measurements, for example, from GPS or wheel encoders, however, these measurements are subject to error, the so called motion noise. In addition, the robot also collects information of its surroundings in order to construct a map, which not only are subject to error by the sensor measurement itself (observation noise), but also because they are corrupted by error in the pose estimate. However, unlike observation noise, the error in the pose estimate will have a systematic effect on the error in the map, or put into other terms, error in the robot's path correlates errors in the map, as stated in [10]. As a result, the true map cannot be estimated without also estimating the true path of the robot, a relationship that was first identified in [11] and evidenced in Figure 1.4.

Since then several approaches to solve the SLAM problem have emerged and various problems identified, one of them being the data association problem that will be introduced in Section 1.3.2 and further discussed in Section 2.5.

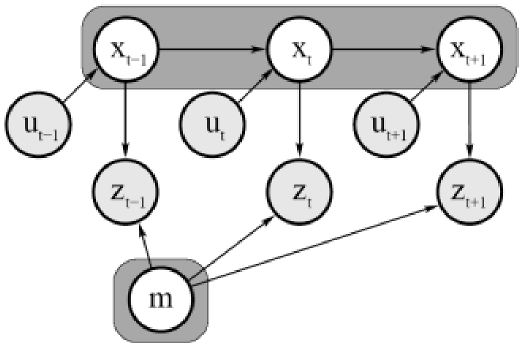
1.3.1 Online SLAM and Full SLAM

From a probabilistic point of view, there are two ways in which the SLAM problem can be formulated into, both of them being equally relevant in terms of practical importance [12].

The first one is formally known as *online SLAM problem*, and tackles the problem of



(a) Online SLAM problem graphical problem formulation;



(b) Full SLAM problem graphical problem formulation;

Figure 1.5: Types of formulations possible for the SLAM problem. As the robot moves from x_{t-1} to x_t as a result of the control command u_{t-1}, observes the landmark m through the observation z_{t-1}. Online slam approaches, in (a), seek to estimate the current position of the robot x_{t+1} along with the position of the landmark m. Whereas full slam approaches, in (b), seek to estimate the most recent robot position along with the collection of the previous pose estimates, as well as the position of the landmark m.

estimating the posterior over the current pose along with the position of the landmarks, forming the map, according to:

$$p(x_t, m|z_{1:t}, u_{1:t}). \tag{1.1}$$

This problem is called the online SLAM problem, in Figure 1.5(a), because it only seeks to estimate the variables at the present instant in time t, and therefore many of the algorithms used for solving the online SLAM problem are incremental, meaning they discard past measurements and controls as they are processed [12]. Typically, these approaches are called filtering approaches, meaning that the estimate is augmented as new measurements become available and it is exactly to highlight their incremental nature that they are referred to as online SLAM methods [13].

The second one, contrarily to the first one, seeks to estimate not only the posterior over the current pose, but also the posterior over the rest of the path $x_{1:t}$ along with the map, as in:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}). \tag{1.2}$$

These approaches address the *full SLAM problem*, in Figure 1.5(b), and they typically rely on least-square error minimization techniques, hence being also known as smoothing approaches to the SLAM problem [13].

Regardless of the formulation chosen (online or full), estimating the full posterior has a big importance since it captures all there is to know about the map and pose or path. However, computing the full posterior was unfeasible for a number of years due to the complexity and dimensionality of the state variables, which makes it impossible to compute probability distribution in real-time without making approximations [12].

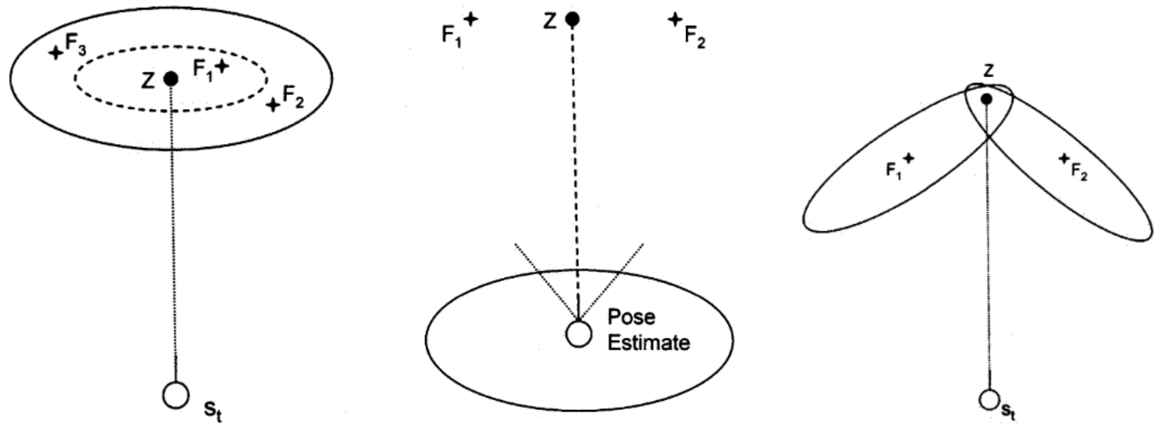
1.3.2 Data Association

As stated previously, although throughout the years several achievements have been made regarding the SLAM problem, SLAM still remains one of the most difficult problems when it comes to an autonomous vehicle moving through an unknown environment. One of the topics that make the SLAM problem hard to solve is the *data association problem*, a problem that arises from the fact that the mapping between the observations and features in the map is unknown and errors in the data association process can lead to catastrophic failures as the divergence of the whole SLAM algorithm [14].

In many real-world problems, landmarks are not identifiable and the total number of landmarks cannot be obtained trivially. The data association process consists of determining, from a set of observations, the ones that correspond to new landmarks and the ones that correspond to already observed landmarks. The data association problem is imposed by the fact that both measurements, pose and landmark locations are subject to uncertainty [15], see Figure 1.6.

The measurements uncertainty, in Figure 1.6(a), leads to data association ambiguity since the larger the uncertainty associated to the measurement is, the larger the number of possible landmark assignments that need to be considered. In a similar way, the uncertainty associated to the pose, in Figure 1.6(b), creates ambiguity due to the fact that the agent is uncertain about its location (position and orientation). Finally, landmark uncertainty, in Figure 1.6(c), leads to assignment ambiguity, since it allows, in a probabilistic sense, for a measurement to be associated with multiple landmarks.

The premise for feature-based data association is that credible features can be extracted from the environment by the sensors. The latter can be particularly difficult in unknown environments and due to the intrinsic uncertainty of the sensors, making, to



(a) Ambiguity caused by measurement uncertainty; (b) Ambiguity caused by pose uncertainty; (c) Ambiguity caused by landmark uncertainty;

Figure 1.6: Ambiguities in the data association process caused by the inherent noise present in measurements, pose estimate and landmark position estimate. Figure obtained from [15].

this day, feature-based data association a difficult research problem [16], with several algorithms being proposed in the literature, such as Maximum Likelihood (ML) [17], Individual Compatibility (IC) [15], Sequential Compatibility Nearest Neighbor (SCNN) [14] and Joint Compatibility (JC) [18]. In Section 2.5 these methods will be discussed in-depth, implemented and the results compared in different SLAM approaches.

1.4 Objectives and Achievements

Keeping in mind the overview presented above, the following objectives are defined:

- Improve the data association process regardless of the SLAM algorithm in use by combining standard data association techniques with tracking information;
- Implement two state-of-the-art feature-based SLAM approaches (FastSLAM 2.0 and GraphSLAM);
- Compare the accuracy and performance of the implementations both using standard data association techniques and using the proposed data association method against each other and the original implementation;

- Implement a localization only algorithm based on GraphSLAM that allows for a quick transition into a localization only mode in case a map is already available;
- Compare the localization capabilities of both FastSLAM 2.0 and GraphSLAM against each other and the original implementation;

Such objectives were defined with the main goal of empowering the FST team with knowledge regarding available SLAM techniques and an algorithm capable of handling all the scenarios in which it is required in the upcoming competitions.

The major achievement with this thesis is, undoubtedly, a SLAM module that provides accurate enough maps in the Formula Student competition context, something that the previous implementation was not able to achieve. The current pipeline not only fulfills the current requirements of the team, but also is faster, modular and more robust overall using the proposed data association method. Lastly, by comparing different SLAM approaches applied to the Driverless competition several limitations, given our current setup, were identified. Knowing these kinds of limitations is very important for the future of the Autonomous Systems department and will greatly contribute for the development of more complex algorithms, using different sensors combinations, which will culminate in a much faster, robust and safer car.

1.5 Thesis Outline

In chapter 2 a background on the Formula Student Competition with focus on the Driverless class is presented. Furthermore, a theoretical overview of the SLAM problem in general and the algorithms to be studied in particular is provided. Finally, an overview of the data association methods in use is presented.

chapter 3 presents the vehicle setup along with the algorithms used in both perception and state estimation pipelines. Additionally, the key particularities and changes that had to be made to the original algorithms in order for them to work in the Formula Student environment are discussed, including the new data association method proposed and the localization only algorithm based on GraphSLAM.

Chapter 4 shows the performance of the algorithms in terms of computational time, data association accuracy and overall map accuracy at different speed profiles. These results are divided into simulation a real scenario results.

In chapter 5, conclusions about the developed work are given along with a track of the fulfilled objectives and suggestions for future work.

Chapter 2

Theoretical Background

In this chapter a theoretical background on SLAM is presented by splitting the problem into two separate ones, a *mapping problem* and a *localization problem*, as well as an explanation on why both of them are important in the Formula Student competition context.

Finally, a theoretical formulation of one of the most common online SLAM approaches based on a particle filter proposed in [19], the *FastSLAM* algorithm is presented in Section 2.3, along with one of the state-of-the-art approaches to the full SLAM problem, presented in Section 2.4, via a graph-based formulation, the *GraphSLAM* algorithm, initially proposed in [20].

2.1 Mapping and Localization Phases

As introduced in Section 1.1, the track where the autocross and trackdrive event takes place is previously unknown and since the landmarks present in the environment are of similar appearance and can only be distinguished by their position and color, a SLAM algorithm that allows for probabilistic landmark incorporation is required. Furthermore, errors in the mapping process during the autocross event are unforeseen and so, the algorithm must not only be able to run in real-time, but also transition to a localization phase when loop closure is detected in the event of the map has to be acquired during the trackdrive event.

Having an algorithm that can quickly transition to a localization-only method is particularly useful not only because it eliminates the necessity of having another algorithm especially dedicated to the localization side of things, but also because the track layout for the skidpad is known beforehand. This being said, by having this duality of modes in the SLAM, the skidpad event can be completed by only providing a previously built map to the SLAM algorithm and only make use of its localization capabilities.

2.2 SLAM Theoretical Overview

As stated in Section 1.3, it is not possible to estimate the map without also estimating the location of the robot, a relationship that was first identified by [11] in their seminal paper on SLAM in 1986, and first implemented by [21].

In a typical SLAM problem, two types of information are available to the robot at a given moment in time: controls and observations. Controls can be taught as predictions of the robot's motion, which are subject to noise and, in a similar way, observations are noisy measurements of the robot's environment. Since both controls and observations are subject to noise, and this noise can be modeled, then they can be taught as probabilistic constraints, in Figure 2.1, where controls define a probabilistic constraint over two consecutive poses and observations constrain relative transformations between the robot and features in the map [17].

One of the possible ways to approach SLAM, would be to estimate the most likely pose and map using the complete set of controls and observations acquired until time t, formally known as solving the *full SLAM problem* as described in Section 1.3.1. These kinds of approaches to the SLAM problem, although very powerful, were discarded for several years due to the complexity of solving such problem by means of standard techniques, posed by the constraints which grow without bond over time [9], and was only possible due to the advances in the fields of sparse linear algebra [13]. An example of this approach will be discussed in detail in Section 2.4.

Another solution for the SLAM problem would be to estimate the posterior probability distribution over all possible maps m and robot poses s_t, conditioned to the full set of controls u^t and observations z^t as described in (1.1). This distribution is referred to as the SLAM posterior [17, 9]. Even though it could appear to be more difficult to estimate the latter, than the full posterior, if we model the way the world evolves by means of some simple assumptions it is possible to estimate the posterior with several advantages over the solutions that consider only the most likely state. This is justifiable

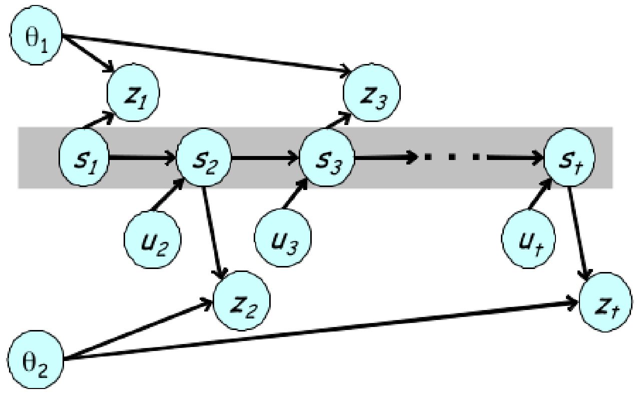


Figure 2.1: Illustration of the probabilistic constraints on the pose of the robot and features of the environment. The robot moves from s_1 to s_2 by input of the control u_2, forming a pose constraint. When in s_2 the robot observes the landmark θ_2 by making the measurement z_2, which in turn establishes a pose-landmark constraint. Figure obtained from [19].

by the fact that considering a distribution of possible solutions leads to a more robust algorithm in noisy environments [6].

The most common approach to the SLAM problem uses an Extended Kalman Filter (EKF) [22] for estimating the posterior distribution over the map and robot's pose [11]. This approach represents the map and pose estimate by means of a high-dimensional Gaussian. The off-diagonal entries of this multivariate Gaussian represent the correlation discussed earlier between errors in the robot's pose and features in the map, as shown in Figure 2.2. This approach allows to accommodate the nature of error in the map, as stated in [19], however, it has two big drawbacks.

The first drawback is the computational complexity, maintaining a multivariate Gaussian requires time and memory quadratic in the number of features in the map, limiting its application to relatively small maps and/or with a small number of features. This quadratic complexity is a consequence of the Gaussian representation in which the EKF is based. The uncertainty of the SLAM posterior is represented by a covariance matrix that reflects the correlations between all possible pairs of state variables, and consequently, the memory required to store the information of this matrix grows quadratically with the number of features in the map. Since all the correlations between pairs of state variables are maintained, incorporating a new sensor observation

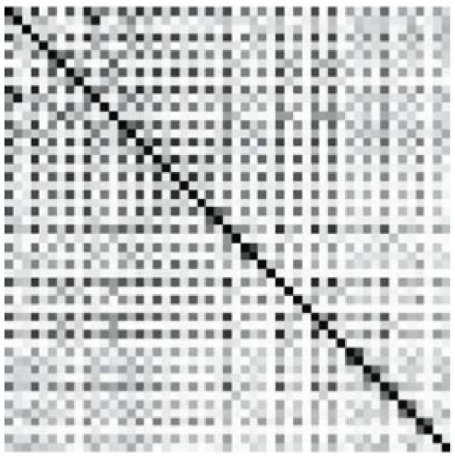


Figure 2.2: Illustration of the covariance matrix maintained by the EKF based SLAM, here represented as a correlation matrix. The darker the matrix entry, the higher the correlation between the pair of state variables. Figure obtained from [9].

requires performing an operation on every entry of the covariance matrix [6]. Several workarounds of this quadratic complexity have been proposed [23, 24, 25], where the basic idea is to decompose the problem of building a large map into a collection of smaller maps, which can then be updated efficiently.

The second drawback, and unequivocally more important, is related to the *data association problem*, introduced in Section 1.3.2. Different data association hypotheses necessarily induce multiple and different maps, this multi-modality cannot be represented using Gaussians. The standard approach to the data association problem in EKF is to assign an observation to a landmark based on the maximum likelihood rule, but since the EKF has no way of representing uncertainty over data associations, the effect of incorporating the wrong data association cannot be undone. Several data association errors will cause the filter to diverge [6]. Albeit other data associations methods are available, as the ones that will be discussed in Section 2.5, however, they do not address the fact that the EKF only considers one data association hypothesis per time step, besides the fact that they induct complexity to the algorithm and, due to the nature of the EKF, make it impossible to use in real time [9].

Bearing in mind the limitations imposed by the EKF approach, a new family of methods based on particle filters together with EKF's for estimation of the landmark locations was proposed by [19], and will be discussed further in detail in Section 2.3.

2.3 FastSLAM

Particle filters [26] are particularly useful when dealing with SLAM problems because they can approximate arbitrarily complex probability distributions, whereas the EKF approaches are limited to Gaussian approximations at all levels of uncertainty [15]. Besides, particle filters are robust to non-linearities in both motion and measurement models, since no linearization is required in the propagation of the state uncertainty. It is inherently carried along with the distribution of the particles. The main drawback of the particle filters is the problem of scalability for high dimensional spaces due to the exponential time complexity of the implementation, a problem that was overcome with the introduction of the Rao-Blackwellized particle filter [27].

The main advantage of the particle filter over the EKF approach is to allow for multi-hypothesis data association, since the posterior is represented by multiple particles. Therefore, particle filters enable the data association to be done on a *per-particle* instead of on a *per-filter* basis, meaning that different particles may have different features correspondences or even different number of landmarks in their maps. The multi-hypothesis data association results in more robust algorithms when it comes to data association errors, since particles with wrong data associations are likely to disappear in the resampling process [6].

Hereinafter the SLAM problem will be formally described as a collection of N features, each of them denoted as θ_n, which together comprise the map m. The vehicle pose, comprising the vehicle's two-dimensional Cartesian coordinates along with its angular orientation is defined in discrete time as s_t and the complete path of the vehicle up to time t as s^t. Additionally, in order to construct a map, the vehicle can sense. These measurements encode information concerning the range and bearing to a nearby landmark, as shown in Figure 2.3. In the name of simplicity and for the sake of this theoretical formulation, a single known landmark will be assumed to be observed at any given instant in time. It should be noticed that this is a matter of convenience and does not imply any loss of generality, since multiple observations can be incorporated sequentially and the data association problem will be tackled in Section 2.5.

At the core of SLAM there is a probabilistic law that describes the process according to which measurements are acquired, the *measurement model* as described in:

$$p(z_t|s_t, \theta_{nt}, n_t) = g(\theta_{nt}, s_t) + \epsilon_t. \tag{2.1}$$

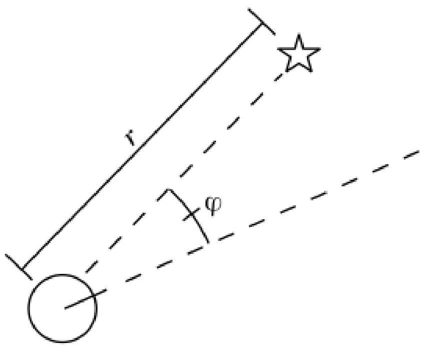


Figure 2.3: Illustration of vehicle observing a range r and a bearing φ to a nearby landmark. Figure obtained from [19].

The measurement model is conditioned to the vehicle's pose s_t, the identity of the landmark n_t and the corresponding feature θ_{nt}. This probability is a function of g, non-linear in the sense that the range and bearing are obtained by trigonometric relations, plus a distortion by noise modeled as ϵ_t with zero mean and covariance R_t.

As stated earlier, the second input to the SLAM problem are the controls of the vehicle, denoted as u_t. Similarly to the measurement model, also the controls are modelled as a probabilistic law that describes the evolution of the poses according to:

$$p(s_t|u_t, s_{t-1}) = h(u_t, s_{t-1}) + \delta_t \,, \tag{2.2}$$

the *motion model*.

The motion model describes the current pose as a function of h, given the previous pose and control command, perturbed by Gaussian noise δ_t with zero mean and covariance P_t. As it was the case with the measurement model, function h is also non-linear with several methods being discussed in Section 3.3.

FastSLAM [10] exploits a property of the SLAM problem pointed out in [28], that concerns the fact that feature estimates are conditionally independent given the robot path meaning that the correlations in the uncertainty of features in the map only arise from the uncertainty associated to the pose. The latter means that if the true path of the vehicle was known, then the error in the landmarks estimates would be independent from each other. This allows for the posterior over the possible maps and features in (1.2) to be represented in a factored way [10]. FastSLAM implements this factored representation by using a particle filter for estimating the path, which means that conditioned to each particle the individual map errors are independent, hence the factorization into separate mapping problems, one for each error. In this approach each particle possesses N EKF for estimating the N landmark locations conditioned to the path estimate. The posterior in (1.2) can be converted into the Bayes Filter equation by making use of the Bayes Rule:

$$p(s_t, m|z^t, u^t, n^t) = \eta\, p(z_t|s_t, \theta_{nt}, n_t) \int p(s_t|s_{t-1}, u_t)p(s_{t-1}, m|z^{t-1}, u^{t-1}, n^{t-1})\, ds_{t-1}\,, \tag{2.3}$$

where η is just a normalization constant.

The Bayes filter in (2.3) is equivalent to the Kalman Filter when both g and h are linear, whereas the EKF allows for non-linear g and h by obtaining a linear approximations through the first order Taylor expansion.

By exploiting the conditional independence property of SLAM, one can rewrite (2.3) in a factorized form:

$$p(s^t, m|z^t, u^t, n^t) = p(s^t|z^t, u^t, n^t) \prod p(\theta_n|s^t, z^t, n^t). \tag{2.4}$$

In (2.4) is evident the separation of the SLAM problem into $N+1$ recursive problems, one over the vehicle's path $p(s^t|z^t, u^t, n^t)$, and N separate landmark estimation problems $p(\theta_n|s^t, z^t, n^t)$. It should be noticed that although this is factored representation of the posterior, it is exact and not just a general approximation [17]. Following a typical Dynamic Bayes Network (DBN) terminology, from (2.4) it is clear that this approach to the SLAM problem *d-separates* the individual feature estimation problems by rendering them independent of each other [9] and that knowledge concerning the location of a given landmark will not give or contribute with any information relatively to the location of the remaining features in the map.

2.3.1 FastSLAM 1.0

The FastSLAM [10] algorithm divides the SLAM problem into two separate ones. A localization problem using a particle filter, where each particle has its own N landmark estimation problems conditioned to the path estimate that compose the mapping problem.

The particle filter is used for estimating the path posterior $p(s^t|z^t, u^t, n^t)$ in (2.4) using a similar filter to the one used in the *Monte Carlo Localization* MCL [29], by maintaining a set of particles S_t that represent the posterior at each point in time, where each particle $s^{t,[m]} \in S_t$ represents a *belief* of the vehicle's path. The superscript notation $[m]$ is used to refer to the *m-th* particle in the particle set.

The particle set S_t is incrementally computed from the previous set S_{t-1}, a control u_t and a measurement z_t. First, each particle $s^{t,[m]} \in S_{t-1}$ is used to generate a probabilistic guess of the robot's pose at time t according to:

$$s_t^{[m]} \sim p(s_t | u_t, s_{t-1}^{[m]}) \,, \tag{2.5}$$

obtained by sampling from the probabilistic motion model in (2.2).

Under the asymptotically correct assumption that the set of particles in S_{t-1} is distributed according to $p(s^{t-1}|z^{t-1}, u^{t-1}, n^{t-1})$, the temporary particle set comprising the new estimate along with the previous path $s^{t-1,[m]}$ is also distributed according to $p(s^t|z^{t-1}, u^t, n^{t-1})$ and is referred as the *proposal distribution*.

After generating all the M particles, according to (2.5), the new particle set S_t is obtained by sampling, from the temporary particle set, M particles $s^{t,[m]}$ with replacement and a probability proportional to the *importance factor* $w_t^{[m]}$, which is calculated according to:

$$w_t^{[m]} = \frac{target\ distribution}{proposal\ distribution} = \frac{p(s^{t,[m]} \mid z^t, u^t, n^t)}{p(s^{t,[m]} \mid z^{t-1}, u^t, n^{t-1})} . \tag{2.6}$$

It should be noticed that this quantity can only be computed in closed form because the conditional landmark location estimates $p(\theta_n|s^t, z^t, n^t)$ in (2.4) are represented using Kalman Filters. From (2.4) it is obvious that since this estimate is conditioned to a robot pose, all the Kalman Filters are attached to an individual particle pose in the sample set S_t, comprising the full posterior over paths and landmark positions according to:

$$S_t = \left\{ s^{t,\,[m]}, \mu_1^{[m]}, \Sigma_1^{[m]}, \cdots,\ \mu_k^{[m]}, \Sigma_k^{[m]} \right\}_m \,, \tag{2.7}$$

where $\mu_i^{[m]} \in \mathbb{R}^2$ and $\Sigma_i^{[m]} \in \mathbb{R}^{2\times 2}$ are the mean and covariance of the Gaussian representing the *i-th* landmark θ_i, attached to the *m-th* particle.

To obtain the posterior over the *i-th* landmark θ_i it is only required knowledge whether the landmark was already observed or not. If not then the Gaussian is left unchanged, otherwise an update is done through an EKF, by performing a linearization of the observation model in (2.1) according to:

$$G_\theta = \bigtriangledown_{\theta_{n_t}} g(\theta_{n_t}, s_t)\Big|_{s_t=\hat{s}_t^{[m]};\, \theta_{n_t}=\hat{\theta}_{n_t}^{[m]}}, \tag{2.8}$$

where $\bigtriangledown_{\theta_{n_t}}$ represents the Jacobian of the measurement model with respect to the landmark location θ_{n_t}.

The posterior over the landmark mean and covariance can then obtained using the standard EKF update equations [22]:

$$\begin{aligned} K_t &= \Sigma_{n_t,t-1}^{[m]} G_{\theta_{n_t}}^T Q_t^{[m]-1} \\ \mu_{n_t,t}^{[m]} &= \mu_{n_t,t-1}^{[m]} + K_t\,(z_t - \hat{z}_t) \\ \Sigma_{n_t,t}^{[m]} &= (I - K_t G_{\theta_{n_t}})\Sigma_{n_t,t-1}^{[m]}\,, \end{aligned} \tag{2.9}$$

where $Q_t^{[m]}$ is the innovation covariance matrix defined by:

$$Q_t^{[m]} = R_t + G_\theta \Sigma_{n_t,t-1}^{[m]} G_\theta^T \tag{2.10}$$

The key characteristic of the FastSLAM use of the EKF is that the update is performed to a Gaussian of just two dimensions since each landmark has its separate EKF, instead of the typical EKF-based SLAM which requires maintaining a covariance matrix that comprises the pose estimate along with the location of all the landmarks. This allows the update to be performed in constant time, instead of the quadratic time required by the EKF, which in the end leads to better scalability.

Although FastSLAM 1.0 opened the door for applying SLAM techniques to environments with more than a few hundred features, it also came with its drawbacks, such as the number of particles required for convergence. This lead to a refinement in the algorithm with a new proposal distribution being suggested, which in turn resulted in a much more accurate and robust algorithm that will be presented in Section 2.3.2, FastSLAM 2.0.

2.3.2 FastSLAM 2.0

The FastSLAM algorithm proposed in 2.3.1 leads to efficient scaling and robust data association, requiring only *O(NM)* in terms of memory, whereas the update step requires *O(M* log *N)*, even with unknown data association. However, it also has its drawbacks associated to its particle filter nature. One is the fact that the performance

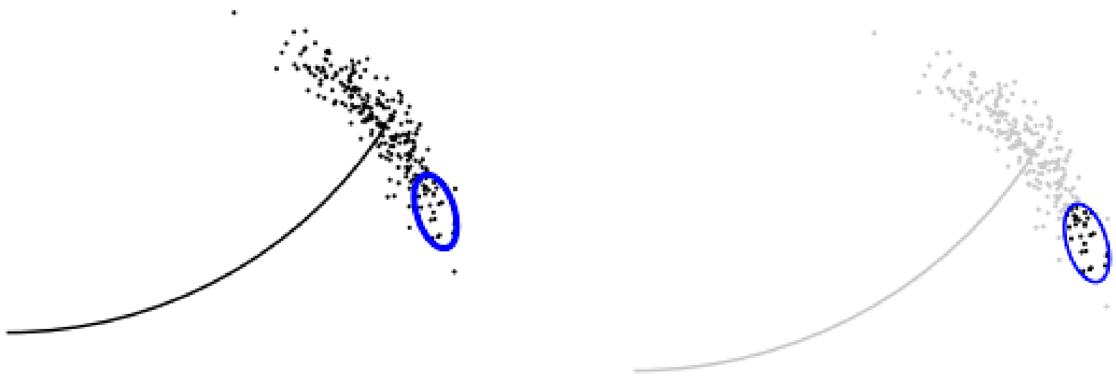


Figure 2.4: Mismatch between proposal and posterior distributions. All the particles outside the ellipse in (a) will receive negligible probability, while the remaining ones inside the ellipse in (b) will be replicated multiple times in the resampling process. Figure obtained from [17].

of the algorithm will degrade when the motion of the vehicle is noisy relative to the observations, a typical problem since most mobile robots have high values of control noise but relatively accurate sensors, causing the proposal distribution to be poorly matched with the posterior, as Figure 2.4 shows. The second is the number of particles required for convergence, although this value is unknown it is suspected, in the worst case, to be exponentially proportional to the size of the map [9].

In this section a new version of the FastSLAM [30] algorithm that incorporates into the proposal distribution not only the importance, but also the current observations in order to obtain a better posterior will be presented. This new version of FastSLAM also proves the convergence of the algorithm for linear SLAM problems, even for a single particle.

In regular FastSLAM, the pose $s_t^{[m]}$ is sampled from (2.2) according to motion command u_t, not taking into account the measurement z_t. Instead, this measurement is only incorporated in the resampling process in (2.6). This approach can be troublesome when the motion model is noisy relative to the measurement model, causing the sampled poses to fall into areas of low measurement likelihood (see Figure 2.4(a)), and subsequently disappearing (see Figure 2.4(b)) in the resampling process with high probability. As the observations become more accurate, fewer unique samples will survive each update step, eventually causing the filter to diverge [17].

FastSLAM 2.0 implements the idea that poses are sampled under the consideration of both the motion u_t and the measurement z_t, according to:

$$s_t^{[m]} \sim p(s_t | s^{t-1,[m]}, u^t, z^t, n^t)\,, \tag{2.11}$$

which explicitly incorporates the most recent sensor measurement z_t, its a data association n_t, the most recent control u_t, and where $s^{t-1,[m]}$ is the path up to $t-1$ of the *m-th* particle.

The proposal distribution in (2.11) can be divided into the product of two factors: the next state distribution $p(s_t | s_{t-1}^{[m]}, u_t)$ and the probability of the measurement z_t. Obtaining the probability of the measurement requires integration over the possible landmark locations θ_{n_t}, which is not possible without approximating the measurement model g in (2.12) to a linear function:

$$g\left(\theta_{n_t}, s_t\right) \approx \hat{z}_t^{[m]} + G_\theta \cdot \left(\theta_{n_t} - \mu_{n_t}^{[m]}\right) + G_s \cdot \left(s_t - \hat{s}_t^{[m]}\right)\,, \tag{2.12}$$

where $\hat{z}_t^{[m]}$ represents the predicted measurement:

$$\hat{z}_t^{[m]} = g(\hat{\theta}_{n_t}^{[m]},\, \hat{s}_t^{[m]})\,, \tag{2.13}$$

$\hat{s}_t^{[m]}$ the predicted pose:

$$\hat{s}_t^{[m]} = h(s_{t-1}^{[m]},\, u_t)\,, \tag{2.14}$$

and $\hat{\theta}_n^{[m]}$ the predicted landmark location:

$$\hat{\theta}_n^{[m]} = \mu_{n,\,t-1}^{[m]}. \tag{2.15}$$

The matrix G_θ in (2.8) is Jacobian of g in respect to θ and G_s the Jacobian of g in respect to s:

$$G_s = \bigtriangledown_{s_t}\, g(\theta_{n_t}, s_t)\Bigg|_{s_t=\hat{s}_t^{[m]};\, \theta_{n_t}=\hat{\theta}_{n_t}^{[m]}} \tag{2.16}$$

According to this EKF approximation, one can rewrite the proposal distribution in (2.12) as a Gaussian with mean and covariance defined by:

$$\Sigma_{s_t}^{[m]} = \left[G_s^T Q_t^{[m]-1} G_s + P_t^{-1}\right]^{-1} \tag{2.17}$$

$$\mu_{s_t}^{[m]} = \Sigma_{s_t}^{[m]} G_s^T Q_t^{[m]-1} (z_t - \hat{z}_t^{[m]}) + \hat{s}_t^{[m]}\,, \tag{2.18}$$

respectively, and where $Q_t^{[m]}$ represents the innovation covariance matrix in (2.10).

The new sample distribution in (2.11) is now parameterized as a Gaussian approximation by (2.17) and (2.18). This Gaussian is constructed for each particle in the particle set S_{t-1}, and a new sample is drawn and placed in the temporary particle set according to:

$$s_t^{[m]} \sim \mathcal{N}(s_t;\, \mu_{s_t}^{[m]}, \Sigma_{s_t}^{[m]}). \tag{2.19}$$

The update of the conditional landmark location estimates remains similar to the one discussed in 2.3.1, since it is not affected by the changes in the proposal distribution. However, the same does not apply to the importance weights, which must be updated to reflect this change.

Recall that the importance weights in (2.6) are defined as the ratio between the target and proposal distributions, and the asymptotically correct assumption made previously, that the paths in $s^{t-1,[m]}$ were generated according to the target distribution one time step earlier, remains valid. The importance weight $w_t^{[m]}$ reflecting the new proposal distribution will be given by:

$$w_t^{[m]} = \frac{p(s^{t,\,[m]} \mid z^t, u^t, n^t)}{p(s^{t-1,\,[m]} \mid z^{t-1}, u^{t-1}, n^{t-1})\, p(s_t^{[m]} \mid s^{t-1,\,[m]}, z^t, u^t, n^t))}. \tag{2.20}$$

The new proposal distribution has another important ramification concerning the creation and update of the landmarks estimate. In the previous FastSLAM implementation, simultaneous observations were incorporated sequentially, each landmark filter was updated separately and the weight of the resulting particle was the product of the weights of each individually handled observation.

In this new implementation, because the observations must be incorporated into the proposal distribution, instead of throwing away the proposal distribution after drawing the sample, the proposal distribution is kept and updated for each observation, causing it to shrink. New samples are sequentially generated from the incremental proposal distribution in order to update the landmark filters and compute the importance weights.

Concerning the creation of new landmarks, [9] refers that observations that generate new landmarks should be processed last since the new robot pose $s_t^{[m]}$ will be drawn from the standard motion model in (2.2) which has already been refined by the new proposal distribution with the incorporation of the previous observations. Furthermore, observations should always be incorporated in ascending order of range, since observations that lead to new landmarks typically occur at the leading edge of the sensor's field-of-view, adding them last ensures a better accuracy of the overall map.

FastSLAM 2.0 outperforms FastSLAM 1.0 in practically every aspect [30], either by converging, and at a faster rate, when the observation error reaches low values, or by requiring a smaller amount of particles maintaining the same level of accuracy. In the end, FastSLAM 2.0 proved the convergence of the algorithm, allowing to achieve good results for even one particle, yet its implementation can be quite overwhelming and its particle nature makes it difficult to tune and easy to diverge. That is why algorithms such as GraphSLAM, that exploit the intuitive graph nature of SLAM and apply optimization techniques to the SLAM problem emerged, allowing to create maps with more than 10^8 features [31]. A broader overview of this new generation of full SLAM techniques will be given in Section 2.4.

2.4 GraphSLAM

The SLAM algorithms presented up until now are based on filtering techniques, meaning that they model the SLAM problem as an online state estimation problem, being the state variables the current position of the agent and the map [13]. This estimate is then augmented and refined by incorporating new measurements as they become available. A key disadvantage of these filtering techniques is that data is processed and then discarded, making it impossible to revisit all data at the time of map building. Smoothing approaches, such as GraphSLAM, a state-of-the-art graph-based SLAM method proposed in [20], address the full SLAM problem, and seek to estimate the full trajectory of the robot from the full set of measurements along with the map.

The posterior of the full SLAM problem in (1.2) naturally forms a sparse graph, where nodes represent pose estimates or landmark locations with edges denoting observations connecting them, see Figure 2.5. This graph leads to a sum of non-linear quadratic constraints, that when linearized, form a least-squares problem that can be optimized using standard optimization techniques. Optimizing these constraints yields a maximum likelihood map and corresponding set of robot poses [31].

Once again, for the sake of this explanation, both observations and odometry estimates will be assumed to be only affected by local Gaussian noise and the data associations are known. This being said, one can rewrite the measurement model (2.1) and the

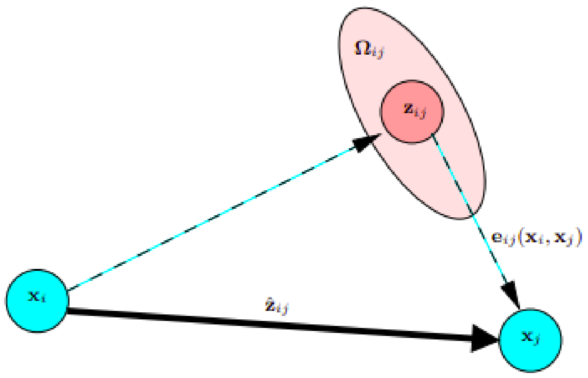


Figure 2.5: Example of an edge connecting the vertex x_i and vertex x_j. originated from the measurement z_{ij}. From the relative transformation between the two nodes is possible to compute the error or *residual* $\mathrm{e}_{ij}(x_i, x_j)$ that represents x_j seen from x_i. An edge is fully characterized by its error function $\mathrm{e}_{ij}(x_i, x_j)$ and by the information matrix Ω_{ij} that accounts for the measurement uncertainty. Figure obtained from [13].

motion model in (2.2) in a general way as:

$$p(z_i|s_{1:N}) = \eta_i \exp\left((-\mathrm{e}_\mathrm{i}(z_i, \hat{z}_i))^T \Omega_i \, \mathrm{e}_t(z_i, \hat{z}_i)\right) \ , \tag{2.21}$$

where $\hat{z}_i(s_{1:N})$ defines the expected measurement associated to the *i-th* observation or odometry measurement given the set of poses $s_{1:N}$:

$$\hat{z}(s_n, s_{n+1}) = s_{n+1} \ominus s_n \ , \tag{2.22}$$

$\mathrm{e}_i(z_i, \hat{z}_i)$ is the *residual* for measurement j defined by:

$$\mathrm{e}(z, \hat{z}) = z \ominus \hat{z} = z \ominus (s_{n+1} \ominus s_n). \tag{2.23}$$

Ω_i is the *information matrix* defined by the inverse of the covariance of either the measurement noise ϵ_t or the motion noise δ_t and η_i is a normalization constant. The operand $\ominus$ represents the inverse pose composition, *i.e.* the inverse of the transformation between s_n and s_{n+1}.

The goal of this graph-based algorithm is to compute a Gaussian approximation of the posterior (1.2), which involves computing the mean of this distribution as the configuration of nodes that maximizes the likelihood of the observations $\mathcal{Z} = \{z_i\}$ as in:

$$\arg\max p(s_{1:N}|\mathcal{Z}). \tag{2.24}$$

Knowing that $p(\mathcal{Z})$ is an unknown constant and that $p(s_{1:N})$ is uniformly distributed [32], one can rewrite (2.24) using the Bayes' rule as:

$$p(s_{1:N}|\mathcal{Z}) = \frac{p(\mathcal{Z}|s_{1:N})\, p(s_{1:N})}{p(\mathcal{Z})} \propto p(\mathcal{Z}|s_{1:N}). \tag{2.25}$$

From (2.21) and (2.25) the GraphSLAM optimization problem can be simplified into:

$$\arg\max p(s_{1:N}|\mathcal{Z}) = \arg\min \sum_{i=1}^{M} (\mathrm{e}_i(z_i,\hat{z}_i))^T \Omega_i \mathrm{e}_i(z_i,\hat{z}_i)\,, \tag{2.26}$$

meaning that the distribution that we seek to minimize is defined by:

$$x^* = \arg\min_x \mathrm{F}(x), \text{ where} \tag{2.27}$$

$$F(x) = \sum_{i=1}^{M} (\mathrm{e}_i(z_i,\hat{z}_i))^T \Omega_i \mathrm{e}_i(z_i,\hat{z}_i). \tag{2.28}$$

If a good initial guess $\breve{x}$ of the parameters is known, a numerical solution of (2.27) can be obtained from standard optimization techniques such as the Gauss-Newton [33] or the Levenberg-Marquardt [34] algorithms, through an approximation of the error function by its first order Taylor expansion around the current initial guess $\breve{x}$ according to:

$$\mathrm{e}_i(\breve{x}_i + \Delta x_i) \simeq \mathrm{e}_i(x) + \mathrm{J}_i \Delta x\,, \tag{2.29}$$

where J_i is the Jacobian of $\mathrm{e}_i(x)$ computed in $\breve{x}$. By substituting (2.29) in the error terms of (2.28) one obtains:

$$\begin{aligned} \mathrm{F}_k(\breve{x} + \Delta x) &\simeq (\mathrm{e}_i + \mathrm{J}_i\Delta x)^T \Omega_i (\mathrm{e}_i + \mathrm{J}_i \Delta x) \\ &= \underbrace{\mathrm{e}_i^T \Omega_i \mathrm{e}_i}_{c_k} + 2\underbrace{\mathrm{e}_i^T \Omega_i \mathrm{J}_i}_{b_k} \Delta x + \Delta x^T \underbrace{\mathrm{J}_i^T \Omega_i \mathrm{J}_i}_{H_k} \Delta x \\ &= c_k + 2b_k \Delta x + \Delta x^T H_k \Delta x. \end{aligned} \tag{2.30}$$

With this local approximation, Eq. 2.28 can be rewritten in a quadratic form by setting $c = \sum c_k$, $b = \sum b_k$ and $H = \sum H_k$ as in:

$$\mathrm{F}(\breve{x} + \Delta x) = c + 2b^T \Delta x + \Delta x^T H \Delta x. \tag{2.31}$$

The quadratic form obtained in (2.31) can then be minimized in Δx by solving the linear system:

$$H\Delta x^* = -b\,, \tag{2.32}$$

where the matrix H corresponds to the information matrix of the system, being sparse by nature, meaning that the only non-zero blocks will be the ones connected by a constraint, resulting in its number of non-zero blocks being twice the number of constraints plus the number of nodes in the graph.

The linearized solution is then obtained by adding to the initial guess the computed increments in (2.32) as in:

$$x^* = \breve{x} + \Delta x^*. \tag{2.33}$$

The Gauss-Newton (GN) [33] algorithm iterates through the linearization in (2.31), the solution in (2.32) and the update step in (2.33), where in every iteration, the previous solution is used as the linearization point and as *initial guess*.

The Levenberg-Marquardt (LM) [34] algorithm is a non-linear variant of the Gauss-Newton that introduces a damping factor and backup actions in order to control and guarantee the convergence. Instead of solving (2.32) directly, this method solves a damped version of it according to:

$$(H + \lambda I)\Delta x^* = -b\,, \tag{2.34}$$

where λ is the damping factor, responsible for controlling the step size in case of non-linear surfaces (the larger λ is the smaller are the Δx). The main advantage of this algorithm is to dynamically control the damping factor by monitoring the error of the new configuration after each iteration. If the new error is smaller than the one in the previous iteration, then the λ is decreased for the next iteration, increasing the rate of convergence. If, on the other hand, the error is larger than in the previous iteration, meaning that the optimum solution was surpassed, then the solution is reverted (backup action) and the λ increased.

GraphSLAM allows to solve the full SLAM problem by mapping the data collected into a sparse graph. This graph is then converted into an information form representation using linearization through the first-order Taylor expansion. This approximation can then be optimized through standard techniques by removing the mapping variables from the optimization problem resorting to exact transformations.

The main advantage of these kind of algorithms is that it allows to accumulate all data during the mapping process and only resolve it after the robot operation is complete. Furthermore, these optimization techniques allow for an abstraction in the SLAM problem by splitting the problem into a back-end problem and a front-end one. Most optimization techniques seek to compute the best map given the constraints (SLAM back-ends) and they typically rely on efficient implementations of common optimization algorithms, such as sparse Cholesky factorization [35] or Preconditioned Conjugate Gradients [36] (PCG). In contrast to that, SLAM front-ends seek to interpret the sensor data to determine the most likely constraint resulting from an observation to obtain the group of constraints that are the basis of optimization approaches. This problem is formally known as the *data association* and will be formally tackled in the next section.

2.5 Data Association

As introduced in Section 1.3.2 in many real-world applications where SLAM techniques are employed, landmarks are not identifiable and the total number of landmarks cannot be known *a priori*. Therefore it is required some kind of process, normally of probabilistic nature, that associates the observations into landmarks in the map in case they already exist, or create a new one when the observation does not match any of the already existing ones. This process is usually referred to as the *data association* and is widely considered in the literature as one of the most difficult problems in SLAM or localization. Successful data association involves associating the correct measurement with the correct state, initializing new tracks and detecting and rejecting spurious measurements [14]. In this Section a brief overview of three of the most broadly used data association methods will be given.

2.5.1 Maximum Likelihood and Individual Compatibility

The Maximum Likelihood or Nearest Neighbor (ML) [37] is one most basic and simplest methods for performing data association. The ML is obtained by calculating the likelihood of each landmark in the map against the individual measurement being considered. This can be done for any probability distribution as long as the probability density can be calculated for any possible measurement. This statement holds, since we consider the measurements to be normally distributed and only affected by Gaussian noise. The ML is obtained according to:

$$n_t = \arg\max_{n_t} p(z_t|s_t, \theta_{n_t}, n_t), \quad \text{with} \tag{2.35}$$

$$p(z_t|s_t, \theta_{n_t}, nt) = \frac{1}{(2\pi)^{n/2}\sqrt{|Q_t|}} \exp\left(-\frac{1}{2}(z_t - \hat{z}_t)^T Q_t^{-1}(z_t - \hat{z}_t)\right), \tag{2.36}$$

where n_t represents the identity of the landmark θ_{n_t}, z_t is the current observation, $\hat{z}_t$ the expected measurement if the landmark θ_{n_t} was observed at the current position s_t and Q_t is the innovation covariance matrix as defined in (2.10).

This method can be extended to include the ability to either reject spurious measurements or allow for the possibility of taking measurements of a previously unknown landmark by using *gate validation* techniques, such as Individual Compatibility (IC). This method, by itself, is not a method through which data associations can be chosen. Instead, it takes out of consideration data associations that are statistically improvable or incompatible. That is why this method is referred to in the literature as an *ambiguity reduction* method as opposed to the ML which is an *ambiguity management* one.

The gate validation is accomplished by noting that the exponent of the likelihood function in (2.36), describing the distribution of innovations, is χ^2 distributed. This is an obvious conclusion considering that $z_t - \hat{z}_t$ naturally forms a Gaussian distribution. These techniques accept the observations that are inside a fixed region of a χ^2 distribution, and reject the observations that make the innovation fall outside these bounds.

The IC test, otherwise known as the Mahalanobis Distance [38] or the Normalised Innovation Square (NIS) is then given by:

$$\frac{1}{2}(z_t - \hat{z}_t)^T Q_t^{-1}(z_t - \hat{z}_t) < \gamma_n. \tag{2.37}$$

In (2.37) the quantity γ_n depends on the dimension n of the innovation vector and on a fixed threshold that is determined by fixing the region of acceptance of the χ^2 distribution (*e.g.* 95%).

These two methods are normally used in conjunction due to their simplicity and computational efficiency ($O(mn)$). However, they have the drawback of failing to recover the true data association when the validation gates overlap and the observations fall within this overlapped region, inducing catastrophic failures in the SLAM problem [18]. One way to overcome this limitation is processing the observations jointly, such as the methods that will be described next, leading to a more robust data association.

2.5.2 Sequential Compatibility Nearest Neighbor

A simple way to assure that the resulting data association contains jointly compatible pairs of observations and landmarks is to use the Sequential Compatibility Nearest Neighbor method (SCNN) [18]. This method still uses ML and IC, however it implements a mutual exclusion property for batch data association.

The algorithm is implemented as follows:

Algorithm 1 Sequential Compatibility Nearest Neighbor Algorithm

```
1: Exclude from the batch all observations that fail the IC test in (2.37);
2: Calculate the ML for the remaining observations according to (2.36);
3: Choose the data association that maximizes the likelihood function (2.35);
4: Exclude from future consideration the data association pair (observation and land-
   mark) chosen in 3;
5: Repeat the process starting from 2 until the batch of observation is empty, i.e, until
   the maximal hypothesis set of data association is reached;
```

This methodology, although it allows for processing observation batches while guaranteeing the consistency with all pairings, has three major downsides. The first and more notorious is its computational complexity, requiring $O(mn^2)$. The second one is the fact that the algorithm never reconsiders the decision of pairing the observation with the most compatible feature. It ignores the fact that these pairs may be incorrect anyway, due to spurious or unrelated measurements. Finally, the third one concerns the fact that this implementation cannot maintain a notion of optimality over the entire batch of data associations. Since measurements are processed one at a time and therefore the data association choices are based only on the compatibility of a given measurement and the greedy mutual exclusion requirement [15].

2.5.3 Joint Compatibility Branch and Bound

Reconsideration of the established pairs is a necessary condition to limit the possibility of accepting spurious pairings, enforcing that all pairs belonging to the resulting hypothesis are jointly compatible. Contrarily to the SCNN, which is a naive application of the individual measurement data association to the batch of observations, Joint Compatibility (JC) is able to maintain a concept of joint optimality across the entire data association set.

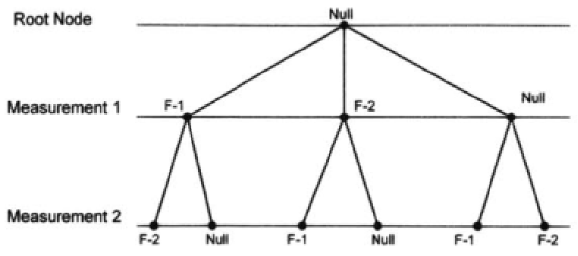


Figure 2.6: Example of tree traverse in the Joint Compatibility Branch and Bound data association method for a two observation-two landmarks scenario. Figure obtained from [15].

JC requires a search algorithm to traverse the interpretation tree in search of hypotheses that include the largest number of jointly compatible pairings. The reason being the fact that the probability that spurious measurements are jointly compatible with all the pairings of a given hypothesis decreases as the number of pairings in the hypothesis increases.

This method works in the following way:

Algorithm 2 Joint Compatibility Branch and Bound Algorithm

```
1: From all possible data association pairings, eliminate the ones that do not satisfy
   the IC test according to (2.37);
2: Verify if the current pairing satisfies the joint compatibility test $(z_t - \hat{z}_t)$;
3: If the joint compatibility test is passed, repeat 2. for the next pairing;
4: Case not, halt the search and prune the current path from future consideration.
   Move back up the tree, expand and consider the next possible pairing and repeat 2.;
```

The idea behind JC is expanding the individual compatibility concept to a set of many measurements and their landmark assignments, traversing the interpretation tree in search of the hypothesis with the largest number of non-null jointly compatible pairings.

Traversing the tree involves an algorithm that, from the viable solutions, finds the optimal one while performing as few operations in the minimum time possible. *Branch and Bound* [39] solves this problem by *bounding* its search to only compatible pairings, using a depth-first search tree, where each level in the tree represents a measurement within the batch being considered and each node represents a pairing between a landmark and a measurement in that level (see Figure 2.6). The node may include null assignments meaning the possibility of a spurious measurement may have occurred.

JC resolves two of the pitfalls identified in SCNN at the expense of exponential computational time, in contrast with the linear with the number of features required by SCNN, making it difficult to use in real-time applications such as the one in study.

Chapter 3

System Integration

This section will cover the specifics of solving the SLAM problem in the context of the Formula Student competition from the inputs of the algorithm up to the novelties that were required to obtain satisfactory results

In the first part an overview of the prototype along with the sensors used will be discussed, followed by a brief presentation of both perception and state estimation pipelines that feed the SLAM algorithm. Next, the motion models in use will be covered.

Finally, the specifics of both SLAM methods in study will be addressed touching aspects such as spurious landmark rejection, loop closure detection, transition to localization phase, terminating with the proposed method for data association.

3.1 Vehicle Setup

The base vehicle in use is the ninth prototype developed by the team, the FST09e in Figure 3.1. This prototype incorporates a full carbon fiber monocoque and complete aerodynamic package empowered by four permanent magnet synchronous electric motors, one attached to each wheel, producing a total peak power of $120kW$ and capable of accelerating from standstill to 100 km/h in under 2.5 seconds. The motors are supplied by an $8kW/600V$ High Voltage (HV) battery.

The FST09e was the most successful and reliable prototype developed and built by the team so far, scoring in 2019 an amazing 9^{th} place out of 39 teams in the most prestigious and challenging competition in the world, the Formula Student Germany (FSG), held at the famous Hockenheimring.

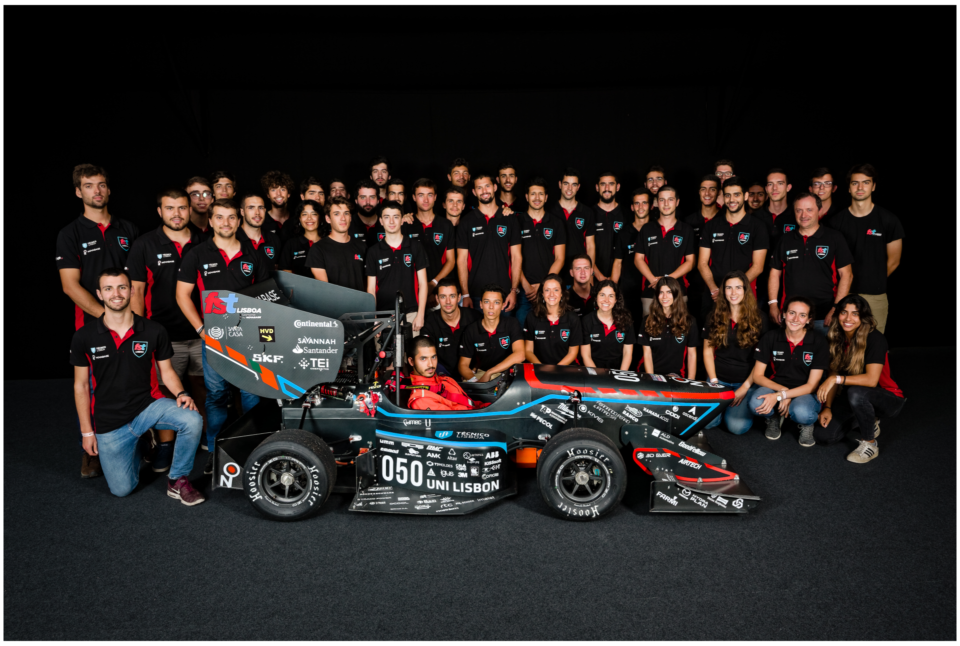
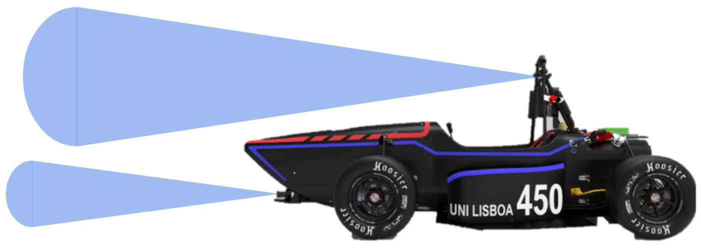


Figure 3.1: FST09e team and prototype in Formula Student Germany 2019.

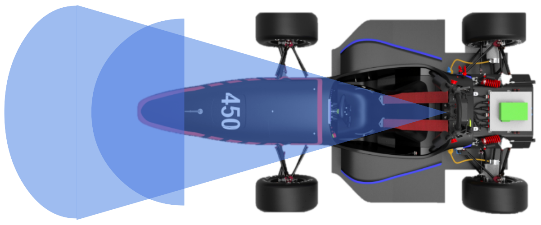
In order to meet the set of challenges posed by the Driverless competition, the prototype was equipped with a series of sensors (see Figure 3.2), in order to replace the driver's ability to perceive the environment. The first one being a *Light Detection and Ranging* (LiDAR) which calculates the distance to obstacles by targeting them with a laser beam and measuring the time for the reflected light to return to the receiver. More specifically, the LiDAR used is a Velodyne VLP-16, which is a 16 channel LiDAR that has a 100 meter range with 360 degrees of horizontal field-of-view and 30 degrees of vertical.

The second one, an RGB camera with a CMOS sensor, responsible for collecting rich information of the environment to complement the LiDAR, which collects sparse information in the form of a point cloud. The camera chosen is a Lucid Triton TRI032S with a resolution of 3.2 megapixels coupled to a Kowa 6 mm $f/1.8$ lens guaranteeing a horizontal FoV of approximately 65 degrees capturing 2048×1536 images at 30 FPS.

Additionally, the prototype was also fitted with an *Attitude and Heading Reference System* (AHRS) which estimates the accelerations and velocities of the vehicle by fusing *Inertial Measurement Unit* (IMU) and GPS data. The information retrieved by these sensors is then forwarded to the autonomous system algorithms which runs on a



(a) Side view



(b) Top View

Figure 3.2: Location and field-of-view of LiDAR and Camera.

powerful onboard computer equipped with an Intel Core i7-8700T CPU and an NVIDIA GeForce GTX 1060 Ti GPU.

The vehicle was also equipped with a DC motor with a gearbox attached to the steering column using a pulley system responsible for transmitting the steering demand requested by the controller pipeline to the wheels. Additionally, and since safety is a primary concern when dealing with any kind of racing vehicle, especially autonomous, the prototype was also fitted with a *Remote Emergency System* (RES), responsible for sending the *"Go Signal"* and, in case of emergency, stopping the vehicle at an average deceleration greater than $8\,\mathrm{m/s^2}$, by deploying the *Emergency Brake System* (EBS).

3.2 SLAM Inputs

The objective of the Autonomous Pipeline is to process the data retrieved from the various sensors and output a control command (steering + pedal) to guide the car along the track.

The Autonomous Pipeline, in Figure 3.3, is based in the ROS (*Robot Operating System*) framework [40] and is implemented in its majority in C++ programming lan-

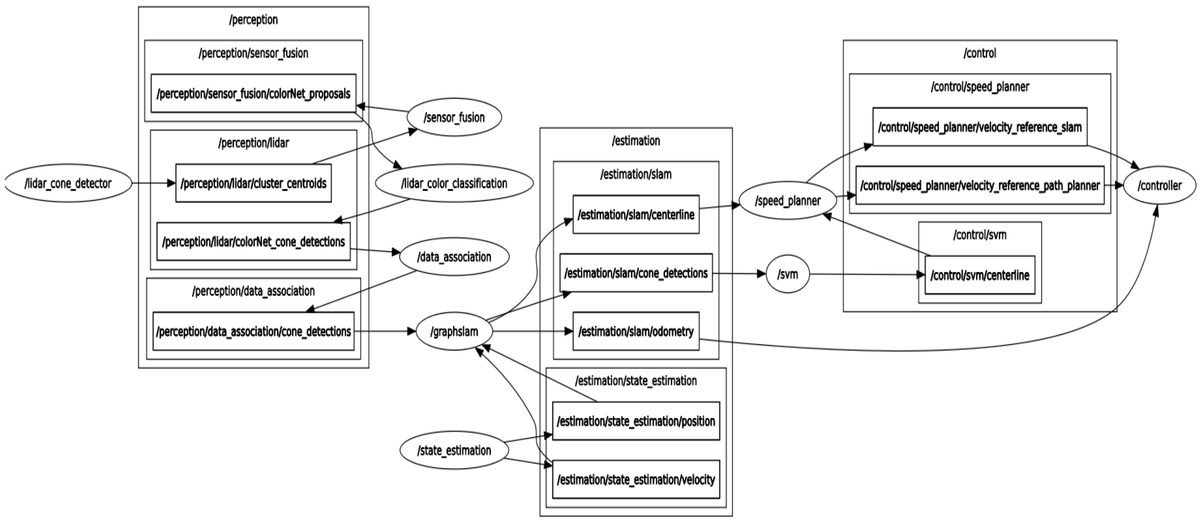

/perception
/perception/sensor_fusion
/perception/sensor_fusion/colorNet_proposals
/sensor_fusion
/perception/lidar
/lidar_cone_detector
/perception/lidar/cluster_centroids
/lidar_color_classification
/perception/lidar/colorNet_cone_detections
/data_association
/perception/data_association
/perception/data_association/cone_detections
/graphslam
/estimation
/estimation/slam
/estimation/slam/centerline
/estimation/slam/cone_detections
/estimation/slam/odometry
/estimation/state_estimation
/estimation/state_estimation/position
/estimation/state_estimation/velocity
/state_estimation
/speed_planner
/svm
/control
/control/speed_planner
/control/speed_planner/velocity_reference_slam
/control/speed_planner/velocity_reference_path_planner
/controller
/control/svm
/control/svm/centerline

Figure 3.3: Representation of the Autonomous System Pipeline node structure. The pipeline is divided into 3 modules (big rectangles): Perception, Estimation and Control, where each module has a number of nodes (oval boxes) responsible for specific tasks that communicate with each other by subscribing or publishing topics (small rectangles connected by arrows) containing useful information.

guage, with the exception being the algorithms that use Neural Networks, which are implemented using Python.

Parallel to this processing, a SLAM algorithm is responsible for mapping the track whistle providing a location of the vehicle. This algorithm receives as inputs the cone detections from the Perception Pipeline in Section 3.2.1, and velocities estimates computed by the State Estimation Pipeline in Section 3.2.2.

3.2.1 Perception Pipeline

The Perception pipeline is responsible for processing the LiDAR raw data and output the cone detections which are then used in both Path Planning and SLAM algorithms. The pipeline starts by processing the point cloud and applying a RANSAC (*Random Sample Consensus*) algorithm [41] to remove the ground plane. This can be done since the track surface is assumed to be near planar. After that, a Euclidean Clustering [42] algorithm is applied to the resulting point cloud which groups into clusters of points that respect a given predefined maximum distance threshold. These clusters are validated against the expected cluster that would be formed by a cone, whose dimensions are

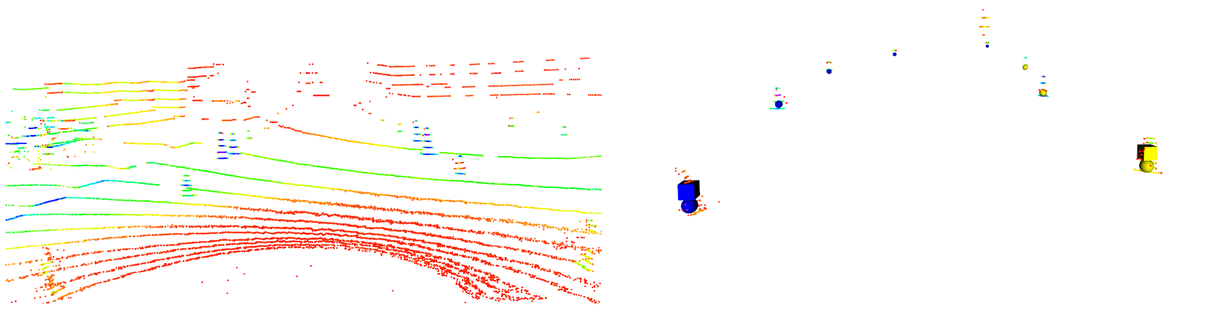

(a) Raw LiDAR point cloud (b) Processed point cloud and sensor fusion output

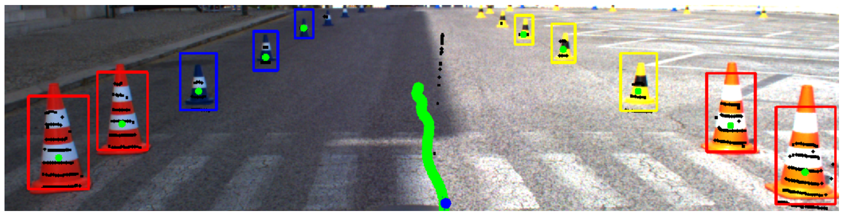
Figure 3.4: Difference between raw and after process point cloud. The sphere markers represent the output after the sensor fusion algorithm and the square markers indicate the color classification obtained through the LiDAR intensity projection.

known. This is done by fitting the resulting clusters to a cylinder with the expected dimensions of a cone. The clusters that passed this validation are reconstructed to recover points near the base that might have been removed by the ground removal algorithm. The before and after results of this processing can be seen in Figure 3.4.

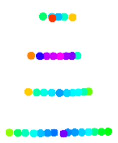
After the LiDAR processing, the cone candidates are projected into the image plane using the calibration matrices and a rectangular bounding box is constructed around the cluster in the image. This bounding box is then passed to a Neural Network which classifies the cone image candidate according to its color (blue, yellow, orange or unknown). The projection, bounding box and color classification can be seen in Figure 3.5(a).

For redundancy or in case of sensor failure, and since the field-of-view of the camera is limited when compared to the LiDAR, and considering each cone possesses a known color pattern, the cone detection candidates are also converted to small mono images. Each pixel value in the image corresponds to the intensity of the projected point reflection, which in theory reflects that same color pattern (lighter colors reflect more than darker ones), see Figures 3.5(b) and 3.5(c). These images are then fed into another Neural Network which classifies the color pattern and outputs the color estimate (blue, yellow or unknown).

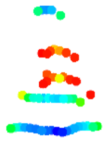
Finally, a sensor fusion algorithm is applied to eliminate duplicated estimates, *i.e.* ,



(a) Point cloud projection into image plane



(b) Blue cone color pattern



(c) Yellow cone pattern

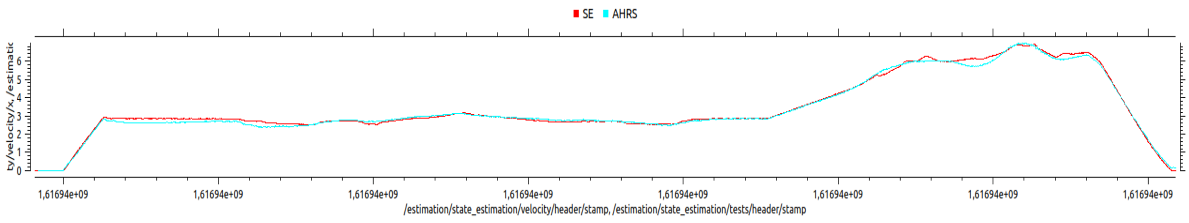
Figure 3.5: In (a) the point cloud projection along with the bounding box and resulting color classification can be seen. In (b) and (c) are shown the known color patterns. Blue cones are characterized by a *Low-High-Low* intensity pattern whereas yellow cones have *High-Low-High* intensity pattern.

if the color classification is only available on either the LiDAR or the camera, then the resulting cone defections would be the 3D position of the cluster along with the color classification from the available sensor, however, if the color classification is available from both sensors, then the camera one is chosen since is much more likely to be correct. The output of this algorithm is shown in Figure 3.4(b).

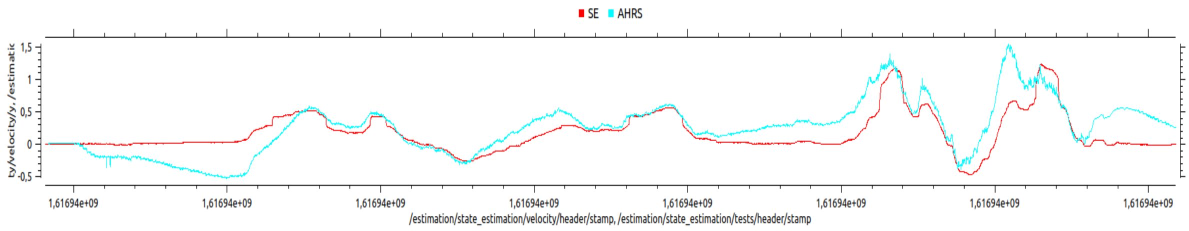
3.2.2 State Estimation Pipeline

The second input required to solve the SLAM problem is an odometry measurement, these measurements can be wheel encoders, IMU, ground speed sensors, steering angle sensors, among others. However, all these measurements have an associated error and are subject to noise. A better way is to combine the information from the available sensors in order to produce a more accurate odometry estimate. In our autonomous pipeline this is done through an EKF which combines information from the wheel speeds given by the wheel encoders, the steering angle and the accelerations measured by the AHRS.

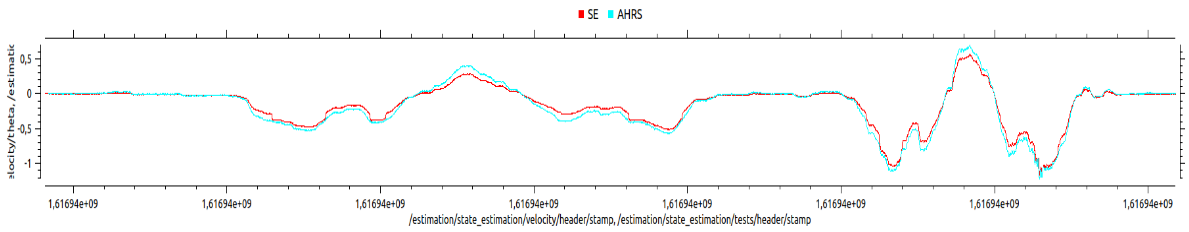
This EKF was presented in a master thesis project [43] developed within the team regarding a torque vectoring control application for vehicles with wheel motors. The



(a) v_x;



(b) v_y;



(c) Yaw Rate;

Figure 3.6: Comparison between AHRS only (red) and EKF (green) velocity estimates in terms of x, y and yaw rate components.

estimator makes use of the *LuGre Tire Model* [44], a dynamic model that attempts to describe the tire-road interactions from a physics point of view, being adequate for both low and high speeds. Moreover, instead of using a traditional motion model, such as the Bicycle Model, it uses a four-wheel model derived from the tire model, which is a much more accurate representation of the true car dynamics.

This implementation delivers better results (see Figure 3.6) than if a single sensor was used, not only because it combines information from the various available sensors, but also because it takes advantage of the car's characteristics and more importantly in a dynamic way. Besides that, by incorporating various sources of information the resulting observer is much more robust to unforeseen sensor failures by guaranteeing that the observability of the system is maintained.

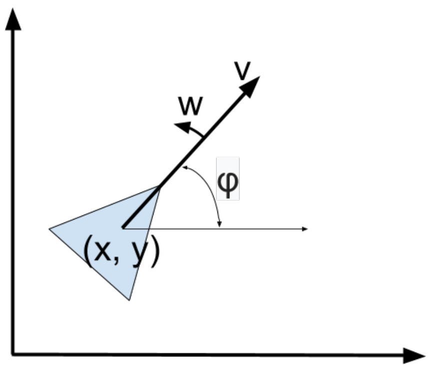


Figure 3.7: Representation of the Unicycle Kinematic Motion Model.

3.3 Motion Models

As described in the previous section, the state estimation pipeline feeds the SLAM algorithm with the velocities estimates. Although it uses a complex motion model to provide those estimates, they still need to be integrated in order to provide the position estimate that the SLAM problem requires. In order to accomplish that, SLAM algorithms normally rely on simpler representations of the car dynamics, such as the Unicycle Model, the Kinematic Bicycle Model [45], or if one wants to consider the tire-road interactions the Dynamic Bicycle Model [46].

The latter is discarded from consideration since the state estimation pipeline already uses a model that takes those interactions into account, meaning that the SLAM pipeline only requires a simple motion model to integrate those estimates. This being said, in this section will be presented the two motion models that were considered for the implementation of the proposed SLAM pipelines, the unicycle and the kinematic bicycle models.

3.3.1 Unicycle Model

The Unicycle Kinematic model presented in Figure 3.7 uses a unicycle vehicle model to simulate a simplified car-like vehicle dynamics. This model approximates a vehicle as a unicycle with a given wheel radius that can spin in place according to the velocity vector and angular velocity φ or yaw rate.

This model receives as inputs the velocity vector:

$$v = \sqrt{v_x^2 + v_y^2}\,, \tag{3.1}$$

and the yaw rate φ. The velocity vector according to the previous robot pose is then given by:

$$\begin{aligned} \dot{x} &= v\cos\theta_{t-1} \\ \dot{y} &= v\sin\theta_{t-1} \\ \dot{\theta} &= \varphi. \end{aligned} \tag{3.2}$$

The next robot position estimate is obtained by integrating the calculated velocities in (3.2) in each component over the time elapsed between measurements and adding them to the current vehicle position estimate, such as:

$$\begin{aligned} x_t &= x_{t-1} + \dot{x}\cdot\Delta_t \\ y_t &= y_{t-1} + \dot{y}\cdot\Delta_t \\ \theta_t &= \theta_{t-1} + \dot{\theta}\cdot\Delta_t. \end{aligned} \tag{3.3}$$

Although very simple and thus easy to implement, this model has a big drawback when it comes to the application in study. The unicycle model considers the vehicle by the center of its axis which in turn means that it implies that the vehicle turns relative to that point. This is not true in our prototype since the vehicle has two axles separate by a fixed wheel-base, being only one of them capable of steering, which means that the rotation is around a point somewhere close to the middle of the wheel-base, a point otherwise known as the *center-of-rotation*.

3.3.2 Kinematic Bicycle Model

The Kinematic Bicycle Model, in Figure 3.8 solves the pitfall posed by the Unicycle Model by combining the front and rear wheels of the four-wheel model into a two-wheel model, hence the name bicycle mode. By doing this approximation we are left with two wheels to deal with, instead of four and only one steering angle, instead of two.

For the simplicity of this explanation it is assumed that the robot has a lumped mass that acts at the *center-of-mass* or *center-of-gravity* (CG) and the no-slip condition, i.e., there is no lateral or longitudinal slip in the tire, meaning that we can consider that the velocity vector acts in the same direction as the wheels are pointing.

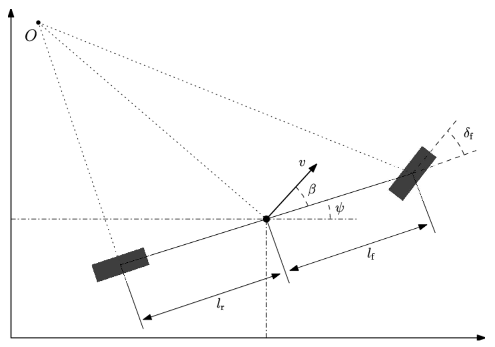


Figure 3.8: Representation of the Kinematic Bicycle Model.

Since the robot is represented by its CG, one can define l_f as the distance from the CG to the front axle and l_r as the distance from this point to the rear axle. Furthermore, the sum of these distances gives the total wheel-base L. The model receives as inputs the velocity vector:

$$v = \sqrt{v_x^2 + v_y^2}\,, \tag{3.4}$$

and the steering angle δ.

The speeds of the robot in the terms of x, y and angular components are given by:

$$\begin{aligned} \dot{x} &= \cos(\beta + \theta_{t-1}) \cdot v \\ \dot{y} &= \sin(\beta + \theta_{t-1}) \cdot v \\ \dot{\theta} &= \frac{\tan\delta \cdot \cos\beta}{L} \cdot v\ , \end{aligned} \tag{3.5}$$

where β represents the slip angle, more specifically the difference between the velocity vector and the current orientation of the vehicle and its defined by:

$$\beta = \arctan\frac{l_r \cdot \tan\delta}{L}. \tag{3.6}$$

The next position of the robot can then be calculated by once again integrating the speeds obtained in (3.5) over the time between measurements and adding the result to the current position, such as described in (3.3).

This model is much more suitable for our application since it considers not only the dimensions of the car but also takes as input the steering angle. Moreover, at the speeds we are running, the assumption regarding the no-slip condition and the consideration of the CG as the point of rotation are both guaranteed.

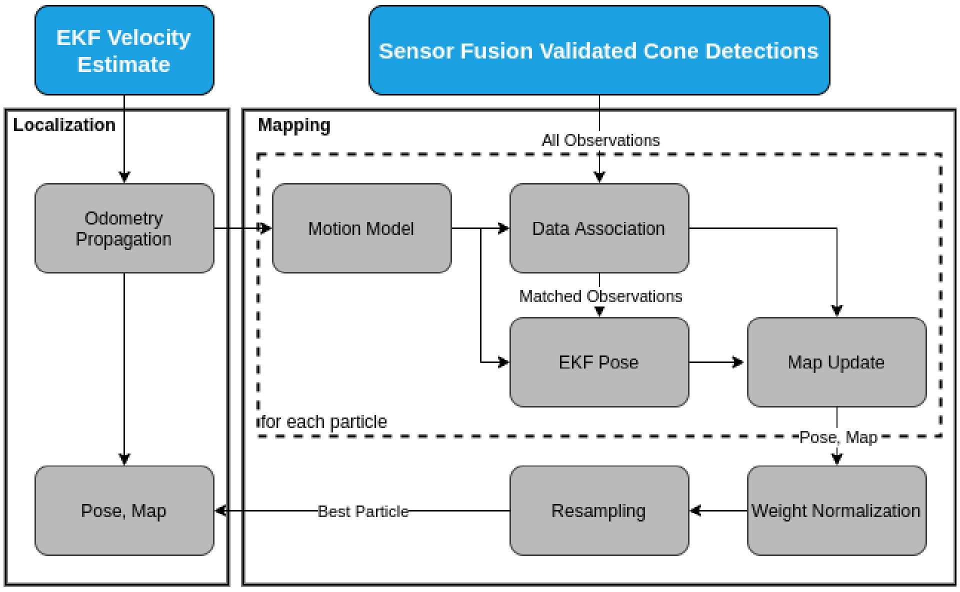


Figure 3.9: Detailed FastSLAM architecture used to fuse landmark observations from camera and LiDAR with velocity estimates into a coherent map and pose within the map. The dashed line visually demonstrates which parts of the algorithm are computed on a *per-particle* basis.

3.4 FastSLAM Implementation

Both FastSLAM 1.0 and 2.0 algorithms were implemented from scratch using the ROS framework and C++ programming language, according to the formulation in [9]. Although the general idea behind the algorithm was maintained, some changes were required in order for it to work in the Formula Student environment, mainly in the weight calculation, loop closure detection and transition to localization phase. This Section will cover the specifics of our FastSLAM implementations, discarding the differences between them but focusing on the common aspects of both algorithms.

As described in Section 3.2, the pipeline (see Figure 3.9) receives as inputs the cone detections and velocity estimates. Cone detections are only accepted provided they were observed a minimum number of times and whose color classification is known.

Every time a new set of landmark observations is received, the particle filter is updated. The observations are in the vehicle reference frame $z = [x_l, y_l]^T$ meaning that the origin is located at the current position of the vehicle $s = [x_v, y_v, \theta_v]^T$. The position of the landmarks within the local reference frame can be described in terms of Cartesian coordinates $\theta = [x_{j,l}, y_{j,l}]^T$ or the respective cylindrical coordinates $\theta = [r_j, \theta_j]^T$, which are the ones used by the measurement model in (2.1) or in terms of global map frame $\theta = [x_{j,g}, y_{j,g}]^T$. The transformations between these coordinates are given by:

$$\begin{pmatrix} x_{j,l} \\ y_{j,l} \end{pmatrix} = \begin{pmatrix} r_j \cos\theta_j \\ r_j \sin\theta_j \end{pmatrix} \tag{3.7}$$

$$\begin{pmatrix} r_j \\ \theta_j \end{pmatrix} = \begin{pmatrix} \sqrt{x_{j,l}^2 + y_{j,l}^2} \\ \arctan(y_{j,l}, x_{j,l}) \end{pmatrix} \tag{3.8}$$

$$\begin{pmatrix} r_j \\ \theta_j \end{pmatrix} = \begin{pmatrix} \sqrt{(x_{j,g} - x_v)^2 + (y_{j,g} - y_v)^2} \\ \arctan(y_{j,g} - y_v, x_{j,g} - x_v) - \theta_v \end{pmatrix}. \tag{3.9}$$

The update starts by propagating the particle poses using the motion model without noise, where the delay between the timestamp of the cone detections and the last pose estimate is compensated. After that the data association process takes place, where the observations are either mapped to existing landmarks in the map or generate new ones. This process will be fully discussed in Section 3.6. In the case of FastSLAM 2.0, the pose estimate is then enhanced using an EKF which is iteratively refined with the incorporation of the matched observations. This requires a linearization of the measurement model in (2.1) according to the first order Taylor expansion. The Jacobian of g with respect to the the pose s in (2.16) according to the motion model in use is defined by:

$$G_s = \begin{pmatrix} -\sqrt{\delta_x^2 + \delta_y^2} \cdot \delta_x & -\sqrt{\delta_x^2 + \delta_y^2} \cdot \delta_y & 0 \\ -\delta_y & \delta_x & -\left(\delta_x^2 + \delta_y^2\right) \end{pmatrix}, \tag{3.10}$$

where δ_x and δ_y are the x and y distances between the *i-th* landmark in the global map frame and the current pose estimate.

With the now known data associations for every particle, the EKF of each landmark can be updated using (2.9). Again, the measurement model g needs to be linearized being the Jacobian with respect to the landmark location θ in (2.8):

$$G_\theta = \begin{pmatrix} \sqrt{\delta_x^2 + \delta_y^2} \cdot \delta_x & \sqrt{\delta_x^2 + \delta_y^2} \cdot \delta_y \\ -\delta_y & \delta_x \end{pmatrix} \tag{3.11}$$

Lastly, the weight $w_t^{[m]}$ of each particle can be calculated, based on the likelihood of the data association process, the number of new landmarks and a penalty for every landmark that is in the *field-of-view* (FoV) but is not observed or whose mapped color does not agree with the observation [47], according to:

$$w_k = w_{k-1} l^v w_b^k w_c^\gamma \prod w_{k,n} \ , \tag{3.12}$$

where w_{k-1} is the particle weight in the previous update, l the weight assigned to new landmarks and v the number of new landmarks, w_b the penalty for landmarks that were in sensor range but were not observed and k the number of not observed landmarks, w_c penalizes color mismatches for all γ landmarks whose color does not match the associated observation and $w_{k,n}$ are the importance weights, in (2.6) or (2.20), of all matched landmarks.

Naturally, the particle weight variance increases over time and therefore resampling is enforced once the effective sample size N_{eff} of the particles drops below a certain threshold:

$$N_{\mathit{eff}}^{[k]} = \frac{1}{\sum \left(w^{[k]}\right)^2} . \tag{3.13}$$

To detect the loop closure a simple finite state machine is used [48], where each landmark possesses a loop closure state that can take one of the following states: *landmarkInView, landmarkLeftView, landmarkReturned.* When landmarks are created they are initialized in the *landmarkInView* state, and when they left the FOV they transition to the *landmarkLeftView* state. The *landmarkReturned* state is triggered when landmarks that were in the *landmarkLeftView* state return to the FOV with a heading not deviating more than a threshold from the initial observed heading.

This state machine is implemented along with a statistics function that keeps track, for each set of observations, of the number of landmarks that were observed or missed according to the expected FoV of the sensors. If a landmark is missed more times than the ones it was observed then it is deleted. The loop closure is detected when the number of returned landmarks is equal or greater than the number of seen plus missed landmarks multiplied by a percentage factor and the standard deviation of all particles drops below a fixed threshold.

After loop closure detection the SLAM algorithm switches to a localization phase using the map of the highest particle weight which is copied to the remaining particles in the particle set. The map is fixed by disabling the landmark EKF update and both track boundaries and centerline are computed. In order to localize the car, a smooth pose update is given by taking the weighted average over all the particles, essentially turning the SLAM algorithm into a Monte Carlo Localization problem.

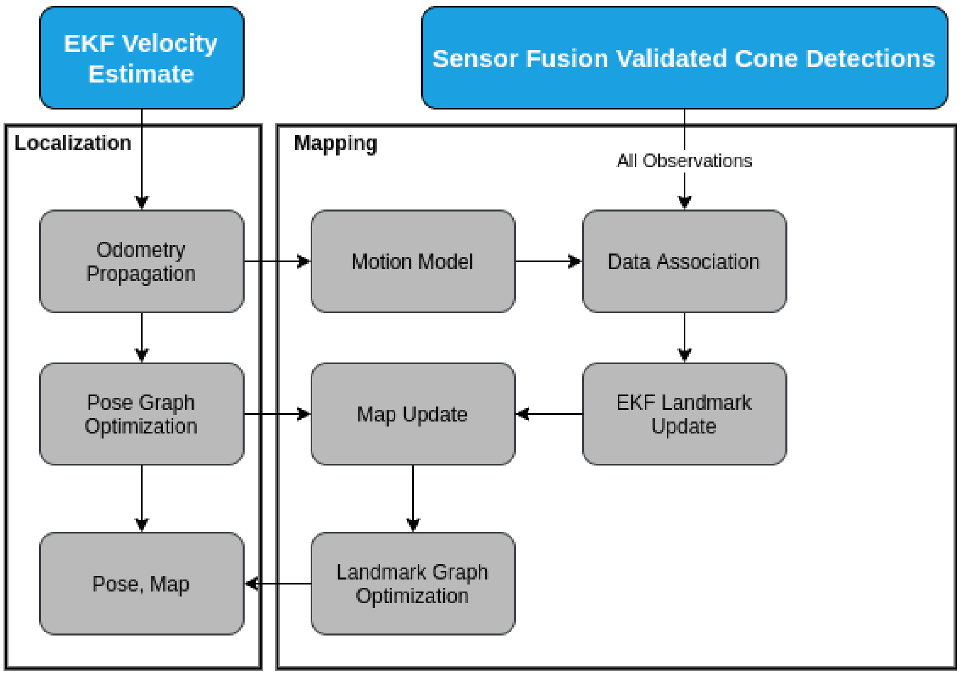


Figure 3.10: Detailed GraphSLAM architecture used to fuse landmark observations from camera and LiDAR with velocity estimates into a coherent map and pose within the map.

3.5 GraphSLAM Implementation

Similarly to the FastSLAM implementation, the GraphSLAM implementation, in Figure 3.10, receives as inputs the validated cone detections from the perception pipeline along with velocity estimates from the state estimation pipeline. Here some novelties to the original algorithm proposed in [13] were introduced by including an EKF for estimating the landmark locations [49] and by performing a multi-level optimization allowing this SLAM algorithm to be used exclusively for localization.

The algorithm is callback-based meaning that an update is performed every time a new set of cone detections is received, although based on the ROS framework which allows for nodes to be run at specific rates. For every new set of cone detections, a pose is sampled from the motion model, and a new odometry vertex is created along with an edge connecting the previous pose vertex to the new one. The error function in (2.23) is defined as the relative transformation between the two pose estimates s_a and

s_b, according to:

$$\begin{aligned} \mathrm{e}^s_{t,t+1}(s_t, s_{t+1}) &= z_{t,t+1} \ominus h^s_{t,t+1}(s_t, s_{t+1}) && (3.14)\\ h^s_{t,t+1}(s_t, s_{t+1}) &= s_a \ominus s_b \\ &= \begin{pmatrix} (x_a - x_b)\cos\theta_b + (y_a - y_b)\sin\theta_b) \\ -(x_a - x_b)\sin\theta_b + (y_a - y_b)\cos\theta_b) \\ \theta_b - \theta_a \end{pmatrix}. && (3.15) \end{aligned}$$

After sampling the pose, the data association takes place, observations that lead to the creation of new landmarks are directly added to the graph with the creation of the respective vertex according to (3.20) in the position in which they were observed along with an edge connecting the new landmark to the pose from which it was observed. Again, the error function is defined as the relative transformation between the landmark and pose vertexes, as in:

$$\begin{aligned} \mathrm{e}^\theta_{t,i}(s_t, \theta_i) &= z_{t,i} - h^\theta_{t,i}(s_t, \theta_i) && (3.16)\\ h^\theta_{t,i}(s_t, \theta_i) &= \begin{pmatrix} (x^s_t - x^\theta_i)\cos\theta^s_t + (y^s_t - y^\theta_i)\sin\theta^s_t \\ -(x^s_t - x^\theta_i)\sin\theta^s_t + (y^s_t - y^\theta_i)\cos\theta^s_t \end{pmatrix}. && (3.17) \end{aligned}$$

The sum of all constraints in (2.28) that we seek to minimize can then be rewritten has:

$$\begin{aligned} F(x) = x_0^T \Omega_o x_0 &+ \sum_t (x^s_t - h(u_t, x^s_{t-1}))^T P_t^{-1} (x^s_t - h(u_t, x^s_{t-1})) \\ &+ \sum_t \sum_i (z_{t,i} - g(\theta_t, x^s_t))^T R_t^{-1} (z_{t,i} - g(\theta_t, x^s_t)) , \end{aligned} \tag{3.18}$$

where $x_0^T \Omega_o x_0$ is the *anchoring constraint* fixing the problem to the global reference frame. This constraint is needed because all the other relative constraints have no information about the global reference frame, meaning that if an anchoring point is not establish, the cost function would be invariant to rigid-body transformations, resulting in the system of equations being undetermined. In Figure 3.11 is presented an example of the graph constructed along with its associated constraints.

The first difference to the original algorithm comes when the observations are mapped to already existing landmarks. In such cases, following the idea of the FastSLAM, this implementation also uses an EKF per landmark to iteratively refine the position estimate of the landmarks has they suffer multiple observations, according to (2.9). This

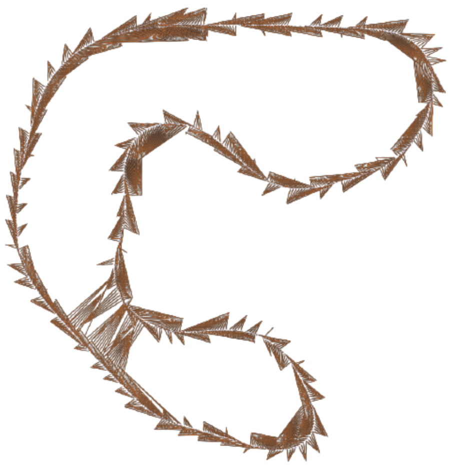


Figure 3.11: Example of the GraphSLAM sparse-graph representation. The edges connecting the nodes define the constraints with an associated error that will be minimized during the optimization process.

is particularly useful because otherwise, the optimization process would try to minimize the error according to the first position in which the landmark was observed, which is not necessarily the most accurate. Observations that lead to new landmarks typically tend to occur at the limit of the sensor range, where the uncertainty associated to the measurement is higher. These new landmarks become closer and are observed multiple times as the car navigates through them, meaning that would be illogical not to consider those observations, with smaller error, to improve the estimate. In such cases, the position of the landmark is updated in the graph following the standard EKF update equations and a new edge connecting the pose and the corresponding landmark is added to the graph. Once again the error function is defined as the relative transformation between the current pose and landmark position generated by the observation, as in (3.16).

The second particularity of the implementation is the multi-level optimization. A feature that is available since the back-end of this GraphSLAM application is based on the *g2o* library [32]. The idea is, just like in the FastSLAM approach, to split the SLAM problem into two, one global localization problem and N landmark mapping problems conditioned to the pose estimate. To do that the optimization is divided into two levels. The first handles the localization problem and the second the landmark location problem. After the update of the landmark locations, the local graph formed by the pose estimate and the observations is optimized. The optimized pose is then recovered and used to improve the estimate sampled from the motion model. When loop closure is detected, using the same statics function and state machine as in Section 3.4, the position of the landmarks in the second level is optimized. The optimized locations of the landmarks are then saved, the EKF update is disabled and their corresponding vertexes set fixed.

The algorithm transitions then to a localization phase where only the first level optimization, concerning the pose estimate, is performed. In this phase new observations do not lead to the creation of new landmarks, they are instead only assigned to already existing ones but do not improve their estimate since the second level graph is set fixed and the EKF update disabled.

3.6 Data Association Implementation

As described earlier the track is delimited by cones of identical size, only distinguishable by their color or color pattern, placed on similar looking asphalt. This invalidates the use of descriptors to aid the data association process.

The first algorithm used by the team to solve the SLAM problem was based on a Rao-Blackwellized particle filter that uses the maximum likelihood principle to solve the data association problem on a *per-particle* basis, which naturally allows for multi-hypothesis data association. This process, although reliable, can become computational expensive depending on the number of simultaneous observations because it involves for each observation loop through all the possible landmark assignments in the map searching for the one that best fits the observation. In order to overcome this drawback a data association method that combines the maximum likelihood principle with tracking information from the visible landmarks was implemented.

The idea behind the method is to avoid looping through all the landmarks calculating the value of the likelihood function. Instead, landmarks that possess tracking information indicating that they are unlikely to correspond to the observation are immediately discarded. Landmarks that do not possess tracking information are tested in terms of likelihood against the observations according to (2.35). It should be added that all observations that fail the individual compatibility test (2.37) are considered as spurious measurements and discarded.

The tracking part of the data association process is implemented at the level of the observations in the perception pipeline, running in parallel with the ML to save on computational time. The tracking is done in a *Iterative Closest Point* (ICP) manner, by comparing each new set of observations with the previous one using the Bhattacharyya distance [50]:

$$D_B(z_p, z_q) = \frac{1}{4}\ln\left(\frac{1}{4}\left(\frac{\sigma^2_{z_p}}{\sigma^2_{z_q}} + \frac{\sigma^2_{z_q}}{\sigma^2_{z_p}} + 2\right)\right) + \frac{1}{4}\left(\frac{(\mu_{z_p} - \mu_{z_q})^2}{\sigma^2_{z_p} + \sigma^2_{z_q}}\right) , \tag{3.19}$$

where μ and σ are the mean and covariance associated to the observations z_p and z_q, respectively.

The Bhattacharyya distance is preferred to the Mahalanobis distance in (2.37) because it explicitly incorporates the covariance of the observation. This is particularly useful since noise levels of the observations often exceed the minimum expected distance between neighboring cones [49], and because it solves the problem raised by the IC test when the validation gates overlap. In such cases, where observations have similar means but different standard deviations, the Mahalanobis distance would erratically, tend to zero, whereas the Bhattacharyya distance grows depending on the difference between the standard deviations.

If the observations are deemed to be the same by comparison of the result in (3.19) with a fixed threshold, they are saved with a unique index and passed to the SLAM pipeline as cone detections, where the EKF update takes place according to (2.9). On the other hand, in the SLAM pipeline when landmarks are created according to:

$$\begin{pmatrix} x_{j,g} \\ y_{j,g} \end{pmatrix} = \begin{pmatrix} x_v + r_j \cos\varphi_v + \theta_j \\ y_v + r_j \cos\varphi_v + \theta_j \end{pmatrix} , \tag{3.20}$$

using the ML method the index of the observation generating the landmark is saved in its data structure.

Later, when new observations are being evaluated during the data association process this index is compared to the one stored in the landmark data structure and in case of a match the observation is set to be of that specific landmark and the maximum likelihood process does not need to take place, see Figure 3.12.

The tracking algorithm also takes care of the color assignment estimate. It works by maintaining a counter for every matched observation of the number of times each color $C \in \{yellowCone, blueCone, orangeCone, unknown\}$ was observed, such as in Figure 3.12. The color with most observations is chosen and passed to the SLAM pipeline along with the observation.

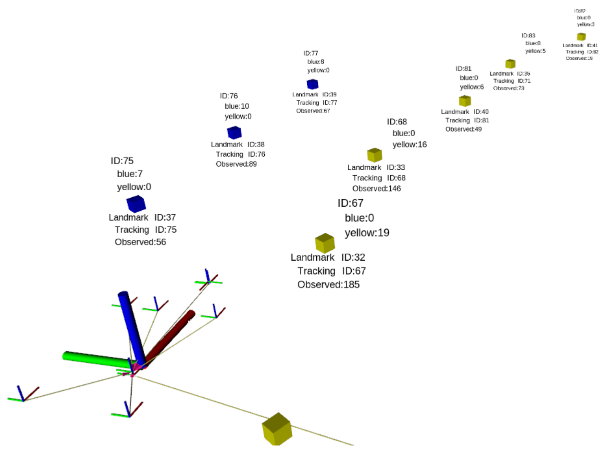


Figure 3.12: Example of the Data Association Process. Above each marker is possible to see the tracking ID of each observation and the number of observations for each color possibility. Bellow the marker is possible to see the SLAM information regarding the particular landmark, such as, the tracking ID with which it was created and the number of times it was observed.

Chapter 4

Results

This chapter will focus on the analysis in terms of accuracy and performance of both proposed pipelines (FastSLAM 2.0 and GraphSLAM) as well as the data association method in study. The initial implementation of the FastSLAM 1.0 will be compared against the new SLAM implementations in study, and the data association method will be compared against the individual methods presented in Section 2.5.

The pipelines were tested both using simulation and real data acquired during testing. The simulation results were obtained using an adapted version of the FSSIM [47], developed by the AMZ Driverless team from ETH Zurich. This simulator is based on Gazebo [51], however, it uses a first principle model instead of the Gazebo's included physics engine. This allows matching the simulation to the actual vehicle performance. Regarding the SLAM inputs, FSSIM does not simulate perception data due to the large computational requirements, instead, it generates direct cone detections affected by noise based on the expected error and FOV of the sensors. The same goes for the velocity estimates which are derived from the ground truth pose and disturbed with noise to better simulate real scenarios. Furthermore, a custom model of the real car was used for testing and the tracks generated from the ground truth data.

A key requirement for validating the accuracy of any SLAM application is having a ground truth map of the track. This is trivial in simulation however the same does not go for real scenarios. The ground truth of the tracks used for testing was obtained using a Total Station Theodolite in Figure 4.1(a) connected to a GNSS-RTK network. The device was stationed at the origin of the track and the distances and bearings to every cone in the track were measured with a sub-centimeter precision in terms of

(a) Total Station Theodolite

(b) Target

Figure 4.1: Device used to obtain the tracks ground truth. This device is connected to GNSS-RTK network for better accuracy and is capable of measuring distance and bearing to targets such as in (b) with millimetric/degree precision.

distance and millidegree angular precision. These measurements were then converted to Cartesian coordinates to facilitate the comparison to the maps from the SLAM pipelines and added to the simulator for initial testing and validation of the algorithms.

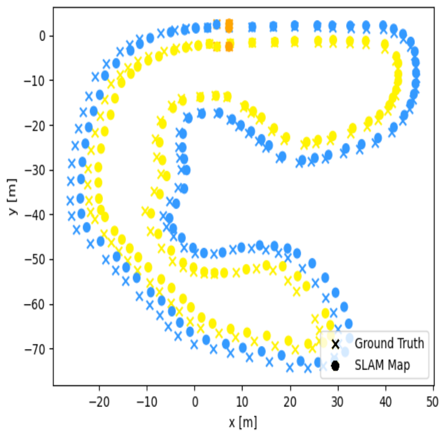
The parameters used for testing both FastSLAM and GraphSLAM implementations are presented in Table 4.1.

Table 4.1: SLAM Parameters

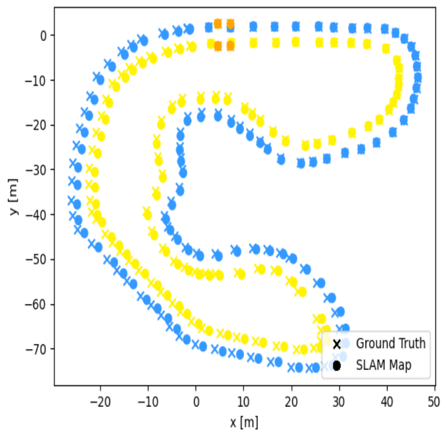
	FastSLAM	**GraphSLAM**
Number of Particles	50	N/A
New Landmark Threshold	0.02	0.02
Loop Closure Factor	0.8	0.8
$\mathbf{R_r}$ $\mathbf{R}_\theta$	0.4 m 4°	0.4 m 4°
$\mathbf{P_x}$ $\mathbf{P_y}$ $\mathbf{P}_\varphi$	0.5 m 0.5 m 5°	0.1 m 0.5 m 1°

4.1 Simulation Results

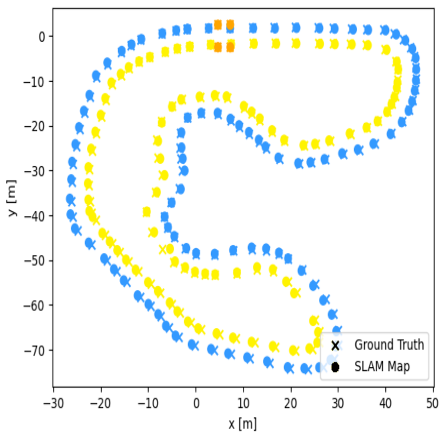
Before the algorithms could be tested in the real setup they were validated in simulation. These tests were conducted in order to evaluate the performance of the algorithms, but also for tuning of parameters and validation of the loop closure and localization features.



(a) FastSLAM 1.0



(b) FastSLAM 2.0



(c) GraphSLAM

Figure 4.2: Simulation results in the 2018 FSG track.

The simulation results for the three algorithms in study, showed in Figure 4.2, held an accuracy in terms of RMSE of 0.96 m, 0.38 m and 0.31 m for the FastSLAM 1.0, FastSLAM 2.0 and GraphSLAM algorithms, respectively.

In Figure 4.2, it is notorious the improvement introduced by the new sample distribution proposed in FastSLAM 2.0 when compared to FastSLAM 1.0. However, GraphSLAM holds the best result overall, benefiting from the fact of being a full SLAM technique. Furthermore, in terms of computational efficiency, the GraphSLAM implementation is considerably faster than the remaining filtering approaches, with an average computational time for one iteration of 0.00115 ms against the 0.0153 ms of FastSLAM 1.0 and 0.027 ms of FastSLAM 2.0. The lower performance in FastSLAM 2.0 is justifiable by the added complexity of sampling the pose based on the observations and processing observations that generate new landmarks last, instead of incrementally. It should be noticed that these higher values of computational time have no implication in the overall algorithm performance since cone observations are processed at a rate of 10 Hz $\approx$ 0.1 ms.

4.2 Real Scenario Results

With proven results in terms of synthetic data, the implementations were also tested regarding their mapping, localization and data association performance using the prototype autonomous capabilities in real competition scenarios.

4.2.1 Mapping Results

In these tests were evaluated the overall map accuracy as well as individual cone class accuracy in two different tracks. The maps were acquired at a constant speed of 3.5 m/s, the minimum imposed by the competition regulations [3]. The maps obtained are plotted against the ground truth in Figures 4.3, 4.4 and 4.5.

In Figures 4.3, 4.4 and 4.5 are evident the improvement from the first implementation of SLAM within the team (FastSLAM 1.0 in Fig.4.3) to FastSLAM 2.0 in Fig.4.4 and even more when compared to proposed GraphSLAM implementation in Fig.4.5.

In order to quantify this improvement, the *root mean squared error* (RMSE) per cone class, in Figures 4.6, 4.7 and 4.8, as a function of the travelled distance was calculated, according to:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left\|\hat{\theta}_i - \theta_i\right\|^2}. \tag{4.1}$$

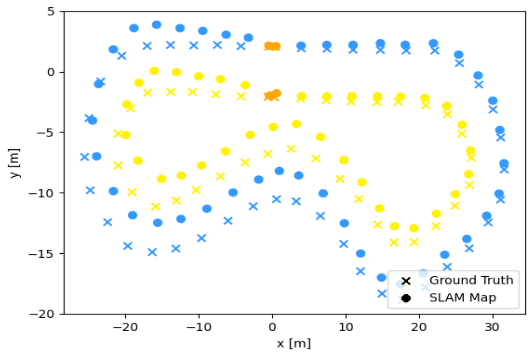
From the analysis of the Figures 4.6, 4.7 and 4.8 is notorious the trend evidenced in the simulation results in Section 4.1, the error decreases from FastSLAM 1.0 to 2.0 and even more in the case of GraphSLAM. Moreover, the propagation of the error due to the drift in the pose estimate is evident in the FastSLAM implementations, see Figures 4.6 and 4.7. In addition, in Figure 4.8 the role of the optimization in the GraphSLAM implementation is well demonstrated by reducing drastically the map overall error after loop closure detection from 1.23 m to 0.35 m for track 1 and from 1.37 m to 0.42 m in case of track 2.

The total RMSE of the maps obtained with each algorithm is presented in Table 4.2.

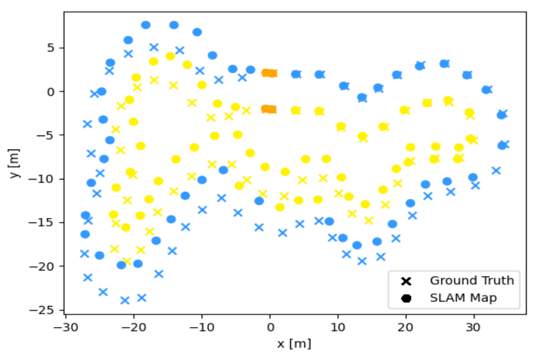
Table 4.2: Total RMSE of each implementation per track

	Total RMSE [m]		
	FastSLAM 1.0	**FastSLAM 2.0**	**GraphSLAM**
Track 1	1.44	0.96	0.35
Track 2	1.68	1.06	0.41

Through the analysis of Table 4.2 one can see that the error values confirm the values expected from the simulation results. Furthermore, the current GraphSLAM implementation represents an improvement in terms of mapping accuracy of 75% over the initial FastSLAM 1.0 implementation and 62% over FastSLAM 2.0, whistle being considerably more efficient and robust.

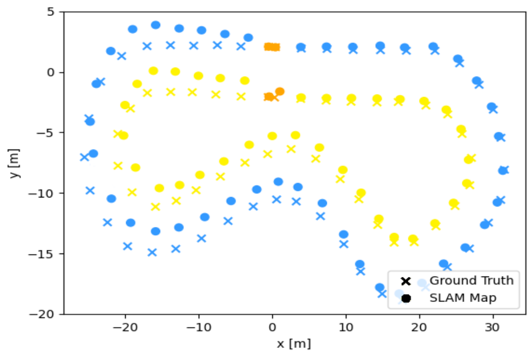


(a) Track 1

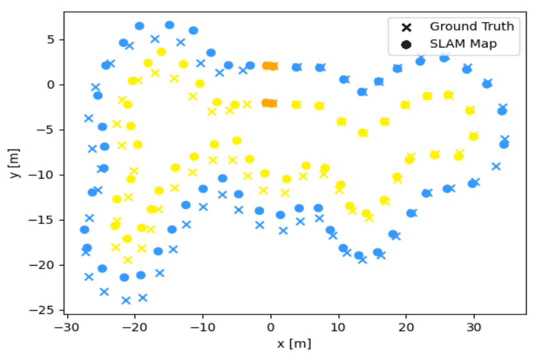


(b) Track 2

Figure 4.3: FastSLAM 1.0 maps against ground truth map

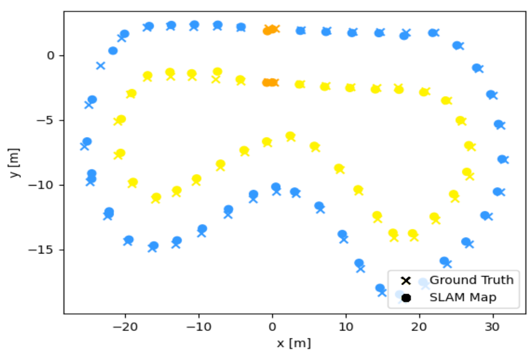


(a) Track 1

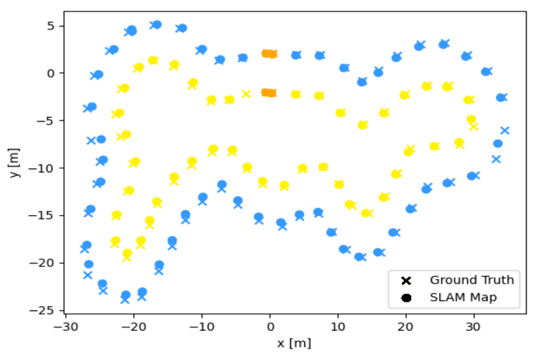


(b) Track 2

Figure 4.4: FastSLAM 2.0 maps against ground truth map

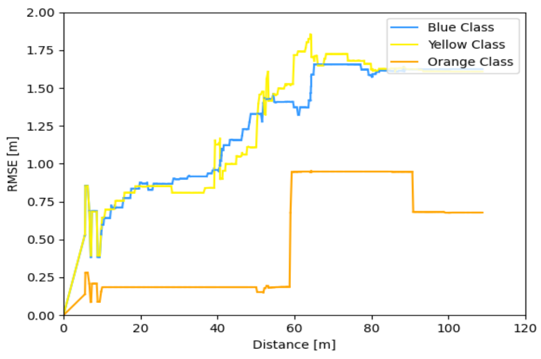


(a) Track 1

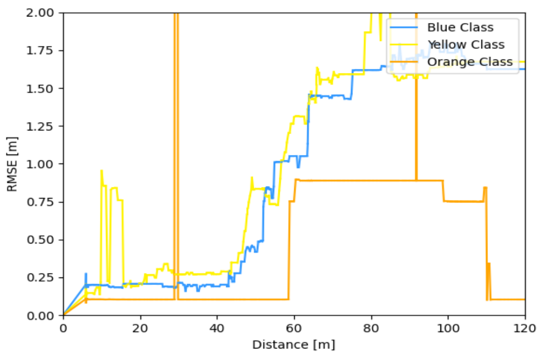


(b) Track 2

Figure 4.5: GraphSLAM maps against ground truth map

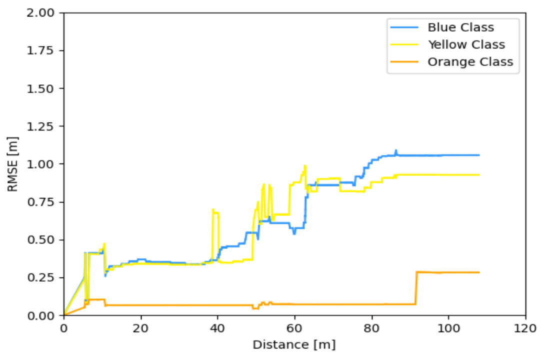


(a) Track 1

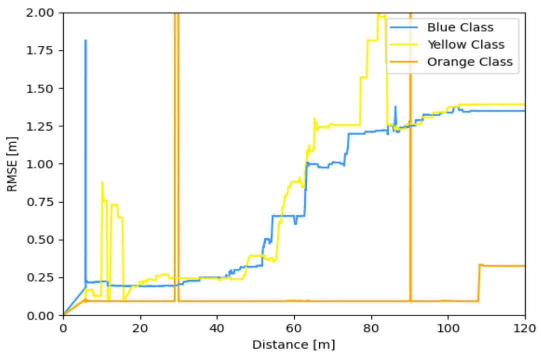


(b) Track 2

Figure 4.6: FastSLAM 1.0 map RMSE over distance.

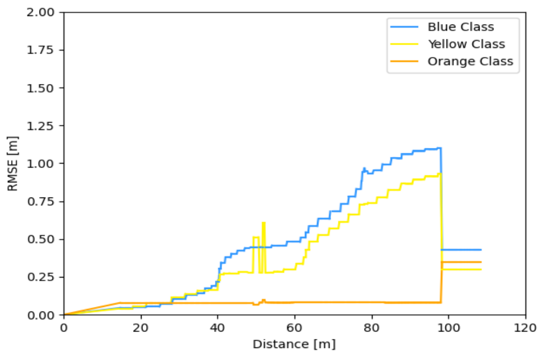


(a) Track 1

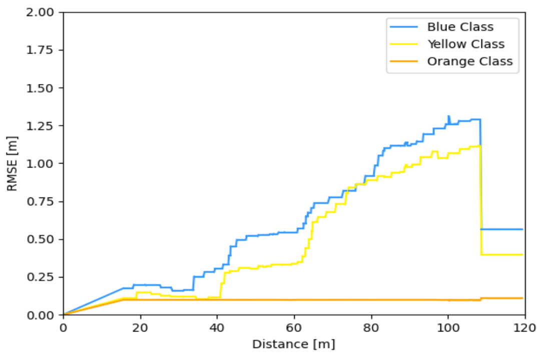


(b) Track 2

Figure 4.7: FastSLAM 2.0 map RMSE over distance.



(a) Track 1



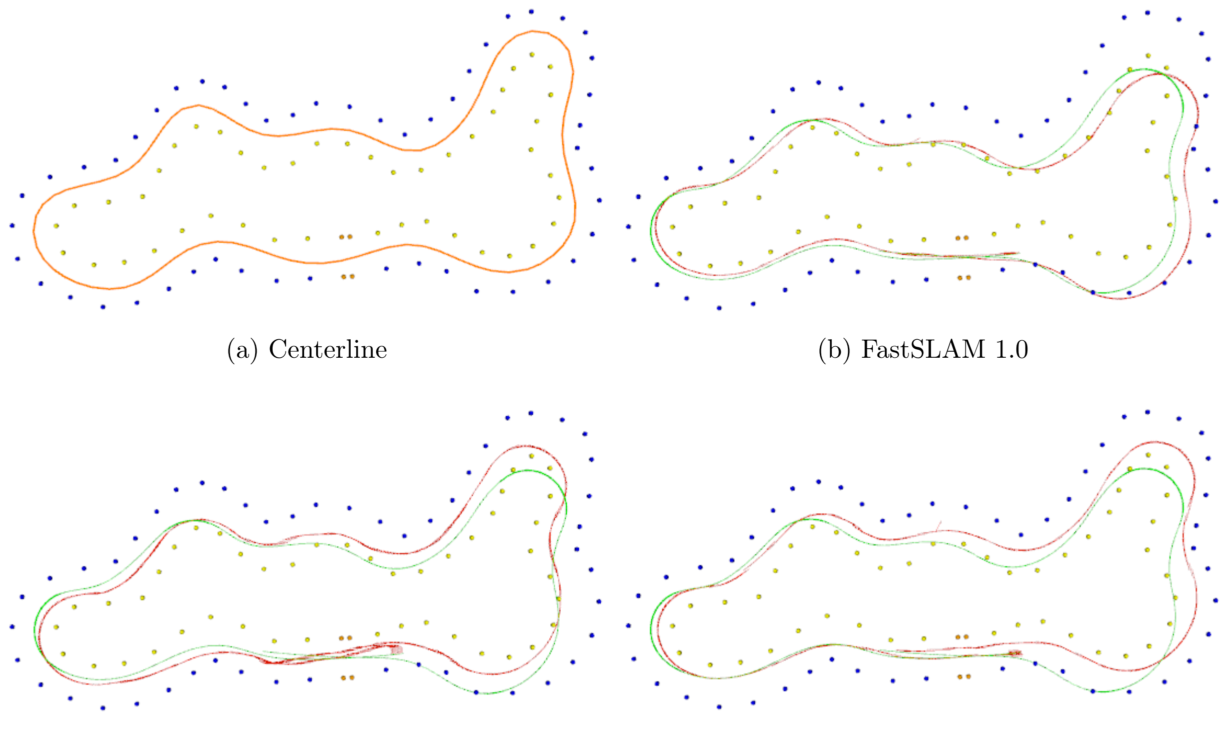
(b) Track 2

Figure 4.8: GraphSLAM map RMSE over distance.

4.2.2 Localization Results

In this section will be presented the results regarding the localization capabilities of the proposed pipelines. In the absence of a way to estimate the real position of the car along the track, the validation was done by overlapping the uncorrected and corrected trajectories over the ground truth map and by performing a qualitative analysis of the results.

The test was conducted using a previously attained map of the track, during one the autocross runs, and utilizing only the localization capabilities of the algorithms. In Figure 4.9 are shown the results in terms of localization for the most demanding track used in testing.


(a) Centerline

(b) FastSLAM 1.0

(c) FastSLAM 2.0

(d) GraphSLAM

Figure 4.9: Comparison of the localization results (in red) against the trajectory obtained from raw odometry data (in green). In (a) is represented, in orange, the path that the car is following.

From the analysis of Figure 4.9 several important conclusions can be taken. The first is the point from which the pose estimate obtained from the raw odometry measures, *i.e.* , without correction, starts to divert, caused by the error associated to the odometry measurements accumulated over time, causing a drift in the pose. The second is the improvement in FastSLAM 2.0 (Fig.4.9(c)) pose estimate over FastSLAM 1.0 (Fig.4.9(b)), thanks to the new sample distribution that takes into account the current observations to enhance the pose sampled from the motion model. The third and last conclusion is the superior results attained using the proposed method for localization using GraphSLAM in Figure 4.9(d). Albeit the results are not perfect, mainly due to the fact that GPS data was not available at the time of testing, the pose estimate using the proposed multi-level optimization is closer to the center of the track, which is the computed trajectory that the prototype tries to follow, in Figure 4.9(a).

4.2.3 Data Association Results

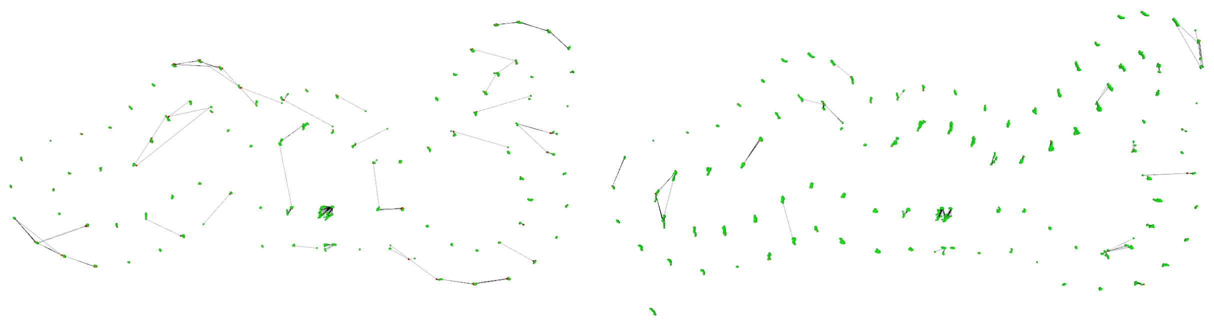
The data association accuracy of the proposed implementation in Section 3.6 was compared to the individual methods presented in Section 2.5 by mapping the observations in the global map frame connected to the associated landmarks.

The results of this test for the data association methods in study are present in Figure 4.10. Just by observing this figure, it is noticeable that the errors in the data association process are significantly reduced when using JCBB instead of the ML+IC method. On the other hand, when comparing the proposed data association method combining ML, IC and tracking information to JCBB, a smaller amount of data association errors is also noticeable. Putting the results into quantitative measures the ML method held an accuracy of 91%, the JCBB of 95% and the proposed method of 98%. The accuracy of the data association process was obtained by evaluating if the distance between the observation and the associated landmark was within the expected radius of a cone. A discrimination of these accuracy measurements is presented in Table 4.3.

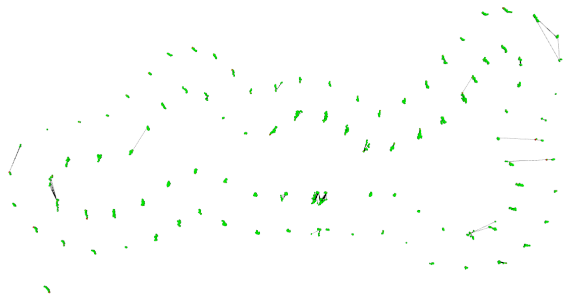
Table 4.3: Accuracy of the different data association methods

	ML + IC	**JCBB**	**Proposed Method**
Correct Data Associations	2029	2118	2186
Miss Data Associations	201	112	44

The data association errors are clearly evidenced in Figures 4.6, 4.7 and 4.8, corresponding to the spikes seen in the error plots of the different cone classes. The



(a) Maximum Likelihood + Individual Compatibility (b) Joint Compatibility Branch and Bound



(c) Maximum Likelihood + Tracking Information

Figure 4.10: Accuracy of the data association methods in real data. The green dots correspond to observations mapped into the world frame, red dots correspond to the landmarks in the map and the edges represent the mapping between observations and landmarks after the data association process.

GraphSLAM algorithm using the proposed data association method had a maximum data association error of 1.6 m, whereas FastSLAM 1.0 in the same conditions reached 1.76 m. The larger data association error in FastSLAM does not necessarily evidence an error in the data association algorithm itself, but instead, is a consequence of the correlation between the error in the pose estimate and the mapping of the landmarks.

In terms of computational time, as expected the JCBB was the slowest one due to the exponential complexity associated to the traversing of the tree. The best result was achieved by the ML + IC method, which is to be expected since the proposed method for data association also makes use of the ML + IC, but is then combined with the tracking information from the perception pipeline. The average computational achieved by using ML was 12.7 μs, 24.4 μs for the proposed method and 131 μs for the JCBB.

Chapter 5

Conclusions

At the beginning of this thesis, we set as objective providing FST with a SLAM algorithm that held consistent and accurate results. To accomplish that, the natural progression was transitioning from FastSLAM 1.0 to 2.0 which, even so, did not deliver satisfactory results, something that is evident in the mapping results presented in Section 4.2.1. Bearing in mind this conclusion, a full SLAM approach was implemented and tested which proven, not only, much more accurate but also much more reliable, robust and easy to tune, given our current setup.

Given all the constraints faced during the development and writing of this thesis, both for the team and companies that support our work and are the basis of our existence, the results accomplished are very satisfactory and fulfill the objectives set at the beginning of this thesis. More importantly, valuable lessons were learned, as well as much knowledge was acquired regarding SLAM applications in the Formula Student context, which will hopefully translate into even bigger improvements for future cars.

A complete autonomous pipeline was developed from scratch using only three basic sensors, one 16-layer LiDAR, one RGB camera and one IMU. This is nothing compared to the sensors setups that other teams are using, many times comprising two or three cameras and at least two LiDARs. Nevertheless, we are able to achieve good performances regarding mapping speeds and recover an accurate map that can be used in the subsequent laps. Building this prototype required much more than programming skills. A good simulation performance means nothing until it is translated to the real platform. The latter required mechanical and electrical knowledge far beyond the scope of this thesis, and even this course.

5.1 Future Work

Regarding future work, if we revisit the localization results in 4.2.2, it is clear that this is a weak point of our current pipeline. Future work should include testing of different locations of our current ARHS, combining it with another one for estimating y velocities more accurately or even incorporate the IMU available in the LiDAR.

Additionally, SLAM could provide much more information than a simple map for the next laps. For example, the detections from the perception pipeline are noisy and irregular, one way to explore the SLAM capabilities would be for the path planning algorithm to use the current data associations from the SLAM pipeline as cone detections. This would make the algorithm much more stable and robust to miss-classifications, not to mention that the cone detections from the SLAM pipeline are far more accurate since are being constantly improved by the EKFs. Another hypothesis would be to use the SLAM algorithm as a path planner. Given that our current implementation is based on Support Vector Machines (SVM), and that SLAM uses the same algorithm to find the center path of the computed map, it would be logical to incrementally generate this path and feed it to the control pipeline instead of generating the center line only after loop closure detection.

5.2 Final Remarks

As a final remark, I would like to express my gratitude to all my colleagues at FST Lisboa, especially to the Autonomous Systems department. This project and this competition provide an incredible opportunity to develop not only hard skills and gain experience, but also soft skills, such as teamwork. Everyday we challenged ourselves to solve problems that were many times out of our field of knowledge. Together we failed, learned from it and explored our limits individually and as a team. More support and acknowledgment from this institution would play a big role in what is, first and foremost, a student initiative with far-reaching potential for our future as engineers, individuals, and ultimately, the image of this learning institution.

Figure 5.1: Autonomous Systems department after the first fully autonomous ride.

Bibliography

[1] Javier Ibañez-Guzman, Christian Laugier, John-David Yoder, and Sebastian Thrun. Autonomous driving: Context and state-of-the-art, 2012. 1

[2] Johannes Betz, Alexander Wischnewski, Alexander Heilmeier, Felix Nobis, Tim Stahl, Leonhard Hermansdorfer, and Markus Lienkamp. A software architecture for an autonomous racecar. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–6. IEEE, 2019. 1

[3] Formula Student Germany. FS Rules 2020, 2019. 2, 4, 58

[4] Formula Student Germany. FSG Competition Handbook 2021, 2021. 4, 5

[5] Lisboa, FST. Autonomous Design Report - FST10d, 2020. 5, 6

[6] Michael Montemerlo and Sebastian Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 1985–1991. IEEE, 2003. 6, 15, 16, 17

[7] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017. 6

[8] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015. 6

[9] Michael Montemerlo and S Thrun FastSLAM. *A Scalabale Method for the simultaneous localizaton and mapping problem in robotics*, volume 27. Springer, 2007. 6, 14, 16, 19, 22, 24, 45

[10] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002. 7, 18, 19

[11] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986. 7, 14, 15

[12] Sebastian Thrun. *Probabilistic robotics*, volume 45. ACM New York, NY, USA, 2002. 7, 8, 9

[13] 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. 8, 9, 14, 25, 26, 48

[14] 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. 9, 10, 29

[15] Aron Jace Cooper. *A comparison of data association techniques for Simultaneous Localization and Mapping*. PhD thesis, Massachusetts Institute of Technology, 2005. 9, 10, 17, 31, 32

[16] Wu Zhou, E Shiju, Zhenxin Cao, and Ying Dong. Review of slam data association study. In *2016 International Conference on Sensor Network and Computer Engineering*. Atlantis Press, 2016. 10

[17] Michael Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem With Unknown Data Association.* PhD thesis, Carnegie Mellon University, 2003. 10, 14, 19, 22, 23

[18] 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. 10, 30, 31

[19] 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. 13, 15, 16, 18

[20] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997. 13, 25

[21] Philippe Moutarlier and Raja Chatila. An experimental system for incremental environment modelling by an autonomous mobile robot. In *Experimental Robotics I*, pages 327–346. Springer, 1990. 14

[22] Yaakov Bar-Shalom, Peter K Willett, and Xin Tian. *Tracking and data fusion*, volume 11. YBS publishing Storrs, CT, USA:, 2011. 15, 21

[23] Jose E Guivant and Eduardo Mario Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *IEEE transactions on robotics and automation*, 17(3):242–257, 2001. 16

[24] John J Leonard and Hans Jacob S Feder. A computationally efficient method for large-scale concurrent mapping and localization. In *Robotics Research*, pages 169–176. Springer, 2000. 16

[25] Sebastian Thrun, Daphne Koller, Zoubin Ghahramani, Hugh Durrant-Whyte, and Andrew Y Ng. Simultaneous mapping and localization with sparse extended information filters: Theory and initial results. In *Algorithmic Foundations of Robotics V*, pages 363–380. Springer, 2004. 16

[26] Jun S Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044, 1998. 17

[27] Arnaud Doucet, Nando De Freitas, Kevin Murphy, and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. *arXiv preprint arXiv:1301.3853*, 2013. 17

[28] Kevin P Murphy. Bayesian map learning in dynamic environments. *Advances in Neural Information Processing Systems*, 12:1015–1021, 1999. 18

[29] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 2, pages 1322–1328. IEEE, 1999. 19

[30] 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 *International Joint Conference on Artificial Intelligence*, pages 1151–1156, 2003. 22, 25

[31] 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. 25

[32] Giorgio Grisetti, Rainer Kümmerle, Hauke Strasdat, and Kurt Konolige. g2o: A general framework for (hyper) graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, pages 9–13, 2011. 27, 50

[33] William H Press, H William, Saul A Teukolsky, William T Vetterling, A Saul, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007. 27, 28

[34] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978. 27, 28

[35] Yanqing Chen, Timothy A Davis, William W Hager, and Sivasankaran Rajamanickam. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software (TOMS)*, 35(3):1–14, 2008. 29

[36] Erik F Kaasschieter. Preconditioned conjugate gradients for solving singular systems. *Journal of Computational and Applied mathematics*, 24(1-2):265–275, 1988. 29

[37] Shalom Y Bar, TE Fortmann, et al. *Tracking and data association.* PhD thesis, Academic press Cambridge, 1988. 29

[38] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000. 30

[39] Tim Bailey. *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments.* PhD thesis, University of Sydney, 2002. 32

[40] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009. 37

[41] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010. 38

[42] Christian Darken and John Moody. Fast adaptive k-means clustering: some empirical results. In *1990 IJCNN international joint conference on neural networks*, pages 233–238. IEEE, 1990. 38

[43] Nuno Salgueiro. *Torque vectoring control of an electric vehicle with in-wheel motors.* PhD thesis, Instituto Superior Técnico, 2021. 40

[44] C Canudas De Wit, Hans Olsson, Karl Johan Astrom, and Pablo Lischinsky. A new model for control of systems with friction. *IEEE Transactions on automatic control*, 40(3):419–425, 1995. 41

[45] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE intelligent vehicles symposium (IV)*, pages 812–818. IEEE, 2017. 42

[46] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099. IEEE, 2015. 42

[47] 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. 47, 55

[48] 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. 47

[49] Leiv Andresen, Adrian Brandemuehl, Alex Honger, Benson Kuan, Niclas Vödisch, Hermann Blum, Victor Reijgwart, Lukas Bernreiter, Lukas Schaupp, Jen Jen Chung, et al. Accurate mapping and planning for autonomous racing. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4743–4749. IEEE, 2020. 48, 52

[50] Anil Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–109, 1943. 52

[51] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004. 55