A SLAM Method for the Formula Student Driverless Competition

Luís Afonso Nazaré Correia Lopes
Instituto Superior Técnico, Lisboa
luis.afonso.lopes@tecnico.ulisboa.pt

***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 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.**

***Index Terms*—SLAM, Data Association, Autonomous Driving, Formula Student**

I. 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]. 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. 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 and Formula Student present a unique opportunity to develop, test and validate new technologies under challenging conditions.

A. 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 [2] 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.

In the Driverless class the track boundaries are delimited by blue cones on the left and yellow cones on the right, small orange cones delimit stopping zones and big orange cones mark timekeeping zones. The shape, dimensions and color pattern of these cones must always be the same, and is regulated by the FSG rules [2]. 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.

B. 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.

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.

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 [3]. This problem is widely considered in the literature as one of the most difficult [4] and at the same time essential for truly autonomous vehicles.

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.

II. Theoretical Background

A. *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 [5].

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 [6]. 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 [7]. 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 II-B.

The most common approach to the SLAM problem uses an Extended Kalman Filter (EKF) [8] for estimating the posterior distribution over the map and robot's pose [7]. 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. This approach allows to accommodate the nature of error in the map, as stated in [9], 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 requires performing an operation on every entry of the covariance matrix [3]. Several workarounds of this quadratic complexity have been proposed [10]–[12], 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*, discussed in Section II-B. 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 [3]. Albeit other data associations methods are available, 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 [5].

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 [9], and will be discussed further in detail in Section II-C.

B. *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 [13].

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 [14].

The measurements uncertainty 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 creates ambiguity due to the fact that the agent is uncertain about its location (position and orientation). Finally, landmark uncertainty 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 this day, feature-based data association a difficult research problem [15], with several algorithms being proposed in the literature, such as Maximum Likelihood (ML) [16], Individual Compatibility (IC) [14], Sequential Compatibility Nearest Neighbor (SCNN) [13] and Joint Compatibility (JCBB) [17].

C. FastSLAM 1.0

Particle filters [18] 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 [14]. 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 [19].

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 [3].

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. 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 was discussed in Section II-B.

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:

$$p(z_t|s_t, \theta_{nt}, n_t) = g(\theta_{nt}, s_t) + \epsilon_t. \tag{1}$$

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:

$$p(s_t|u_t, s_{t-1}) = h(u_t, s_{t-1}) + \delta_t\,, \tag{2}$$

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.

FastSLAM [6] exploits a property of the SLAM problem pointed out in [20], 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 to be represented in a factored way [6]. 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 can be converted into the Bayes Filter equation by making use of the Bayes Rule:

$$\begin{aligned} p(s_t, m|z^t, u^t, n^t) &= \eta\, p(z_t|s_t, \theta_{nt}, n_t) \\ &\int p(s_t|s_{t-1}, u_t) p(s_{t-1}, m|z^{t-1}, u^{t-1}, n^{t-1})\, ds_{t-1}\,, \end{aligned} \tag{3}$$

where η is just a normalization constant.

The Bayes filter in (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 (3) in a factorized form:

$$p(s^t, m|z^t, u^t, n^t) = p(s^t|z^t, u^t, n^t) \prod p(\theta_n|s^t, z^t, n^t). \tag{4}$$

In (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 [16]. Following a typical Dynamic Bayes Network (DBN) terminology, from (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 [5] 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.

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.

D. FastSLAM 2.0

The FastSLAM algorithm proposed in II-C 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 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. 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 [5].

FastSLAM 2.0 incorporates into the proposal distribution not only the importance weights, but also the current observations in order to obtain a better posterior. 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) according to motion command u_t, not taking into account the measurement z_t. Instead, this measurement is only incorporated in the resampling process. 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, and subsequently disappearing 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 [16].

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:

$$s_t^{[m]} \sim p(s_t|s^{t-1,[m]}, u^t, z^t, n^t)\,, \tag{5}$$

which explicitly incorporates the most recent sensor measurement z_t, its 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 (5) 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 (6) to a linear function:

$$g\left(\theta_{n_t}, s_t\right) \approx \hat{z}_t^{[m]} + G_\theta \cdot \left(\theta_{n_t} - \mu_{n_t}^{[m]}\right) + G_s \cdot \left(s_t - \hat{s}_t^{[m]}\right), \tag{6}$$

where $\hat{z}_t^{[m]}$ represents the predicted measurement, $\hat{s}_t^{[m]}$ the predicted pose and $\hat{\theta}_n^{[m]}$ the predicted landmark location. The matrix G_θ is Jacobian of g in respect to θ, and G_s the Jacobian of g in respect to s.

According to this EKF approximation, one can rewrite the proposal distribution in (6) as a Gaussian with mean and covariance defined by:

$$\Sigma_{s_t}^{[m]} = \left[G_s^T Q_t^{[m]-1} G_s + P_t^{-1}\right]^{-1} \tag{7}$$

$$\mu_{s_t}^{[m]} = \Sigma_{s_t}^{[m]} G_s^T Q_t^{[m]-1} (z_t - \hat{z}_t^{[m]}) + \hat{s}_t^{[m]}\,, \tag{8}$$

respectively, and where $Q_t^{[m]}$ represents the innovation covariance matrix:

$$Q_t^{[m]} = R_t + G_\theta \Sigma_{n_t, t-1}^{[m]} G_\theta^T. \tag{9}$$

The new sample distribution in (5) is now parameterized has a Gaussian approximation by (7) and (8). 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:

$$s_t^{[m]} \sim \mathcal{N}(s_t;\, \mu_{s_t}^{[m]}, \Sigma_{s_t}^{[m]}). \tag{10}$$

The new proposal distribution has an 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.

E. 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 [21]. 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 [22], 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 naturally forms a sparse graph, where nodes represent pose estimates or landmark locations with edges denoting observations connecting them. 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 [23].

Since both observations and odometry estimates are assumed to be only affected by local Gaussian noise and the data associations as known, one can rewrite the measurement model (1) and the motion model in (2) in a general way as:

$$p(z_i|s_{1:N}) = \eta_i \exp\left((-\mathrm{e_i}(z_i,\hat{z}_i))^T \Omega_i \, \mathrm{e}_t(z_i,\hat{z}_i)\right) \,, \tag{11}$$

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 $s_{1:N}$:

$$\hat{z}(s_n, s_{n+1}) = s_{n+1} \ominus s_n \,, \tag{12}$$

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

$$\mathrm{e}(z,\hat{z}) = z \ominus \hat{z} = z \ominus (s_{n+1} \ominus s_n). \tag{13}$$

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, 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:

$$\arg\max p(s_{1:N}|\mathcal{Z}). \tag{14}$$

Knowing that $p(\mathcal{Z})$ is an unknown constant and that $p(s_{1:N})$ is uniformly distributed [24], one can rewrite (14) using the Bayes' rule as:

$$p(s_{1:N}|\mathcal{Z}) = \frac{p(\mathcal{Z}|s_{1:N})\, p(s_{1:N})}{p(\mathcal{Z})} \propto p(\mathcal{Z}|s_{1:N}). \tag{15}$$

From (11) and (15) the GraphSLAM optimization problem can be simplified into:

$$\arg\max p(s_{1:N}|\mathcal{Z}) = \arg\min \sum_{i=1}^{M} (\mathrm{e}_i(z_i,\hat{z}_i))^T \Omega_i \mathrm{e}_i(z_i,\hat{z}_i) \,, \tag{16}$$

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

$$x^* = \arg\min_x \mathrm{F}(x), \text{ where} \tag{17}$$

$$F(x) = \sum_{i=1}^{M} (\mathrm{e}_i(z_i,\hat{z}_i))^T \Omega_i \mathrm{e}_i(z_i,\hat{z}_i). \tag{18}$$

If a good initial guess $\breve{x}$ of the parameters is known, a numerical solution of (17) can be obtained from standard optimization techniques such as the Gauss-Newton [25] or the Levenberg-Marquardt [26] algorithms, through an approximation of the error function by its first order Taylor expansion around the current initial guess $\breve{x}$.

The Gauss-Newton (GN) [25] algorithm iterates through the linearization of (17) where in every iteration, the previous solution is used as the linearization point and as *initial guess*.

The Levenberg-Marquardt (LM) [26] 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 the result of the linearization directly, this method solves a damped version of it according to:

$$(H + \lambda I)\Delta x^* = -b \,, \tag{19}$$

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.

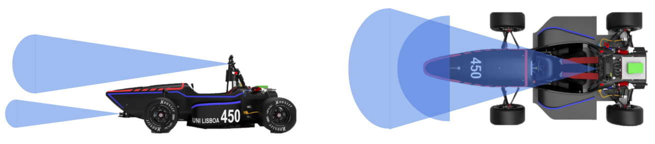
The main advantage of these these optimization techniques is that it allow for an abstraction in the SLAM problem by splitting the problem into a back-end problem and a front-end one. Most optimization techniques seek to compute the best map given the constraints (SLAM back-ends) and they typically rely on efficient implementations of common optimization algorithms, such as sparse Cholesky factorization [27] or Preconditioned Conjugate Gradients [28] (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.

III. Implementation

A. Vehicle Setup

The FST09e was the most successful and reliable prototype developed and built by the team so far, scoring in 2019 an amazing 9th 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.

In order to meet the set of challenges posed by the Driverless competition, the prototype was equipped with a series of



(a) Side view (b) Top View

Fig. 1: Location and field-of-view of LiDAR and Camera.

sensors (see Fig. 1), in order to replace the driver's ability to perceive the environment. The added sensors include a LiDAR, one RGB camera one ARHS and a powerful computer.

B. 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 is based in the ROS (*Robot Operating System*) framework [29] and is implemented in its majority in C++ programming language, 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 and velocities estimates computed by the State Estimation Pipeline.

C. 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 [5]. 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 III-B, the pipeline 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 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 III-E. 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. With the now known data associations for every particle, the EKF of each landmark can be updated. 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 [30], according to:

$$w_k = w_{k-1} l^v w_b^k w_c^\gamma \prod w_{k,n} \ , \tag{20}$$

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.

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:

$$N_{eff}^{[k]} = \frac{1}{\sum \left(w^{[k]}\right)^2}. \tag{21}$$

To detect the loop closure a simple finite state machine is used [31], 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.

D. GraphSLAM Implementation

Similarly to the FastSLAM implementation, the GraphSLAM implementation 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 [21] were introduced by including an EKF for estimating the landmark locations [32] 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 (13) is defined as the relative transformation between the two pose estimates s_a and s_b, according to:

$$\mathrm{e}^s_{t,t+1}(s_t, s_{t+1}) = z_{t,t+1} \ominus h^s_{t,t+1}(s_t, s_{t+1}) \tag{22}$$

$$\begin{aligned} h^s_{t,t+1}(s_t, s_{t+1}) &= s_a \ominus s_b \\ &= \begin{pmatrix} (x_a - x_b)\cos\theta_b + (y_a - y_b)\sin\theta_b \\ -(x_a - x_b)\sin\theta_b + (y_a - y_b)\cos\theta_b \\ \theta_b - \theta_a \end{pmatrix} . \end{aligned} \tag{23}$$

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 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:

$$\mathrm{e}^\theta_{t,i}(s_t, \theta_i) = z_{t,i} - h^\theta_{t,i}(s_t, \theta_i) \tag{24}$$

$$h^\theta_{t,i}(s_t, \theta_i) = \begin{pmatrix} (x^s_t - x^\theta_i)\cos\theta^s_t + (y^s_t - y^\theta_i)\sin\theta^s_t \\ -(x^s_t - x^\theta_i)\sin\theta^s_t + (y^s_t - y^\theta_i)\cos\theta^s_t \end{pmatrix} . \tag{25}$$

The sum of all constraints in (18) that we seek to minimize can then be rewritten has:

$$\begin{aligned} F(x) = x_0^T \Omega_o x_0 + \\ \sum_t (x^s_t - h(u_t, x^s_{t-1}))^T P_t^{-1} (x^s_t - h(u_t, x^s_{t-1})) \\ + \sum_t \sum_i (z_{t,i} - g(\theta_t, x^s_t))^T R_t^{-1} (z_{t,i} - g(\theta_t, x^s_t)) \ , \end{aligned} \tag{26}$$

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.

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. This 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 (24).

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 [24]. 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 III-C, 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.

E. 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:

$$p(z_t|s_t,\theta_{n_t},nt) = \frac{1}{(2\pi)^{n/2}\sqrt{|Q_t|}} \exp\left(-\frac{1}{2}(z_t-\hat{z}_t)^T Q_t^{-1}(z_t-\hat{z}_t)\right). \quad (27)$$

It should be added that all observations that fail the individual compatibility test 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 [33].

The Bhattacharyya distance is preferred to the Mahalanobis distance 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 [32], 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 they are saved with a unique index and passed to the SLAM pipeline as cone detections, where the EKF update takes place using the standard equations. On the other hand, in the SLAM pipeline when landmarks are created 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.

IV. 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 II-B.

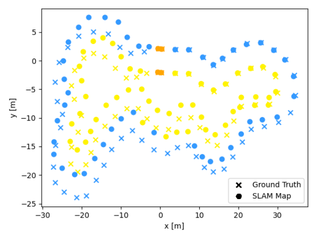
The parameters used for testing both FastSLAM and GraphSLAM implementations are presented in Table I.

A. Mapping Results

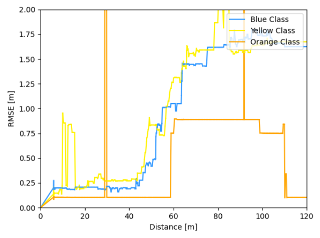
In these tests were evaluated the overall map accuracy as well as individual cone class accuracy in two different tracks. The maps were acquired at a constant speed of $3.5\,\mathrm{m/s}$, the minimum imposed by the competition regulations [2]. The maps obtained for the most demanding track are plotted against the ground truth in Figs. 2(a), 3(a) and 4(a).

TABLE I: SLAM Parameters

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

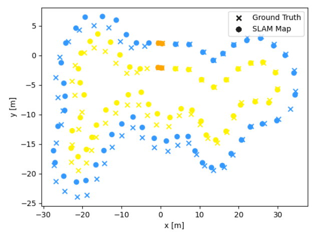


(a) Map

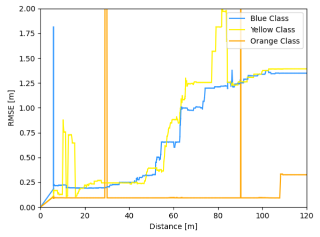


(b) RMSE per cone class

Fig. 2: FastSLAM 1.0

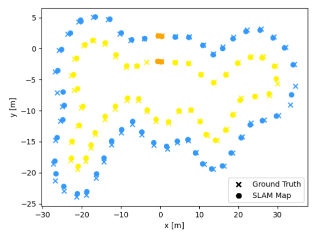


(a) Map

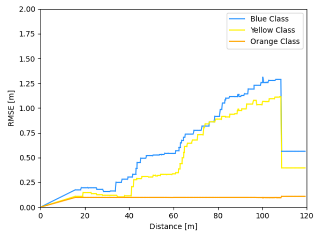


(b) RMSE per cone class

Fig. 3: FastSLAM 2.0



(a) Map



(b) RMSE per cone class

Fig. 4: GraphSLAM

In Figs. 2(a), 3(a) and 4(a) are evident the improvement from the first implementation of SLAM within the team to FastSLAM 2.0 and even more when compared to the proposed GraphSLAM implementation.

In order to quantify this improvement, the *root mean squared error* (RMSE) per cone class as a function of the travelled distance was computed and is presented in Figs. 2(b), 3(b) and 4(b), .

From the analysis of the Figs. 2(a), 3(a) and 4(a) is notorious that 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 Figs. 2(b) and 3(b). In addition, in Fig. 4(b) the role of the optimization in the GraphSLAM implementation is well demonstrated by reducing drastically the map overall error after loop closure detection from $1.37\,\mathrm{m}$ to $0.42\,\mathrm{m}$.

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

TABLE II: Total RMSE of each implementation per track

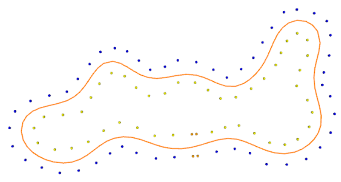
	Total RMSE [m]		
	FastSLAM 1.0	FastSLAM 2.0	GraphSLAM
Track 1	1.44	0.96	0.35
Track 2	1.68	1.06	0.41

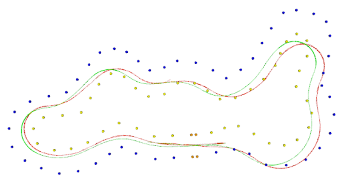
Through the analysis of Table II that the current GraphSLAM implementation represents an improvement in terms of mapping accuracy of 75% over the initial FastSLAM 1.0 implementation and 62% over FastSLAM 2.0, whistle being considerably more efficient and robust.

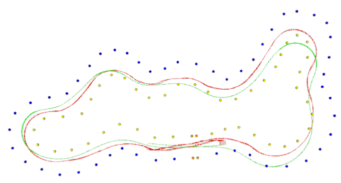
B. Localization Results

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 Fig. 5 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

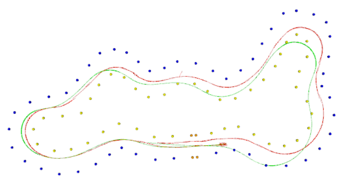
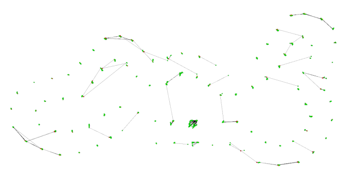

(d) GraphSLAM

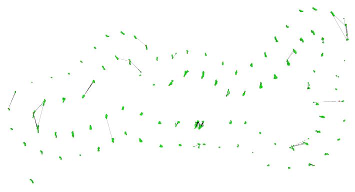
Fig. 5: Comparison of the localization results (in red) against the trajectory obtained from raw odometry data (in green). In (a) is represented, in orange, the path that the car is following.

From the analysis of Fig. 5 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.5(c)) pose estimate over FastSLAM 1.0 (Fig.5(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 Fig. 5(d).

C. Data Association Results

The data association accuracy of the proposed implementation in Section III-E was compared to the individual methods presented in Section II-B by mapping the observations in the global map frame connected to the associated landmarks.


(a) Maximum Likelihood + Individual Compatibility


(b) Joint Compatibility Branch and Bound

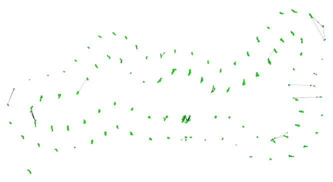

(c) Maximum Likelihood + Tracking Information

Fig. 6: 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.

The results of this test for the data association methods in study are present in Fig. 6. Just by observation, 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 III.

TABLE III: 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 Figs. 2(b), 3(b) and 4(b), corresponding to the spikes seen in the error plots of the different cone classes. The GraphSLAM algorithm using the proposed data association method had a maximum data association error of $1.6\,\mathrm{m}$, whereas FastSLAM 1.0 in the same conditions reached $1.76\,\mathrm{m}$. 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.

V. Conclusion

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 IV-A. 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. With this new approach we are able to achieve good performances regarding mapping speeds and recover an accurate map that can be used in the subsequent laps.

References

[1] J. Ibañez-Guzman, C. Laugier, J.-D. Yoder, and S. Thrun, "Autonomous driving: Context and state-of-the-art," 2012. 1

[2] Formula Student Germany, "FS Rules 2020," pp. 22–47, 2019. 1, 8

[3] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using fastslam," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 2. IEEE, 2003, pp. 1985–1991. 2, 3

[4] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017. 2

[5] M. Montemerlo and S. T. FastSLAM, *A Scalabale Method for the simultaneous localizaton and mapping problem in robotics*. Springer, 2007, vol. 27. 2, 4, 6

[6] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," *Aaai/iaai*, vol. 593598, 2002. 2, 3

[7] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986. 2

[8] Y. Bar-Shalom, P. K. Willett, and X. Tian, *Tracking and data fusion*. YBS publishing Storrs, CT, USA:, 2011, vol. 11. 2

[9] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot, "Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association," *Journal of Machine Learning Research*, vol. 4, no. 3, pp. 380–407, 2004. 2

[10] J. E. Guivant and E. M. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," *IEEE transactions on robotics and automation*, vol. 17, no. 3, pp. 242–257, 2001. 2

[11] J. J. Leonard and H. J. S. Feder, "A computationally efficient method for large-scale concurrent mapping and localization," in *Robotics Research*. Springer, 2000, pp. 169–176. 2

[12] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A. Y. Ng, "Simultaneous mapping and localization with sparse extended information filters: Theory and initial results," in *Algorithmic Foundations of Robotics V*. Springer, 2004, pp. 363–380. 2

[13] J. Nieto, J. Guivant, E. Nebot, and S. Thrun, "Real time data association for fastslam," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 1. IEEE, 2003, pp. 412–418. 2, 3

[14] A. J. Cooper, "A comparison of data association techniques for simultaneous localization and mapping," Ph.D. dissertation, Massachusetts Institute of Technology, 2005. 2, 3

[15] W. Zhou, E. Shiju, Z. Cao, and Y. Dong, "Review of slam data association study," in *2016 International Conference on Sensor Network and Computer Engineering*. Atlantis Press, 2016. 3

[16] M. Montemerlo, "Fastslam: A factored solution to the simultaneous localization and mapping problem with unknown data association," Ph.D. dissertation, Carnegie Mellon University, 2003. 3, 4

[17] J. Neira and J. D. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on robotics and automation*, vol. 17, no. 6, pp. 890–897, 2001. 3

[18] J. S. Liu and R. Chen, "Sequential monte carlo methods for dynamic systems," *Journal of the American statistical association*, vol. 93, no. 443, pp. 1032–1044, 1998. 3

[19] A. Doucet, N. De Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," *arXiv preprint arXiv:1301.3853*, 2013. 3

[20] K. P. Murphy, "Bayesian map learning in dynamic environments," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1015–1021, 1999. 3

[21] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010. 4, 6

[22] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997. 5

[23] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006. 5

[24] G. Grisetti, R. Kümmerle, H. Strasdat, and K. Konolige, "g2o: A general framework for (hyper) graph optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, 2011, pp. 9–13. 5, 7

[25] W. H. Press, H. William, S. A. Teukolsky, W. T. Vetterling, A. Saul, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007. 5

[26] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116. 5

[27] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate," *ACM Transactions on Mathematical Software (TOMS)*, vol. 35, no. 3, pp. 1–14, 2008. 5

[28] E. F. Kaasschieter, "Preconditioned conjugate gradients for solving singular systems," *Journal of Computational and Applied mathematics*, vol. 24, no. 1-2, pp. 265–275, 1988. 5

[29] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5. 6

[30] J. Kabzan, M. I. Valls, V. J. Reijgwart, H. F. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan *et al.*, "Amz driverless: The full autonomous racing system," *Journal of Field Robotics*, vol. 37, no. 7, pp. 1267–1294, 2020. 6

[31] M. I. Valls, H. F. Hendrikx, V. J. Reijgwart, F. V. Meier, I. Sa, R. Dubé, A. Gawel, M. Bürki, and R. Siegwart, "Design of an autonomous racecar: Perception, state estimation and system integration," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 2048–2055. 6

[32] L. Andresen, A. Brandemuehl, A. Honger, B. Kuan, N. Vödisch, H. Blum, V. Reijgwart, L. Bernreiter, L. Schaupp, J. J. Chung *et al.*, "Accurate mapping and planning for autonomous racing," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 4743–4749. 6, 8

[33] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bull. Calcutta Math. Soc.*, vol. 35, pp. 99–109, 1943. 8