Benchmarking SLAM Algorithms for Autonomous Formula Student Vehicle

Łukasz Michalski, lukasz.michalski.pwrrt@gmail.com
Wrocław University of Science and Technology, PWR Racing Team

I. Introduction

The rapid advancement of autonomous mobility has motivated innovation in various domains, including the development of self-driving vehicles for motorsport competitions. Formula Student, a renowned global platform for engineering students, presents a unique challenge: to create autonomous racing vehicles that can navigate both known and unknown environments while sticking to competition regulations.

A. Formula Student

Formula Student is a global engineering competition that challenges university students to design, build, and race formula-style cars. Hosted by organizations like Formula Student Germany (FSG) and the Society of Automotive Engineers (SAE), it provides a practical platform for applying theoretical knowledge. Teams from universities worldwide compete to construct high-performance, open-wheel race cars. In addition to traditional races with human drivers, there is the Formula Student Driverless category, where teams develop autonomous race cars. The Driverless Cup focuses exclusively on self-driving vehicles, challenging teams to create autonomous race cars capable of navigating racetracks and adhering to regulations without human intervention. This requires the use of cutting-edge technologies such as sensor fusion, computer vision, machine learning, and advanced control systems. The Driverless Cup promotes innovation in autonomous vehicle technology, serving as a bridge between academic research and real-world applications. It provides a unique test bed for developing safe and efficient autonomous driving algorithms in high-performance racing environments. Recently, the Driverless Cup has gained significant attention among students, researchers, and industry professionals, contributing to the advancement of autonomous vehicle technology and educating future experts in the field.

B. Motivation and Research Objectives

The primary aim of this project is to advance autonomous driving systems in high-performance contexts by evaluating SLAM-based systems for Formula Student vehicles. This research addresses the demand for efficient and safe autonomous technologies in competitive environments and promotes innovation and knowledge transfer.

II. Problem Statement

The challenge of simultaneous localization and mapping (SLAM) has been a fundamental pursuit in robotics since the mid-1980s. It involves two key tasks: determining the robot's position in a given environment (localization) and constructing a map of that environment.

Localization hinges on estimating the robot's position through sensor observations and odometry. For instance, laser sensors can provide distance measurements to nearby obstacles, while radar or camera sensors may detect specific objects like trees. Effective localization relies on having a reliable map of the environment.

Mapping, on the other hand, entails creating a map of the environment based on sensor data. Each sensor reading must be associated with a known robot location to generate an accurate map.

The interdependence of localization and mapping is evident: without a map, accurate localization is challenging, and without localization, mapping accuracy suffers. Hence, the fusion of localization and mapping poses a common challenge in robotics.

A. Localization

To solve the localization problem, we aim to derive a sequence of robot poses $\mathbf{x}_{1:T}$ given odometry readings $\mathbf{u}_{1:T}$, observations $\mathbf{z}_{1:T}$, and map $\mathbf{m}$. Each robot pose $\mathbf{x}_t$ comprises its position and orientation in a two-dimensional environment, represented as $\mathbf{x}_t = (x, y, \theta)^T$.

The localization problem can be categorized into position tracking and global localization. In position tracking, the robot's initial pose is known, and the objective is to update this pose with new measurements. Conversely, during global localization, the robot's initial pose is unknown, and its estimation relies solely on observations of the surroundings. Since SLAM assumes an unknown map, the robot's pose can be initialized as $\mathbf{x}_t = (0, 0, 0)^T$.

Although updating the pose with known starting pose and odometry readings may seem straightforward, it often proves challenging in practice. Many robots lack sensors for direct pose measurement, necessitating pose estimation using a motion model based on odometry data. This model incorporates various measurements such as robot speed, distance travelled, and angular velocity obtained from different sensors at different times. Due to imperfections in sensor data and measurement uncertainties, pose estimation based solely on odometry leads to the accumulation of noise over time, causing deviation from the true pose. Hence, correction of the pose based on observations and their correlation with the map becomes necessary.

B. Simultanous Localization and Mapping

The combination of the localization problem with simultaneous mapping of the environment in which we are located is known as a problem of simultaneous localization and mapping (SLAM). This is one of the fundamental problems of robotics. The SLAM problem occurs when the robot does not know its pose or the map of the environment in which it moves, and the only information available is observations $z_{1:T}$ and odometry readings $u_{1:T}$. Based on this information, we want to create a map m of the environment and designate the robot's route $\mathbf{x}_{1:T}$.

This is where the greatest difficulty of simultaneous localization and mapping becomes apparent. Both problems are closely dependent on each other. For localization, you need a map, and for mapping, you need a current location. The SLAM problem can be compared to the chicken and egg dilemma; one cannot exist without the other. If errors are made during mapping, they will affect the location. Not only that, the robot will be located on an incorrect map, which will additionally affect the association of observations with the map, leading to incorrect estimations. Thus, the map and location estimation will quickly deviate from reality. The situation is analogous the other way around. Without good localization and estimation of the robot's current pose, it is impossible to create a good map. High pose noise results in poor map estimation, which in turn affects even poorer pose estimation. The quality of the robot's equipment therefore directly affects the quality of mapping and localization. More accurate sensors allow for more accurate estimation. An equally well-selected model of traffic and its appropriate integration allows for minimizing location noise, thus improving the accuracy of the map (Figure II.1).

The SLAM problem can be divided into two types. The first is Online SLAM, in which we are interested in the robot's route, only its current pose:

$$p(\mathbf{x}_t, m|\mathbf{z}_{1:t}, \mathbf{u}_{1:t}) \tag{II.1}$$

With this approach, we estimate the position based on the data that is valid at time t, hence the term "online" in the name. Many algorithms solving the online SLAM problem discard data that is already processed and only focus on updating the status based only on new data. A graphical model for Online SLAM is presented in Figure II.2.

The second type, Full SLAM, is more general, the goal is to mark the entire track vehicle, i.e., the full history of poses. This means determining

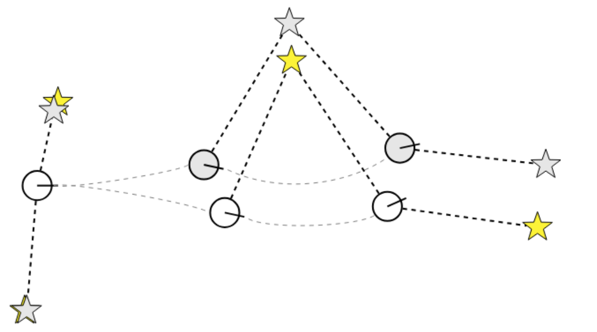

Fig. II.1. Example of map dependence on location. The white robot positions correspond to the real ones, and the grey ones are estimated. Yellow stars correspond to the actual positions of map elements, while grey ones correspond to estimated positions. [1]

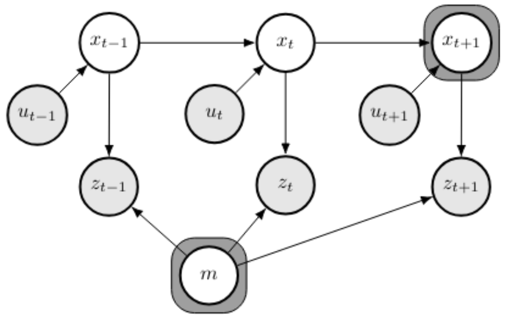

Fig. II.2. Graphical model of online SLAM, having the map m, the measurements $\mathbf{z}$, and the controls $\mathbf{u}$. The goal of localization is to estimate a posterior over the current robot pose variables $\mathbf{x}$ and the map. [2]

a common posterior probability distribution, based on the received data, for the entire path $\mathbf{x}_{1:t}$:

$$p(\mathbf{x}_{1:t}, m|\mathbf{z}_{1:t}, \mathbf{u}_{1:t}) \tag{II.2}$$

Therefore, we want to update the map history based on newly received data that may affect previous path poses. A model representing the full SLAM problem is presented in the figure II.3. This change has consequences in the type of algorithms that can be used. There is a problem with online SLAM, in particular, the result of integrating the previous poses from the full SLAM problem that are calculated one at a time when receiving new data:

$$p(\mathbf{x}_t, m|\mathbf{z}_{1:t}, \mathbf{u}_{1:t}) = \int \dots \int p(\mathbf{x}_{1:t}, m|\mathbf{z}_{1:t}, \mathbf{u}_{1:t})\, d\mathbf{x}_{t-1} \dots d\mathbf{x}_2 d\mathbf{x}_1 \tag{II.3}$$

Depending on the environment and available sensors, the representation of the map in robotics can vary between an occupancy grid (grid-based) or a feature-based map. Each type has its advantages and disadvantages, and the choice depends on the purpose of the mapping.

For applications where the goal is to create a more analogue representation, defining the boundaries of the robot's movement space, the occupancy grid approach is optimal. This approach is suitable for tasks such as room mapping by an autonomous vacuum cleaner. In an occupancy grid, occupied spaces (e.g., walls or cabinets) are assigned values different from those of free areas where the robot can move. These grid cells need not be completely filled; uncertainty in occupancy levels can be represented by partially filled cells.

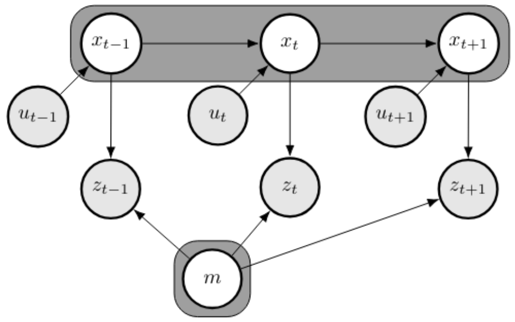

Fig. II.3. Model of the full SLAM problem. The goal is to compute a joint posterior over the whole path of the robot and the map [2]

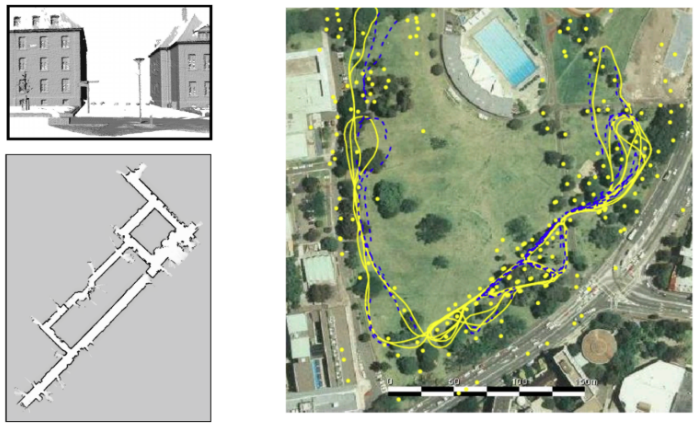

Fig. II.4. Comparison of a map based on the occupancy grid (left side) to a map of landmarks (right side). [1] ©E. Nebot

Alternatively, if the robot is tasked with mapping specific objects (e.g., trees) within a discrete space, a feature-based map is preferable. Such a map consists of a list of elements and their positions in space. Although this approach does not allow for the mapping of continuous wall elements, it simplifies working with discrete data. Figure II.4 illustrates the difference between these two approaches.

C. SLAM problem in Formula Student

In the domain of autonomous racing within Formula Student competitions, one of the primary challenges lies in establishing an accurate map representation of the track environment and precisely localizing vehicles on that map. Unlike conventional racing events, Formula Student competitions do not furnish participant teams with detailed track maps. Instead, teams are provided with foundational track specifications, necessitating the construction of the track from these basic guidelines.

Formula Student, governed by regulations from FS Germany, features four autonomous dynamic competitions. Skidpad and Acceleration contribute to overall scoring for all participating EV teams, while Autocross and Trackdrive form the optional Driverless Cup exclusively for autonomous vehicles. Track boundaries are designated by road cones, with blue and yellow cones marking the left and right edges respectively, supplemented by painted lines on the asphalt. Orange cones facilitate time measurement.

Skidpad and Acceleration competitions feature well-defined tracks with set regulations. Skidpad involves navigating an eight-shaped track for four laps — two clockwise and two counterclockwise. Acceleration focuses on achieving maximum velocity along a 75-meter linear section II.5. While

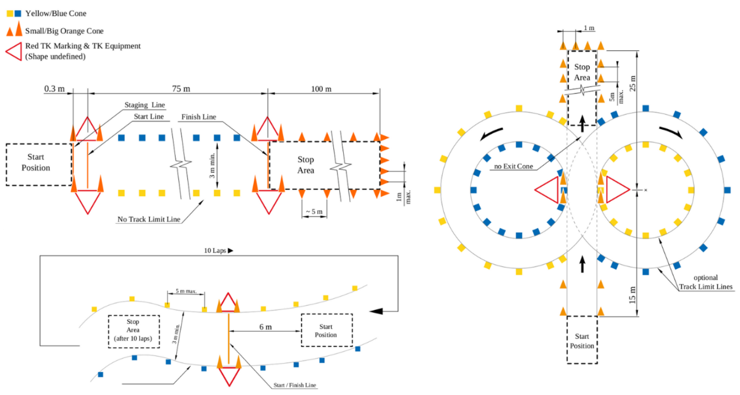

Fig. II.5. Technical regulations of Driverless Cup tracks: Top left - Acceleration; Below - Trackdrive and Autocross; Right - Skidpad

it may seem sufficient to adhere to the regulatory mandate of locating the prescribed map, achieving this task is not without its complexities. Despite possessing a clear understanding of the track's layout, generating an accurate map for precise car localization presents notable challenges. Notably, the manual placement of cones introduces potential disparities in their positions compared to regulatory specifications. Moreover, the absence of precise dimensions for the area where the car is expected to conclude the competition further complicates matters. These formidable challenges underscore the necessity of incorporating a SLAM module, which, surprisingly, proves to be equally demanding or even simpler than navigating within an ambiguously defined map.

Conversely, Driverless Cup competitions like Autocross and Trackdrive offer less precisely defined tracks. While regulations provide constraints, there's more flexibility. Teams are prohibited from mapping the track before runs, as per Formula Student regulations (D6.4.1 [3]) and that is why a reliable and accurate real-time SLAM algorithm is required.

Another significant consideration is the challenge of closing loops, which involves revisiting areas of the map that have already been mapped. In the context of the Acceleration competition, loop closure is relatively straightforward since the car continuously moves forward without retracing its path. While the SkidPad presents a slightly more intricate scenario, loop closure typically occurs accurately due to the confined area of movement, facilitating encounters with previously mapped elements.

However, in the Autocross and TrackDrive competitions, spanning expansive tracks, the infrequent or nonexistent encounters with previously mapped cones pose a significant challenge. In some cases, loop closure can only occur upon completing a lap. If mapping errors accumulate prior to loop closure, it can result in erroneous closures or failure to close the loop. This dependency creates a critical requirement for an exceptionally precise traffic model, minimizing noise and mapping inaccuracies to prevent the rapid propagation of errors in the map.

To mitigate this challenge, the SLAM module must operate seamlessly in real-time alongside numerous other modules on the autonomous system's controlling computer, such as the NVIDIA Jetson Xavier AGX (Appendix B), tailored specifically for autonomous vehicle applications. Despite satisfactory performance, the SLAM module typically operates within limited resource constraints, emphasizing the paramount importance of achieving optimal performance. Based on the AMZ Driverless autonomous system paper [4] and literature overview the research is focused on the Kalman filter, particle filter and least squares approaches for the SLAM problem.

D. Motion and observation models

The robot pose x_t results from the previous pose x_{t-1} and the control u_t that the robot performed. The robot's current pose can be written as a probabilistic function:

$$p(x_t \mid x_{t-1}, u_t)$$

This feature is called the motion model. It describes how controlling the robot affects its position on the map. Additionally, the motion model describes how noise in the control affects uncertainty in the robot's pose estimation. Noise in control may be measurement uncertainties of sensors that tell what is happening with the robot. For example, let's assume that the robot's orientation is determined using a gyroscope. Depending on how good the gyroscope the robot is equipped with, the uncertainty of the determined pose will be greater or less.

In the case of the student formula, since the robot is a racing car, the motion model is based mainly on the processing of data from inertial sensors and vehicle odometry. The motion model cannot be described as a list of control commands for the car that are associated with some inaccuracy. This is due to the limited freedom of movement of the racing car. For example, a car cannot turn ninety degrees to the right while standing still. Although in theory defining a control system that would accept this type of command is possible, a better solution is to base the motion model on the estimation of the car's state based on inertial sensors and odometry. In this work, the motion model uses motion estimation based on the Extended Kalman Filter (EKF). The Extended Kalman Filter is a well-known and commonly used estimator in robotics for slightly non-linear processes with Gaussian noise. The estimator itself is not a separate module of the autonomous system and is developed separately from the simultaneous localization and mapping module, therefore its implementation will not be described in detail in this work. The vehicle motion estimator gives the state vector:

$$\mathbf{s} = \begin{bmatrix} v_x & v_y & \omega & a_x & a_y \end{bmatrix}^T \in R^5 \tag{II.4}$$

where $v = (v_x, v_y)^T$ are the vehicle speeds, ω is the yaw rate, and $a = (a_x, a_y)^T$ are the linear accelerations. The motion estimator is based largely on the implementation in [4], excluding the estimation of tyre slip angles. The motion estimator uses inertial sensors such as an accelerometer, gyroscope, GNSS/INS (inertial-assisted satellite positioning), and wheel speed odometry to determine the state vector. The SLAM module uses the state vector to implement the motion model. Using the state vector and the old pose, a new pose is determined along with the appropriate estimation uncertainty. Since the state estimator is based on the Extended Kalman Filter, which performs sensor fusion, uncertain measurement capacity is significantly reduced. The use of EKF allows data from different sensors to complement each other, making the movement model significantly more accurate than if the position was determined using data from one sensor.

After updating the robot's pose using the motion model, the map and location, which depends on the map, must be updated. The map is updated thanks to the observations received from the perception module:

$$\mathbf{z}_t = \begin{bmatrix} r_t & \varphi_t & k_t \end{bmatrix}^T \Sigma \tag{II.5}$$

where r_t is the distance, φ_t is the angle to the observed cone, k_t is the color of this cone, and $\Sigma = \begin{pmatrix} \sigma_r & \sigma_\varphi \end{pmatrix}^T$ describes the measurement uncertainties of the observations. In other words, it is a representation of the position of the observed cone relative to the vehicle using a polar coordinate system. Such representation observation is dictated by how the perception system determines the position of the observed cones. It also allows for easy definition of measurement uncertainties when integrating the model in SLAM algorithms. If Cartesian coordinates were used instead of polar coordinates, this would require including a coordinate covariance matrix with the measurement. Integration of data presented in this way in SLAM algorithms is more difficult than in the case of a model with polar coordinates. When integrating the observations provided by the perception module through SLAM, you need to associate the data, i.e., decide which of the already mapped cones a given observation corresponds to. The observation may also not correspond to any of the previously mapped cones, but to a completely new cone that needs to be added to the map. Incorrect association of observations to the cone may result in map estimation diverging from reality, therefore appropriate association is critical for the proper operation of the SLAM algorithm. Two parameters are available to make the association: the cone's coordinates and its color. This does not allow for associations based on some unique feature of each map element. For this reason, it was decided to use associations based on the distance to the nearest mapped cone. If the distance from the observation to the nearest cone is smaller than the set threshold, the observation is associated with this cone. If the distance exceeds the threshold, the observation corresponds to a new cone.

The color contained in the observation is another attribute that can be used for association. Rejecting association cones based on color is risky, however, because there is no guarantee that the color determined on the first observation is correct. Therefore, color should be considered a possible aid if multiple cones were at a similar viewing distance. This data association problem makes the SLAM problem under consideration the so-called SLAM with unknown correspondences. Observations are determined by the perception module using two stereoscopic cameras and LiDAR. Such redundancy allows the perception module to operate, without which it would not be possible to map the environment, even if one of the sensors fails. A single failure will lead to an increase in measurement uncertainty, but it will mean failure of the entire passage.

E. Extended Kalman Filter

The previously mentioned Extended Kalman Filter (EKF) is one of the basic tools used by many SLAM algorithms. The Extended Kalman Filter has been considered a standard tool for nonlinear state estimation in robotics for many years. It is also widely used in satellite navigation systems, economics, computer vision, and many other fields. To understand how the Extended Kalman Filter works, you first need to understand the Kalman Filter. The Kalman Filter is an estimator for linear noisy states with a Gaussian distribution (normal distribution). Moreover, it is the optimal estimator for such a case. If the given problem is linear in nature, it will not work better to estimate its state than using the Kalman Filter.

The Kalman filter assumes that the state x_t is determined according to:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{II.6}$$

where A_t is the state transition model, B_t is the observation model, and ϵ_t is the process noise. The noise of the process ϵ_t is assumed to come from a multivariate normal distribution with expected value $\mu = 0$ and covariance R_t.

$$\epsilon_t \sim \mathcal{N}(0, R_t)$$

The matrices A_t, B_t, and Q_t do not have to be constant and can change with each t. For each state x_t, the expected observation z_t is determined according to:

$$z_t = C_t x_t + \delta_t \tag{II.7}$$

where C_t is the observation model and δ_t is the observation noise, which, similarly to process noise, comes from a multivariate normal distribution with expected value $\mu = 0$ and covariance Q_t.

$$\delta_t \sim \mathcal{N}(0, Q_t)$$

Equation II.6 describes the process model for the probability $P(x_t \mid u_t, x_{t-1})$, which is determined by inserting equation II.6 into the definition of the multivariate normal distribution. We proceed analogously with equation II.7, which describes the model of predicted observation for probability $P(z_t \mid x_t)$.

$$p(x_t \mid u_t, x_{t-1}) = \frac{\exp\left(-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right)}{\sqrt{(2\pi)^n |R_t|}} \tag{II.8}$$

$$p(z_t \mid x_t) = \frac{\exp\left(-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\right)}{\sqrt{(2\pi)^m |Q_t|}} \tag{II.9}$$

Due to the fact that linear models are used in equations II.8 and II.9 and the input probability is normally distributed, it is guaranteed that the output probability will also be normally distributed. If the model were not linear, the output probability would not be normally distributed. The Kalman filter is not able to work with a probability distribution other than normal. Hence the limitation to linear models for the Kalman Filter. The algorithm for calculating the probabilities from equations II.8 and II.9 consists of two stages. The first stage is the prediction stage, in which a priori probabilities are determined. This allows you to predict what the model state should be and what the observation should be, based on the current state. This is followed by the second stage of the update. Its purpose is to update the state estimate based on the received observations and previously calculated predictions. For this purpose, the Kalman gain K_t is used. Kalman gain describes whether the algorithm should trust the predictions or the received observation more.

Algorithm 1: Kalman Filter

```
1: Input: $\mu_{t-1}$, $\Sigma_{t-1}$, $u_t$, $z_t$
2: Output: $\mu_t$, $\Sigma_t$
3: $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$                          ▷ Prediction step
4: $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
5: $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$     ▷ Update step
6: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t)$
7: $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$
```

In reality, however, the vast majority of things are non-linear and do not have a perfect normal distribution. If the model includes orientation in any way, it will use trigonometric functions, which are non-linear and their use spoils the Kalman Filter. If we have a motion model that includes the robot's orientation or observation with an included angle to the observed object, it cannot be estimated using the Kalman Filter. This introduces a big problem and makes the Kalman Filter fail to have practical applications in the real world. For this reason, the Extended Kalman Filter was created, the purpose of which is to estimate nonlinear models. It does this by locally linearizing a nonlinear function. Thanks to this, you can use the Kalman Filter; however, this is no longer an optimal estimation, which is the result of the previously used linearization, which is only an approximation of the function in a given place. Additionally, the quality of estimation using the Extended Kalman Filter depends on the function that is linearized. The more nonlinear it is, the worse its linearization will be, which translates into the quality of the estimation. The linearization technique that EKF uses is the Taylor series (first order). To linearize the function g locally, the Taylor series uses the value and slope of g at a given point. The slope is determined using the partial derivative

$$g_0'(u_t, x_{t-1}) = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}}$$

Since the value and slope of g depend on the place of linearization, it should be performed for the state that is considered the most probable at the time of linearization. In other words, since the estimation is normally distributed, linearization is performed on the expected value of this normal distribution. The state prediction (equation II.10) and predicted observation (equation II.11) are therefore determined as follows

$$\begin{aligned} g_0'(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1}) \\ &= g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \end{aligned} \tag{II.10}$$

$$\begin{aligned} h(x_t) &\approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)(x_t - \bar{\mu}_t) \\ &= h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \end{aligned} \tag{II.11}$$

where G_t and H_t are the Jacobi matrices and $h_0'(x_t) = \frac{\partial h(x_t)}{\partial x_t}$ is determined analogously to $g'(u_t, x_{t-1})$. The matrix G_t has size $n \times n$, which corresponds to a state vector of size n and is as follows

$$G_t = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \dots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \dots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial x_1} & \frac{\partial g_n}{\partial x_2} & \dots & \frac{\partial g_n}{\partial x_n} \end{pmatrix} \tag{II.12}$$

Having a linear approximation of the prediction of the state and the predicted observation, analogously to equations II.8 and II.9, the update of the probability distribution of the state and the predicted observation using the definition of a multivariate normal distribution is carried out in the following linearized model.

$$p(x_t \mid u_t, x_{t-1}) = \frac{\exp\left(-\frac{1}{2}(x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1}))^T R_t^{-1}(x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1}))\right)}{\sqrt{(2\pi)^n |R_t|}} \tag{II.13}$$

$$p(z_t \mid x_t) = \frac{\exp\left(-\frac{1}{2}(z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t))^T Q_t^{-1}(z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t))\right)}{\sqrt{(2\pi)^m |Q_t|}} \tag{II.14}$$

Linearizing the models ensures that the output probability distribution remains normal. Without it, the output distribution would not be normal and the Kalman Filter could not be applied. The Augmented Kalman Filter algorithm is presented in pseudocode 2. As you can see, the differences between the extended and regular versions of the Kalman Filter include the use of $g(u_t, \mu_{t-1})$ instead of $A_t\mu_{t-1} + B_t u_t$ and $h(\bar{\mu}_t)$ instead of $C_t\bar{\mu}_t$. Additionally, instead of the A_t matrix, the Jacobi matrix G_t is used. Similarly, H_t is used instead of C_t.

Algorithm 2: Extended Kalman Filter

```
1: Input: $\mu_{t-1}$, $\Sigma_{t-1}$, $u_t$, $z_t$
2: Output: $\mu_t$, $\Sigma_t$
3: $\bar{\mu}_t = g(u_t, \mu_{t-1})$                                ▷ Prediction step
4: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
5: $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$          ▷ Update step
6: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
7: $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
```

III. EKF SLAM

The basic algorithm used in robotics to solve the problem of simultaneous localization and mapping (SLAM) is an algorithm using the extended Kalman filter (EKF SLAM). Maps created by EKF SLAM are maps of orientation elements, which is part of the SLAM problem in the context of autonomous vehicles. Additionally, in most cases, EKF SLAM is considered for the online SLAM problem, which also fits well into the problem solved in this work. The EKF SLAM algorithm will be first considered to solve the SLAM problem for the RT12e car, due to its impact on the entire literature dealing with the problem of SLAM.

The state vector estimated by the extended Kalman filter for the SLAM problem contains the vehicle's pose and the map. A map is a list of landmark positions. This means that the normal distribution of the state probability has $3 + 2n$ dimensions and is presented as follows:

$$\mu = \begin{pmatrix} x \\ y \\ \theta \\ m_{1,x} \\ m_{1,y} \\ \vdots \\ m_{n,x} \\ m_{n,y} \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xm_{1,x}} & \sigma_{xm_{1,y}} & \cdots & \sigma_{xm_{n,x}} & \sigma_{xm_{n,y}} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} & \sigma_{ym_{1,x}} & \sigma_{ym_{1,y}} & \cdots & \sigma_{ym_{n,x}} & \sigma_{ym_{n,y}} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} & \sigma_{\theta m_{1,x}} & \sigma_{\theta m_{1,y}} & \cdots & \sigma_{\theta m_{n,x}} & \sigma_{\theta m_{n,y}} \\ \sigma_{m_{1,x}x} & \sigma_{m_{1,x}y} & \sigma_{m_{1,x}\theta} & \sigma_{m_{1,x}m_{1,x}} & \sigma_{m_{1,x}m_{1,y}} & \cdots & \sigma_{m_{1,x}m_{n,x}} & \sigma_{m_{1,x}m_{n,y}} \\ \sigma_{m_{1,y}x} & \sigma_{m_{1,y}y} & \sigma_{m_{1,y}\theta} & \sigma_{m_{1,y}m_{1,x}} & \sigma_{m_{1,y}m_{1,y}} & \cdots & \sigma_{m_{1,y}m_{n,x}} & \sigma_{m_{1,y}m_{n,y}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sigma_{m_{n,x}x} & \sigma_{m_{n,x}y} & \sigma_{m_{n,x}\theta} & \sigma_{m_{n,x}m_{1,x}} & \sigma_{m_{n,x}m_{1,y}} & \cdots & \sigma_{m_{n,x}m_{n,x}} & \sigma_{m_{n,x}m_{n,y}} \\ \sigma_{m_{n,y}x} & \sigma_{m_{n,y}y} & \sigma_{m_{n,y}\theta} & \sigma_{m_{n,y}m_{1,x}} & \sigma_{m_{n,y}m_{1,y}} & \cdots & \sigma_{m_{n,y}m_{n,x}} & \sigma_{m_{n,y}m_{n,y}} \end{pmatrix} \tag{III.1}$$

Since the above notation is very complex, the simplified notation is clearer:

$$\mu = \begin{pmatrix} x \\ m \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xm} \\ \Sigma_{mx} & \Sigma_{mm} \end{pmatrix}$$

where $x = \begin{pmatrix} x & y & \theta \end{pmatrix}^T$ is the current pose of the vehicle, and $m = \begin{pmatrix} m_{1,x} & m_{1,y} & \cdots & m_{n,x} & m_{n,y} \end{pmatrix}^T$ is the map. The matrices Σ_{xx} and Σ_{mm} are the covariance matrices for pose and map, respectively. Σ_{xm} and Σ_{mx} describe the relationship between the pose and the map.

Before starting localization and mapping with EKF SLAM, the state vector μ_0 and the covariance matrix Σ_0 need to be initialized. The vehicle pose is assumed to be $\begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T$ and all map elements have infinite variance $\sigma_m = \infty$. This is a simplification for presenting the operation of the algorithm; in practice, map elements are added to the matrix when they are first observed.

$$\mu_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \Sigma_0 = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \infty & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \infty \end{pmatrix}$$

The purpose of the Kalman filter prediction stage in this case is to update the robot's pose based on the current condition of the vehicle. The vehicle condition is also determined using an extended Kalman filter, which enables the connection of SLAM and the motion estimation module. However, for simplicity, motion estimation will remain a separate module. According to equation II.5, we can use the state vector obtained from the motion estimation module for prediction:

$$\begin{pmatrix} x_0 \\ y_0 \\ \theta_0 \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} v_x \Delta t \\ v_y \Delta t \\ \omega \Delta t \end{pmatrix}$$

The above equation can be used to update the full state vector:

$$\begin{pmatrix} x_0' \\ y_0' \\ \theta' \\ \vdots \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \\ \vdots \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{pmatrix}^T \begin{pmatrix} v_x \Delta t \\ v_y \Delta t \\ \omega \Delta t \\ \vdots \end{pmatrix}$$

The above equation describes the state prediction. It is worth noting that in this case it is linear, because the velocity vector $\mathbf{v}_t = (v_x, v_y)^T$ already contains information about the direction of velocity, so there is no need to determine it based on the current orientation. The $\mathbf{F}_x^T$ matrix allows you to update the pose without changing information about map elements. It remains to determine the Jacobian matrix for the above function, which is given by:

$$\mathbf{G}_t = \begin{pmatrix} \mathbf{G}_x^t & 0 \\ 0 & \mathbf{I} \end{pmatrix}$$

where

$$\mathrm{G}_x^t = \tfrac{\partial}{\partial(x,y,\theta)^T}\left[\begin{pmatrix}x\\y\\\theta\end{pmatrix}+\begin{pmatrix}v_x\Delta t\\v_y\Delta t\\\omega\Delta t\end{pmatrix}\right] = I + \tfrac{\partial}{\partial(x,y,\theta)^T}\begin{pmatrix}v_x\Delta t\\v_y\Delta t\\\omega\Delta t\end{pmatrix} = \begin{pmatrix}1&0&\Delta t\\0&1&\Delta t\\0&0&1\end{pmatrix} \tag{III.2}$$

Performing the update step requires determining the predicted observation. We assume that it is known which element corresponds to each observation from $\mathbf{z}_i^t = (r_t^i, \phi_t)^i$. In the case of the first observation of a given element, you only need to initialize its position:

$$\begin{pmatrix}\overline{\mu}_{j,x}\\\overline{\mu}_{j,y}\end{pmatrix} = \begin{pmatrix}\overline{\mu}_{t,x}\\\overline{\mu}_{t,y}\end{pmatrix} + r_t^i\begin{pmatrix}\cos(\phi_t)^i+\overline{\mu}_{t,\theta}\\\sin(\phi_t)^i+\overline{\mu}_{t,\theta}\end{pmatrix}$$

In case the element has been previously observed, you need to determine the predicted observation for the given element:

$$\delta = \begin{pmatrix}\delta_x\\\delta_y\end{pmatrix} = \begin{pmatrix}\overline{\mu}_{j,x}-\overline{\mu}_{t,x}\\\overline{\mu}_{j,y}-\overline{\mu}_{t,y}\end{pmatrix}$$

$$q = \delta^T\delta$$

$$\hat{\mathbf{z}}_t^i = h(\mu_t) = \begin{pmatrix}\sqrt{\delta^T\delta}\\\arctan 2(\delta_y,\delta_x)-\mu_{t,\theta}\end{pmatrix}$$

All that remains is to determine the Jacobian matrix $\mathbf{H}_t$. According to the lecture on EKF SLAM from [15], the $\mathbf{H}_t$ matrix for a given element j is:

$$q = \delta^T\delta \tag{III.3}$$

$$\widetilde{H_t^j} = \frac{\partial h(u_t)}{\partial u_t} = \frac{1}{q}\begin{pmatrix}-\sqrt{q}\delta_x & -\sqrt{q}\delta_y 0 & \sqrt{q}\delta_x & \sqrt{q}\delta_y & \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x\end{pmatrix} \tag{III.4}$$

$$F_{x,j} = \begin{pmatrix}1&0&0&0\cdots0&0&0&0\cdots0\\0&1&0&0\cdots0&0&0&0\cdots0\\0&0&1&0\cdots0&0&0&0\cdots0\\0&0&0&0\cdots0&1&0&0\cdots0\\0&0&0&\underbrace{0\cdots0}_{2j-2}&0&1&\underbrace{0\cdots0}_{2N-2j}\end{pmatrix} \tag{III.5}$$

$$H_t^j = \widetilde{H_t^j}F_{x,j}$$

The time and memory complexity of the simultaneous localization and mapping algorithm based on the Kalman filter depends on the number of map elements. Both time complexity and memory usage are $O(n^2)$. Because the module is supposed to work in real-time and is implemented on an embedded system, whose resources are severely limited by other modules operating simultaneously, it was decided to look for a better solution than EKF SLAM even if for [4] solution was working.

IV. Particle Filter

The Kalman Filter presented earlier only accepts normally distributed models, which is a significant limitation, especially when the modeled phenomenon's distribution deviates significantly from normal. An example of a filter that assumes an arbitrary probability distribution, even one from which sampling is not straightforward, is the particle filter. In a particle filter, a given distribution is represented by a finite number of samples (particles). States with more particles have higher probabilities. Additionally, each particle may be assigned a weight to indicate its likelihood relative to others, reflecting the state's probability.

However, a drawback of this representation is its computational complexity. A large number of particles is required to accurately approximate the probability distribution. Specifically, the accuracy of a particle filter is directly proportional to the number of particles used. Moreover, if the model exhibits a wide distribution (high noise), more particles are needed compared to a model with a narrow distribution (low noise). This contrasts with the Kalman filter, which performs consistently well regardless of the distribution's width.

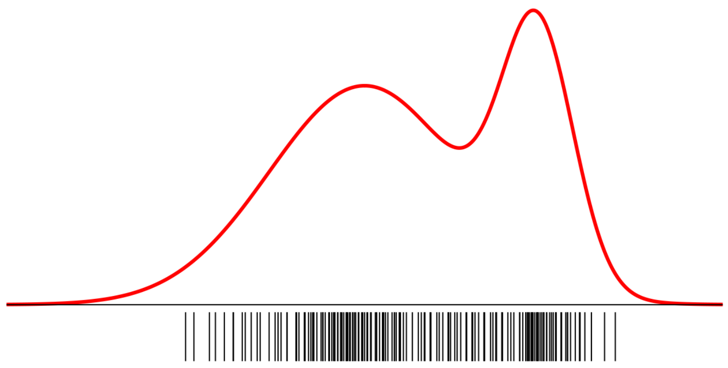
Fig. IV.1. Example of a particle filter on a distribution defined by f(x). The samples (particles) are there condensed in places of greater probability. red if $f(x)$, — are samples

Thus, a particle filter consists of a collection of particles, where each particle corresponds to a sample from the probability distribution of a given model and represents a hypothesis about the state in which the model resides:

$$X = \left\{\left\langle x^{[j]}, w^{[j]}\right\rangle\right\}_{j=1,\cdots,J}$$

Above equation describes the set of particles comprising a particle filter. Here, $x^{[j]}$ denotes a state hypothesis with weight $w^{[j]}$. Figure IV.1 provides an example of such a sampling distribution.

The particle filter provides a solution for representing unconventional distributions by utilizing particles. However, if the distribution in question does not facilitate straightforward probability-based sampling, efficiently creating a population of particles can be challenging. The particle filter addresses this issue through importance sampling, a technique for approximating a target distribution using another distribution from which efficient sampling is feasible. Particles are sampled from this proposed distribution, denoted as π:

$$x^{[j]} \sim \pi(x^{[j]} \mid \ldots)$$

where j indexes the particles. Typically, π could be a distribution like the normal distribution, which allows for efficient sampling. The weight of each particle is then determined based on its compatibility with the target distribution p, as depicted in Figure IV.2:

$$w^{[j]} = \frac{target(x^{[j]})}{proposal(x^{[j]})} \tag{IV.1}$$

The final step involves resampling, where a new population of particles is generated based on these weights. Particles with higher weights, indicating better fit to the target distribution, have a higher likelihood of surviving in the resampling process. This procedure condenses states with higher probability under the target distribution by sampling new particles from the proposed distribution.

An efficient algorithm for this process is Universal Stochastic Sampling, which allows linear-time sampling from a given distribution. Figure IV.2 illustrates the process of sampling using the proposed distribution $\pi(x)$

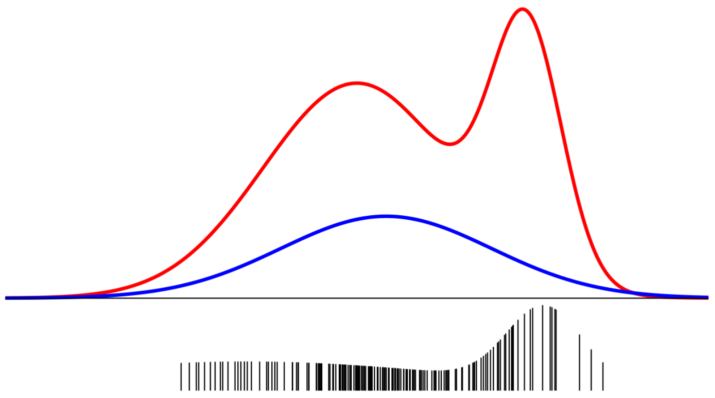

Fig. IV.2. Sampling using the proposed g(x) distribution and then determining sample weights using a given distribution f (x). Probable states will be condensed after resampling for the distribution f(x). red is $f(x)$, blue is $g(x)$ — are samples

and determining weights using the target distribution $p(x)$. If all particles have identical weights, the resulting population would be identical as well.

The particle filter algorithm is outlined in pseudocode 3. Particle filters are employed in various localization algorithms, such as Monte Carlo localization, particularly effective when dealing with distributions that deviate significantly from normal. An example scenario is bimodal localization, where there are two highly probable locations where a robot might be situated. This situation arises in maps containing multiple identical rooms, where without additional information, the robot cannot definitively determine its precise location, leading to particles being distributed across potential rooms.

Algorithm 3: Particle Filter

```
Input: $X_{t-1}, u_t, z_t$
Output: $X_t$
1  $\bar{X}_t = \emptyset$, $X_t = \emptyset$;
2  for $j = 1$ to $J$ do
3                                  ▷ Sampling from proposed distribution
4      sample $x_t^{[j]} \sim \pi(x_t \mid \ldots)$;
5      $w_t^{[j]} = \frac{p(x_t^{[j]})}{\pi(x_t^{[j]} \mid \ldots)}$;
6      $\bar{X}_t = \bar{X}_t \cup \{(x_t^{[j]}, w_t^{[j]})\}$;
7  end for
8  for $j = 1$ to $J$ do
9                                  ▷ Resampling
10     get $i \in \{1, \ldots, J\}$ with probability $w_t^{[i]}$, add $x_t^{[i]}$ to $X_t$;
11 end for
12 return $X_t$;
```

V. FastSLAM

Trying to solve the problem of simultaneous localization and mapping using a particle filter reveals a challenge: poor performance in high-dimensional states. To illustrate, estimating a state space with numerous dimensions requires an exponentially large number of particles. Specifically, the particle count increases exponentially with the number of dimensions. Since SLAM inherently involves estimating both the robot's pose and the positions of map elements, where each map element adds dimensions to the state vector, particle filters are impractical for estimating even small maps with a few dozen elements. However, they are suitable for estimating the robot's pose, which led to the design of the FastSLAM 1.0 algorithm.

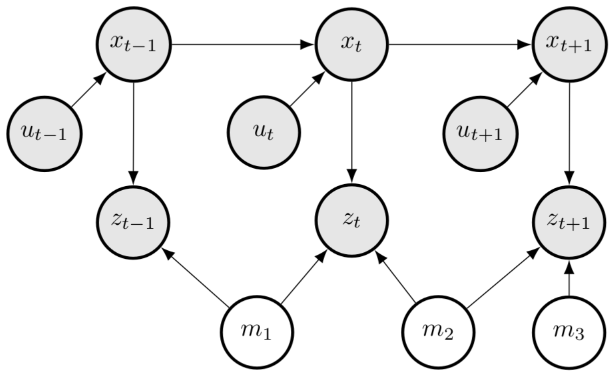

Fig. V.1. Mapping relationships with a known robot path. All map elements become separated from each other independent. No path from one element to another passes through an unknown value.

FastSLAM 1.0, introduced in 2002 by Michael Montemerlo and Sebastian Thrun [5], leverages a key insight into the mapping problem. Specifically, when the robot's path is known, map elements become conditionally independent. This means that given the vehicle's current position, each map element (e.g., cones) can be estimated independently of others, as shown in Figure V.1. Consequently, the SLAM problem can be decomposed as follows:

$$\begin{aligned} p(x_{1:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) &= p(x_{1:t} \mid z_{1:t}, u_{1:t}) P(m_{1:M} \mid x_{1:t}, z_{1:t}) \\ &= p(x_{1:t} \mid z_{1:t}, u_{1:t}) \prod_{i=1}^{M} p(m_i \mid x_{1:t}, z_{1:t}) \end{aligned} \tag{V.1}$$

Thus, each map element's position can be estimated using a separate 2x2 EKF covariance matrix, rather than a single large EKF with an MxM covariance matrix, thereby speeding up map updates from quadratic to linear complexity.

However, the robot's pose is not known a priori. To maintain this pose uncertainty during map updates, a particle filter is employed to represent the pose. Each particle estimates the robot's pose and maintains its own map, where each map element is represented by a separate EKF. The particle filter takes the form:

$$X_t = \{\left\langle x_t^{[n]}, \mu_{1,t}^{[n]}, \Sigma_{1,t}^{[n]}, \ldots, \mu_{M,t}^{[n]}, \Sigma_{M,t}^{[n]} \right\rangle\}_{n=1,\cdots,N}$$

Here, only the pose is included in the proposal distribution, while each map element is represented by a normal distribution. This approach is known as Rao-Blackwellized particle filtering.

When updating the vehicle's position using the state vector from the motion estimation module, new particle positions are sampled based on the motion model:

$$x_t^{[n]} \sim p(x_t \mid x_{t-1}^{[n]}, u_t)$$

This operation causes particles to spread out, increasing uncertainty about the vehicle's position. Subsequently, upon receiving observations, the weight and map update for each particle are determined. Observations are more aligned with particle maps closer to the true state, hence the weight of each particle is determined based on the difference between predicted and actual observations:

$$w^{[n]} = \frac{\exp\left(-\frac{1}{2}(z_t - \bar{z}^{[n]})^T Q^{-1}(z_t - \bar{z}^{[n]})\right)}{\sqrt{(2\pi)^m |Q|}} \tag{V.2}$$

where $\bar{z}^{[n]}$ is the predicted observation for the given particle. The derivation of this formula from equation IV.1 can be found in [2].

Map updates are performed separately for each particle, as each particle maintains its own map. Since each map element is stored as a separate 2x2 EKF, updates are executed independently for each element, resulting in $N \cdot M$ EKF updates.

Finally, resampling is conducted based on previously computed particle weights. This concentrates particles in more likely states while reducing particles in less likely states. New particles inherit both the position and map from their parent particle; however, map information is not shared between particles, only their common history.

VI. Simulation

To test the operation of the algorithms before implementing the module in an autonomous system, a test application was implemented. Its purpose was to get acquainted with the algorithm and a comparison of possible solutions for its implementation.

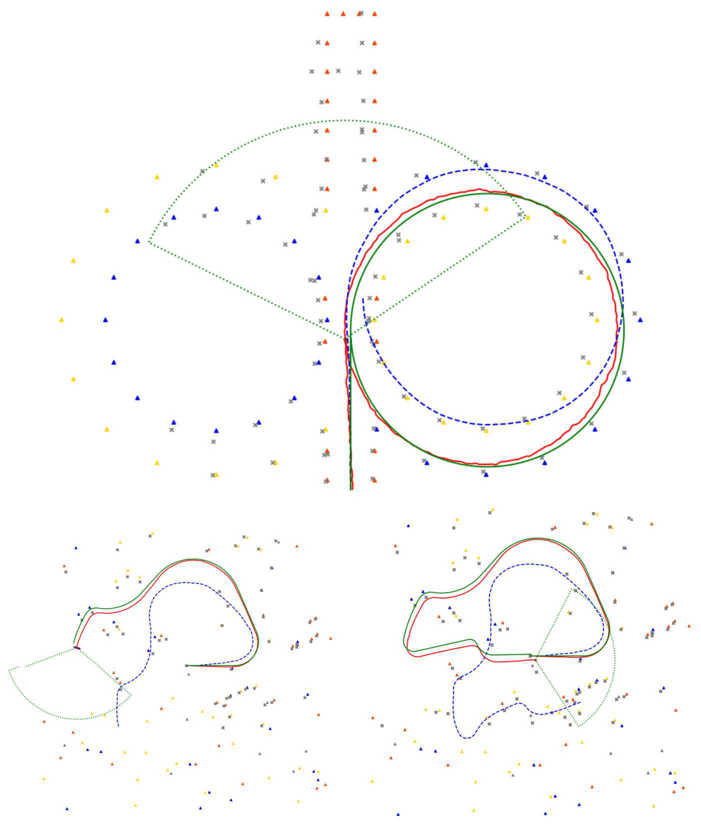

Fig. VI.1. On top: EKF SLAM on Skidpad mission, below: FastSLAM working in simulation and benchmark environment (left). On the right loop closure attempt, blue line is the odometry path, SLAM path in red and green path is the real simulation trajectory of the car. The odometry-only localization problem can be seen very quickly as it begins to diverge from the real path very quickly. The green dashed line indicates the range of perception, gray crosses mark map positions of mapped landmarks / cones

The test application was written in Python and the matplotlib library was used for visualization. It was decided to implement the FastSLAM algorithm in version 1.0 instead of version 2.0 due to the lower complexity of the first version. This allowed to achieve the intended effect with less effort, without the need for additional debugging of a complicated part of the code that should be implemented in the second version. This allowed us to quickly get the test application up and running. The map is randomly generated by the test application. It is based, similarly to the target implementation, on landmarks. The robot moves along a predefined path. This path circles the map (Skidpad), which makes it possible to test the loop closure. Perception and odometry readings are simulated in the application along with the corresponding noise. This allowed for effective testing of the algorithm's operation in conditions similar to those present in the default implementation The test application turned out to be a success and allowed the author to familiarize himself with the operation of the EKF SLAM and FastSLAM algorithms, the evaluation of the algorithms and the problem of simultaneous localization and mapping. The operation of the application is presented in Figure VI.1

VII. Results

A. Performance with Increasing Landmarks

- FastSLAM: Shows varying pose errors with decreasing landmarks number. Notably, the pose error tends to decrease slightly as the number of landmarks decreases, indicating the pose correction when it comes to more reference cones. This is evident in the error bars which represent the variability across runs or simulations.
- EKF: Displays a steady trend where the pose error also fluctuates in 50 landmarks but generally shows lower error values compared to FastSLAM.

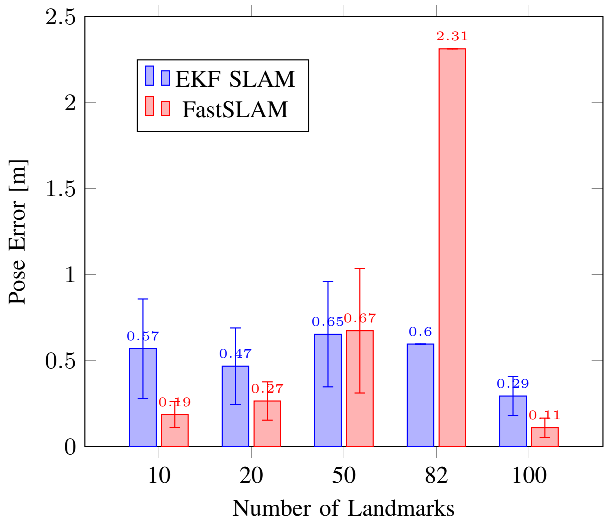

Fig. VII.1. Pose error comparison based on landmarks number (82 skidpad).

VIII. Summary

The basic goal of this work was achieved, the analysis of simultaneous localization and mapping algorithms that are able to correctly map the Formula Student tracks. However, it was not possible to optimize the modules' performance and configure them optimally. As a result, the modules are not reliable and consume significant amounts of computing resources, which means that there are situations when they are unable to operate in real-time. Moreover, the implementation and tests of GraphSLAM were not performed which was one of the main goals of the research mostly because of the complexity of the algorithm and lack of time.

Thanks to the work done during writing, By analyzing the problem of simultaneous localization and mapping in this work, the author knows where to look for problems with the operation of algorithms and in which directions to push its development.

The goal of implementing a more advanced module (GraphSLAM) so that it would work during real driving of the RT15e car at the Formula Student competition in 2025 remains valid. The author will make every effort to ensure that the module and the entire autonomous system are ready for the competition in 2025.

References

[1] C. Stachniss, “Robot mapping.” [Online]. Available: http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/
[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Mass.: MIT Press, 2005.
[3] ©FSG, “Formula Student Rules 2024.” [Online]. Available: https://www.formulastudent.de/fileadmin/user_upload/all/2024/rules/FS-Rules_2024_v1.1.pdf
[4] J. Kabzan, M. de la Iglesia Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dubé, A. Gawel, M. Pfeiffer, A. Liniger, J. Lygeros, and R. Siegwart, “Amz driverless: The full autonomous racing system,” 2019.
[5] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, “Fastslam: a factored solution to the simultaneous localization and mapping problem,” in *Eighteenth National Conference on Artificial Intelligence*. USA: American Association for Artificial Intelligence, 2002, p. 593–598.

Appendix A
PWR Racing Team

The PWR Racing Team was the first established Polish Formula Student team. It has been operating continuously since 2009 by the Wrocław University of Science and Technology. Throughout its history, the team has constructed 12 combustion cars, achieving podium finishes numerous times. Last year marked the end of this era, as the team with RT11B secured 2nd place in the competition held in Michigan, USA. Additionally, the car is still being utilised by the team for various purposes, including driver skill development.

In March 2020, the team initiated the concept of an electric car with autonomous driving systems. As a result, two such constructions have already been built, being part of a 4-years development plan for an efficient transition into the electric era. It describes the desired goals of each particular construction. The purpose of the first car, a development platform, was to introduce the team to entirely new systems related to the construction of electric vehicles. The subsequent one aimed to utilise the acquired technologies to improve critical, non-functioning components. The victory of RT13e in one of the competitions demonstrated that the direction taken by the team was correct.

As part of the plan, adjustments have been also made to the team’s structure for the electric car project, along with investments in infrastructure to enhance the efficiency and safety of team members. The introduced structure in the previous season fostered interdepartmental collaboration.

Drawing on the experience accumulated over many years of operation, the PWR Racing Team is participating in five editions of competitions in 2024 with the RT14e race car.

Appendix B
Autonomous System Master Unit

One of the decisions taken at the beginning of the development process was to choose the main computing unit, responsible for perception, estimation, and decision-making for the car. In terms of the design assumptions, the team wanted a unit that is compact and easily integrated with ROS 2 software and hardware via industry-standard interfaces. The unit needs sufficient computing power to keep up with vehicle speed while receiving, processing, and sending all the necessary data. In the end, the team chose NVIDIA Jetson AGX Orin for the following reasons:

- very good specification for its size and price
- compact design alongside required interfaces such as CAN, USB 3.1, and Ethernet
- great ROS 2 and other Linux software support
- extensive documentation
- NVIDIA SDK for image processing
- experience with the previous version of the unit - Nvidia Jetson Xavier