ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CENTER FOR PROJECT-BASED LEARNING

Department of Information Technology and Electrical Engineering

Spring Semester 2023

Investigation of Localization Techniques in Autonomous Racing in the Context of F1TENTH

Semester Thesis

Tian Yi **Lim**
tialim@student.ethz.ch

20 June 2023

Supervisors: Nicolas Baumann, nicolas.baumann@pbl.ee.ethz.ch
Edoardo Ghignone, edoardo.ghignone@pbl.ee.ethz.ch
Professor: Prof. Dr. Florian Dörfler, dorfler@ethz.ch

Acknowledgements

Firstly, I would like to thank my supervisors, Nicolas Baumann and Edoardo Ghignone, for their guidance and leadership. This semester project was rewarding and getting to see my algorithms running in real life was often the highlight of my week. I also learned organization and leadership from both of you, from the weekly race team meetings to the motivational and organizational talks through the weeks.

I would also like to thank my ForzaETH teammates, Filippo, Luca, Michael, Niklas, and Tobias. I am glad for your cooperation and teamwork, and it was refreshing working with like-minded individuals who love applying engineering to going fast. Of course, having the opportunity to win the F1TENTH Grand Prix in London this year was a *small* bonus. I wish you all best of luck with your future endeavours, and I am sure you will find success in whatever you set your minds to.

Next, I would like to thank the Center for Project Based Learning, particularly Dr Michele Magno, for setting this F1TENTH project. I feel like I learned so much more with the applications of my thesis at hand!

Last but not least, I would like to thank my family and friends for their endless support, especially when the project got stressful.

Abstract

Autonomous racing places strict requirements on the performance of localization algorithms. Such algorithms need to provide accurate pose estimates at high frequency. This work investigates localization in the context of the F1TENTH competition, where 1/10 scale autonomous race cars race in fast-paced, head-to-head scenarios. Previously, the ForzaETH team used *Google Cartographer*, a SLAM-based localization method. While performant in most scenarios, it was found to perform poorly in scenarios with poor LiDAR and odometry sensor input.

An investigation of the conditions that cause poor localization performance with Cartographer was performed. In addition, a Monte-Carlo Localization algorithm, PF2, was developed which integrated several improvements in the context of autonomous racing.

It was found that odometry quality heavily influences the performance of Cartographer. In scenarios with good sensor input, Cartographer was found to perform better than PF2. However, PF2 was more robust to Cartographer in adverse sensor conditions.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix B

Tian Yi **Lim**,
Zurich, 20 June 2023

Contents

List of Figures

List of Tables

List of Acronyms

AMCL . . . . . Adaptive Monte-Carlo Localization

ICP . . . . . Iterative Closest Point

IMU . . . . . Inertial Measurement Unit

LiDAR . . . . . Light Detection and Ranging

MCL . . . . . Monte-Carlo Localization

MIT . . . . . the Massachusetts Institute of Technology

NDT . . . . . Normal Distributions Transform

PBL . . . . . Center for Project-Based Learning

PF . . . . . Particle Filter

PGO . . . . . Pose Graph Optimization

ROS . . . . . Robot Operating System

SLAM . . . . . Simultaneous Localization and Mapping

TUM . . . . . the Technical University of Munich

Chapter 1

Introduction

1.1. The F1TENTH Competition

The F1TENTH Autonomous Vehicle System [1] is an open-source platform for autonomous systems research and education at a 1/10 scale. A system diagram is shown in fig. 1.1. The system also defines specifications for yearly competitions held during robotics conventions.

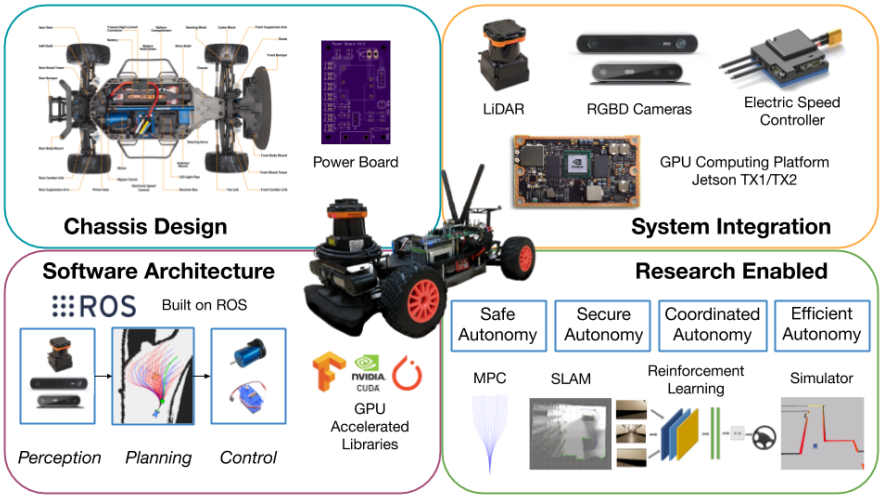

Figure 1.1.: An overview of the F1TENTH Competition from [1].

While the system specifies the chassis, motors, and LiDAR sensor to be used, the components for the rest of the system (sensors, compute unit, etc.) are unrestricted. This means that F1TENTH is essentially a competition of algorithms and sensors – teams should not get an advantage simply from a bigger motor, for instance.

The competition has two component: a **time-trials** component where cars aim to complete the most consecutive laps around a pre-mapped track over a set duration, and a **head-to-head** component where two cars are pitted against each other on the same track, and the first car to complete a set number of laps wins.

The Center for Project-Based Learning (PBL) team, *ForzaETH*, has competed at the F1TENTH competitions at ICRA since 2022. The competition is a platform for development into high-performance, robust embedded sensors and algorithms. ForzaETH uses an autonomous driving stack structured similarly to the one in fig. 1.2.

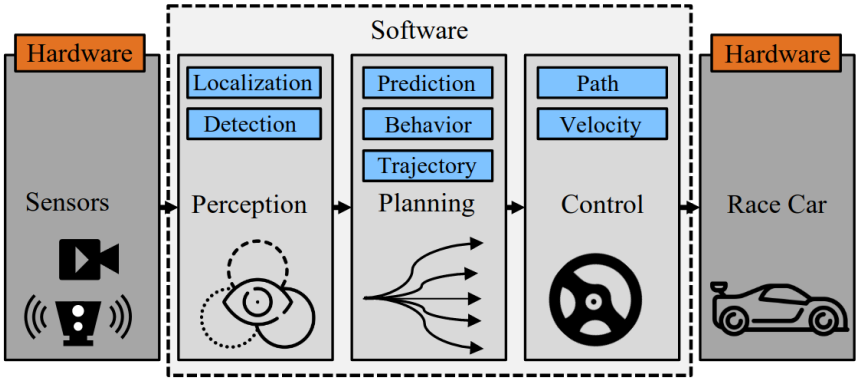

Figure 1.2.: Autonomous Driving Stack, from [2].

The software stack comprises three components:

1. **Perception** – Raw sensor inputs are fed into this block, and the state of the car and of other objects (for example other cars) are determined.
2. **Planning** – The car determines a high-level action to execute depending on its state relative to the other objects in the environment. For example, planning a trajectory to overtake an opponent.
3. **Control** – Based on the desired trajectory, actuator inputs are determined to execute it on the real system.

In this work, the **Localization** subsection of the Perception block was developed.

1.2. Motivation

Localization plays a key role in the performance of the car. It lies at the very start of the software stack, and therefore all subsequent sections of the stack are impacted by localization quality. Therefore, it is imperative to have a reliable, performant localization algorithm.

Previously, localization was done using a Simultaneous Localization and Mapping (SLAM) algorithm, *Cartographer* [3]. This worked well in good conditions, giving a high-frequency, smooth, and accurate pose estimate. However, at the F1TENTH competition at ICRA in 2022, the conditions proved challenging for localization.

Figure 1.3.: Track conditions at ICRA'22, with noisy LiDAR and odometry measurements.

The mentioned conditions are depicted in fig. 1.3. The black tubes used were absorbent to the lasers used by LiDAR sensors, resulting in noisy readings beyond short ranges. Furthermore, the floor was slippery, causing odometry measurements to be unreliable due to wheel-spin. These factors contributed to poor performance of Cartographer, where pose estimates would be inaccurate. This then led to crashes, as the wrong control inputs were applied.

1.3. Objective

The research objective of this thesis is therefore to determine the localization algorithm to be used in adverse environments.

This encompasses tweaking existing methods and investigating other algorithms. Subsequently, a comparison would then be made based on computation effort, localization performance, and robustness.

1.4. Outline

The thesis will be organized as follows: chapter 2 gives a review of localization methods in autonomous racing, covering the main two areas: SLAM and Monte-Carlo Localization (MCL) based methods. Subsequently, section 2.3 describes the algorithms chosen for further evaluation.

Chapter 3 details the operating principles of a Particle Filter (PF) for localization in 2D with odometry and LiDAR input. Chapter 4 details the implementation improvements and tweaks made on the PF.

Chapter 5 then details the qualitative and quantitative comparisons in different test scenarios between the algorithms under test, along with the results. These results are then discussed in the context of the research objective, along with qualitative comments, in chapter 6.

Lastly, chapter 7 concludes and details some areas for future work.

Chapter 2

Related Work

Research in robot localization is not new. However, autonomous racing requires the localization estimate to be both accurate and low-latency, which puts constraints on the algorithms used.

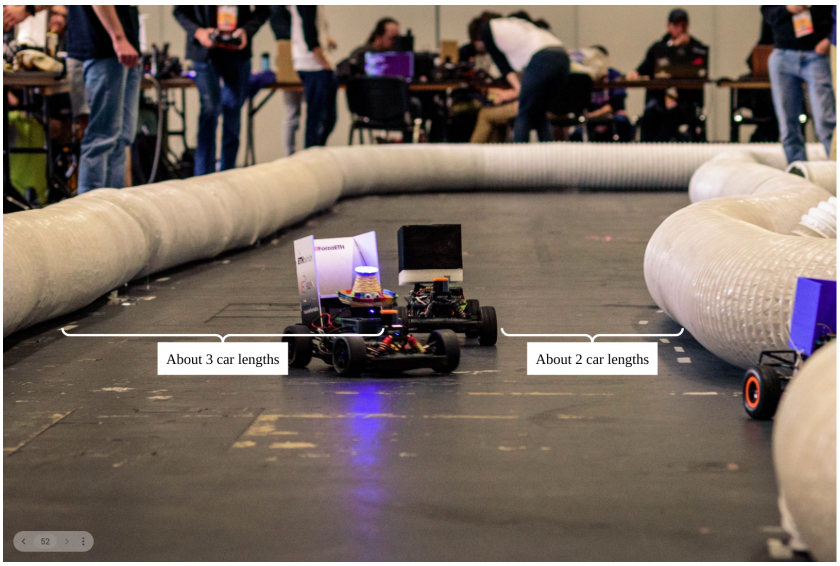

Figure 2.1.: An autonomous overtake by the ForzaETH car (white), illustrating the need for precise and high-frequency localization.

Figure 2.1 shows the ForzaETH car overtaking an opponent autonomously. There is not much space around the opponent, especially at high speeds. Laggy or inaccurate localization may result in a missed chance for an overtake, or worse, a crash, illustrating the stringent localization requirements in autonomous racing.

A thorough review of works in autonomous racing can be found in [2], and provides insight to the different localization methods possible. These fall broadly into two camps: SLAM-based methods, or MCL-based methods.

For an existing algorithm to be considered, it should be easy to integrate into the software stack. To that end, it should have bindings to Robot Operating System (ROS) [4], the software backend used in the race stack.

Furthermore, localization algorithms chosen must perform well with the available sensor modalities on the car: LiDAR-based range measurements and noisy Inertial Measurement Unit (IMU)-based odometry estimates.

2.1. SLAM based methods

SLAM refers to the process of simultaneously constructing a map (*mapping*) while keeping track of an agent's location within it (*localization*).

2.1.1. Odometry with scan-matching

The localization aspect is typically solved using *scan-matching*. Scan-matching algorithms take two LiDAR scans and attempt to match them with a rigid-body transform. This rigid-body transform is then used to obtain the robot's motion. Scan-matching algorithms include Iterative Closest Point (ICP), Normal Distributions Transform (NDT) [5], and optimization-based approaches.

2.1.2. Types of SLAM algorithm

The SLAM problem is typically solved with one of three approaches.

1. Kalman-Filter based approaches, eg. *Hector SLAM* [6]
2. Particle-Filter based approaches, eg. *FastSLAM* [7], *GMapping*[8]
3. Pose Graph based approaches, eg. *Cartographer* [3], *SLAM Toolbox* [9]

Of these methods, Pose Graph based approaches are the most mature, given their good integration with numerical solvers and their extensibility to large maps. Therefore, methods from this family will be chosen for this work.

As it is typically non-trivial to come up with effective, performant SLAM implementation, especially in the context of a semester thesis, an "off-the-shelf" implementation will be used.

2.1.3. Cartographer

Google *Cartographer* [3] is a SLAM implementation that uses an optimization-based approach to scan-matching. By implementing a unique branch-and-bound technique to compute scan-to-submap matches, it is able to map large spaces of tens of thousands of square meters in real-time. Furthermore, Cartographer has ROS integration.

2.1.4. SLAM Toolbox

Similar to Cartographer, *SLAM Toolbox* is a SLAM implementation with ROS bindings. It uses a different optimization-based scan-matching implementation. Unlike Cartographer, it is still under active development.

2.2. Monte-Carlo Localization based methods

MCL-based methods [10] are sampling-based methods. Such methods maintain a (large) number of hypotheses that represent possible states of the car. These hypothesis states are then propagated forward based on odometry information by a *motion model*, and the hypothesis states are evaluated based on a *sensor model*. Each hypothesis is also referred to as a *particle*, hence a MCL method may also be called a PF. Further details about how MCL methods work are in section 3.1.

A challenge in these methods is maintaining real-time computation with a large number of particles. Each flavor of MCL achieves this in different ways. Two notable examples are the *AMCL ROS Package* and the *MIT RACECAR Particle Filter*.

2.2.1. AMCL ROS Package

The Adaptive Monte-Carlo Localization (AMCL) ROS package [11] (hereafter referred to as "AMCL") implements a PF with an *adaptive* number of particles. The number of particles processed depends on the uncertainty of the position estimate. When the estimate is certain, fewer particles are required to represent the possible positions of the car. This saves computational effort.

2.2.2. MIT RACECAR Particle Filter

Another implementation of MCL was developed by the Massachusetts Institute of Technology (MIT) for their *RACECAR* project [12] (hereafter referred to as "MIT PF"). MIT PF offers `rangelibc`, a fast library for 2D ray-casting which includes GPU acceleration and a constant-time access lookup table method for CPUs. This accelerates the evaluation of the sensor model in MCL, allowing more particles to be maintained.

2.3. Methods Evaluated

To keep the number of comparisons between localization algorithms tractable, a preliminary filtering of algorithms was conducted. One algorithm each representing SLAM and MCL methods would be chosen.

For SLAM based methods, Cartographer was chosen. Cartographer and SLAM Toolbox both work in similar ways (optimization-based scan-matching, pose graph backend), so it would be expected that their performance, advantages, and drawbacks would be roughly similar. However, Cartographer was already well-integrated into the race stack.

For MCL based methods, neither AMCL nor MIT PF were chosen. AMCL was ruled out because it did not provide any method to accelerate the evaluation of the sensor model, compared to the `rangelibc` library offered by MIT PF. Furthermore, while useful for other areas of mobile robotics, the author argues that adaptive particle count is of not much use in autonomous racing.

As mentioned at the start of this chapter, accurate, high-frequency localization is required at all times in autonomous racing. This is regardless of the confidence of the localization estimate, and therefore regardless of the number of particles processed by AMCL. The computation unit should be able to handle scenarios of high uncertainty while running the rest of the race stack *at race pace*. Therefore, it is not *average* computation effort to be optimized, but rather *peak* effort.

On the other hand, MIT PF was observed to output "jittery" poses, illustrated in fig. 2.2. Here, `original_pose` refers to the Cartographer pose output, while `pf1` refers to the MIT PF pose output.

MIT PF's pose output is highly non-smooth, with its pose estimates occasionally doubling back on itself. Therefore, it was decided to implement a MCL method from the ground up, building on the `rangelibc` library for acceleration. This implementation is simply called *PF2*. The reason for jittery poses from MIT PF is detailed in section 6.2.4.

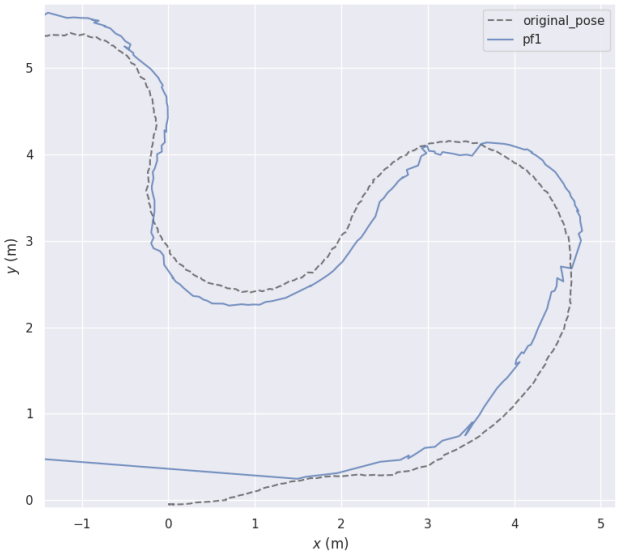

Figure 2.2.: An illustration of MIT PF's jerky pose output.

Chapter 3

Theory / Background

3.1. Monte Carlo Localization

In preparation to implement a PF algorithm, this chapter will focus on the theory behind the MCL algorithm, as detailed in *Probabilistic Robotics* [10].

Algorithm 1: Monte Carlo Localization Algorithm

Function $MCL(\mathcal{X}_{t-1}, u_t, z_t, M)$
  $\bar{\mathcal{X}}_t \leftarrow \emptyset$
  $\mathcal{X}_t \leftarrow \emptyset$
  `// Update proposal distribution with motion and sensor models`
  for $n \in N$ **do**
    $x_t^{[n]} \leftarrow$ `sample_motion_model`$(u_t, x_{t-1}^{[n]})$
    $w_t^{[n]} \leftarrow$ `sensor_model`$(z_t, x_t^{[n]}, M)$
    $\bar{\mathcal{X}}_t \leftarrow \bar{\mathcal{X}}_t + \langle x_t^{[n]}, w_t^{[n]} \rangle$
  end
  `// Perform resampling of particles`
  for $n \in N$ **do**
    draw i with probability $\propto w_t^{[i]}$
    $\mathcal{X}_t \leftarrow \mathcal{X}_t + x_t^{[i]}$
  end
  `// Inferred pose as weighted average of proposal distribution`
  $\hat{x}_t \leftarrow \sum_{i=0}^{M} x_t^{[i]} * w_t^{[i]}$
  return $\hat{x}_t$, $\mathcal{X}_t$
end

Algorithm 1 defines the MCL algorithm. `sample_motion_model` will be defined in section 3.2 and `sensor_model` will be defined in section 3.3.

A set of N particles, $\mathcal{X}_t = \{x_t^{[1]}, x_t^{[2]} \dots x_t^{[N]}\}$ is maintained at each timestep t. This approximates the distribution of the possible states of the car. Each particle represents a hypothesis of where the car may be. Hence, $x_t^{[n]} = \{x_{t,x}^{[n]}, x_{t,y}^{[n]}, x_{t,\theta}^{[n]}\}$ for the three degrees of freedom in the 2D plane.

When initialized, the set of particles $\mathcal{X}_0$ may be initialized randomly over the entire admissible state space, or they may be spread around some known initial pose.

In addition, MCL requires a map of the surroundings, M, specifying whether a location is occupied with an object. An example of a map representation is an *occupancy grid*.

Each iteration of MCL takes odometry (u_t) and LiDAR (z_t) inputs. A *proposal distribution*, $\bar{\mathcal{X}}_t$, is also maintained, containing the possible states $x_t^{[n]}$ computed from the motion model and their corresponding weights $w_t^{[n]}$ computed from the sensor model.

For the next iteration, $\mathcal{X}_t$ is obtained by resampling particles from the proposal distribution $\bar{\mathcal{X}}_t$. Each element of $\bar{\mathcal{X}}_t$ is drawn with replacement with probability proportional to the weight of that particular particle. In this way, particles that agree more with the sensor measurements have a higher chance of being propagated forward.

To obtain an inferred pose $\hat{x}_t$, the weighted average of the particles in $\bar{\mathcal{X}}_t$ is taken.

3.2. Sampling the motion model

With the motion model, each particle's state is advanced based on odometry data plus some noise. The exact implementation of the motion model determines how the noise is applied to the particle.

Two ways of implementing this motion model will be discussed here. The first way is a naive method implemented in MIT PF.

Algorithm 2: Naive Motion Model

Function *sample_motion_model*(u_t, x_{t-1})
  // u_t is the pose difference in the global frame from $t-1$ to t
  $x_t \leftarrow x_{t-1} + u_t$
  $x_{t,x} \leftarrow x_{t,x} + n_x \sim \mathcal{N}(0, \sigma_x^2)$
  $x_{t,y} \leftarrow x_{t,y} + n_y \sim \mathcal{N}(0, \sigma_y^2)$
  $x_{t,\theta} \leftarrow x_{t,\theta} + n_\theta \sim \mathcal{N}(0, \sigma_\theta^2)$
  return x_t
end

u_t is the pose difference in the global frame from $t-1$ to t. That is, $u_{t,x} = x_{t,x} - x_{t-1,x}$, $u_{t,y} = x_{t,y} - x_{t-1,y}$, and $u_{t,\theta} = \texttt{angle\_diff}(x_{t,\theta}, x_{t-1,\theta})$, where `angle_diff` is a function that computes the signed shortest distance between two angles.

Algorithm 2 moves each particle a distance based on odometry input. Subsequently, noise with fixed variances is added. The variances for each axis (σ_x, σ_y, σ_θ) are separate tunable parameters.

This algorithm captures the idea that odometry is unreliable. However, as the noise level is fixed regardless of the magnitude of motion, it can be improved.

Algorithm 3: Diff-Drive Motion Model

1 **Function** *sample_motion_model*(u_t, x_{t-1})
  `// ` u_t `is the pose difference in the global frame from` $t-1$ `to` t
  `// Calc motion primitive differences`
2 $\delta_{rot1} \leftarrow \text{atan2}(u_{t,y}, u_{t,x}) - x_{t-1,\theta}$
3 $\delta_{trans} \leftarrow \sqrt{u_{t,x}^2 + u_{t,y}^2}$
4 $\delta_{rot2} \leftarrow u_{t,\theta} - \delta_{rot1}$
  `// Calc variance for each motion primitive`
5 $\sigma_{rot1} \leftarrow \alpha_1 \delta_{rot1} + \alpha_2 \delta_{trans}$
6 $\sigma_{trans} \leftarrow \alpha_3 \delta_{trans} + \alpha_4 (\delta_{rot1} + \delta_{rot2})$
7 $\sigma_{rot2} \leftarrow \alpha_1 \delta_{rot2} + \alpha_2 \delta_{trans}$
  `// Add noise to each motion primitive`
8 $\hat{\delta}_{rot1} \leftarrow \delta_{rot1} + n_{rot1} \sim \mathcal{N}(0, \sigma_{rot1}^2)$
9 $\hat{\delta}_{trans} \leftarrow \delta_{trans} + n_{trans} \sim \mathcal{N}(0, \sigma_{trans}^2)$
10 $\hat{\delta}_{rot2} \leftarrow \delta_{rot2} + n_{rot2} \sim \mathcal{N}(0, \sigma_{rot2}^2)$
  `// Get result`
11 $x_{t,x} \leftarrow x_{t-1,x} + \hat{\delta}_{trans} \cos(x_{t-1,\theta} + \hat{\delta}_{rot1})$
12 $x_{t,y} \leftarrow x_{t-1,y} + \hat{\delta}_{trans} \sin(x_{t-1,\theta} + \hat{\delta}_{rot1})$
13 $x_{t,\theta} \leftarrow x_{t-1,\theta} + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$
14 **return** x_t
15 **end**

Algorithm 3 is an improvement to algorithm 2 by using a *differential-drive* assumption. Figure 3.1 shows a geometrical interpretation of the motion of a differential-drive robot. Each motion can be translated into three motion primitives: a rotation ($rot1$), a translation ($trans$), and another rotation ($rot2$). Intuitively, the larger the value of the motion primitives, the larger the injected uncertainty.

This is reflected in line 5 - 7 of algorithm 3. The variances for each motion primitive are scaled based on a weighted combination of the magnitude of corresponding motion primitives.

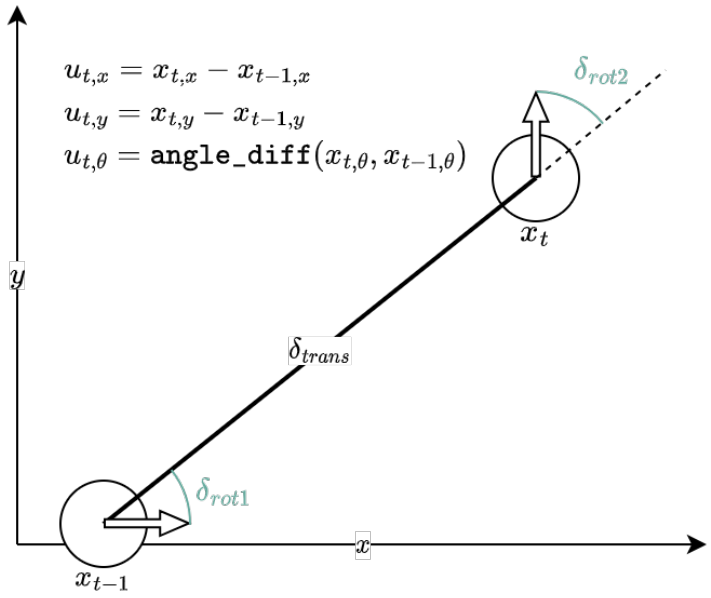

Figure 3.1.: Geometrical interpretation of noise sources in algorithm 3

For rotations, α_1 scales the rotation variance with the magnitude of the rotation, and α_2 scales it with the magnitude of the translation. This captures some idea of "covariance" between rotation and translation. For translations, α_3 scales the translation variance with the magnitude of the translation, and α_4 scales it with the magnitudes of both rotations. The values of α are tunable parameters.

While a differential drive robot model does not exactly match the Ackermann steering model found in the F1TENTH car, it has been applied successfully to other Ackermann vehicles in autonomous racing, for example in [13].

3.3. Evaluating the sensor model

The sensor model compares the LiDAR readings z_t with the readings that would be expected given the map M and a pose in the proposal distribution $x_t^{[n]}$. This is done for each beam of the LiDAR.

A diagram of laser scan operation is shown in fig. 3.2. In the figure, the LiDAR is represented as the white circle, while previously-mapped obstacles in the environment are the gray rectangles. An unmapped obstacle is shown with a yellow square.

The LiDAR takes a number of range measurements z_t^k around the robot, represented by the eight beams. Furthermore, to evaluate the likelihood of a measurement given a pose, expected measurements z_t^{k*} are calculated for each laser beam. In an occupancy grid

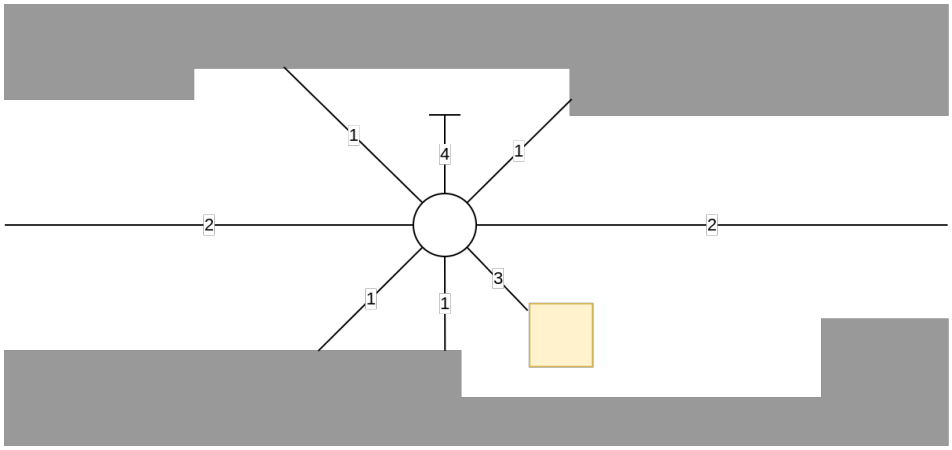

Figure 3.2.: Diagram of laser scan showing possible failure modes.

map, this can be done using *ray-casting*. Essentially, given a robot pose and beam angle, rays are extended from the corresponding position on the map until they collide with an obstacle. The distance of the ray is then the expected measurement.

Each measurement can belong to one of four categories, which are also labelled along the lines in the diagram.

1. **Correct Range**: A *hit* where the reading coincides with the distance of the beam to the obstacle.
2. **Max-Range**: A maximum-range reading, where any reading is at the maximum range z_{max} of the LiDAR.
3. **Unexpected Obstacles**: An unexpectedly short reading, where an unmapped obstacle was detected.
4. **Random Reading**: A random, garbage measurement within the maximum range of the sensor caused by sensor error.

Each of these four measurement cases are represented probabilistically with different probability distributions.

The **Correct Range** reading can be represented with a Gaussian distribution with tunable variance σ_{hit} centered around the expected reading z_t^{k*}. That is,

$$p_{hit}(z_t^k, z_t^{k*}) = \begin{cases} \eta_{hit}\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2), & \text{if } 0 \leq z_t^k \leq z_{max} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

η_{hit} is a normalizer such that $p_{hit}(z_t^k, z_t^{k*})$ is a valid probability distribution (sums to 1 over its support). It can be calculated numerically over the range of input values.

$$\eta_{hit} = (\int_0^{z_{max}} \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2) dz_t^k)^{-1} \tag{3.2}$$

The **Max-Range** readings can be modelled by a point-mass distribution at z_{max}.

$$p_{max}(z_t^k) = \begin{cases} 1, & \text{if } z_t^k = z_{max} \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

The **Unexpected Obstacle** readings result in a shorter-than expected reading. This can be modelled with an exponential distribution with parameter λ_{short}.

$$p_{short}(z_t^k, z_t^{k*}) = \begin{cases} \eta_{short}\lambda_{short} \exp(-\lambda_{short} z_t^k), & \text{if } 0 \leq z_t^k \leq z_t^{k*} \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

Like p_{hit}, these values need to be normalized with η_{short}. Note that the limits of integration are changed, as the density of p_{short} is 0 beyond z_t^{k*}.

$$\eta_{short} = (\int_0^{z_t^{k*}} \lambda_{short} \exp(-\lambda_{short} z_t^k) dz_t^k)^{-1} \tag{3.5}$$

Lastly, **Random Readings** can be modelled with a uniform distribution from 0 to z_{max}.

$$p_{random}(z_t^k) = \begin{cases} \frac{1}{z_{max}}, & \text{if } 0 \leq z_t^k < z_{max} \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

The four different distributions are then added and weighted. This thus gives the overall likelihood of the particle. The four constants z_{hit}, z_{max}, z_{short} and z_{random} should add up to 1 and represent the relative probabilities of each measurement occurring.

$$w_t^{[n]} = z_{hit} p_{hit}(z_t^k, z_t^{k*}) + z_{max} p_{max}(z_t^k) + z_{short} p_{short}(z_t^k, z_t^{k*}) + z_{random} p_{random}(z_t^k) \tag{3.7}$$

Chapter 4

Implementation Details of Particle Filter

The main implementation challenge with MCL algorithms is the number of particles. While it is desirable to have a larger number of particles such that there is a high chance that at least one particle in the particle set represents the state of the car, the number of particles cannot be increased indefinitely.

This is because the motion model and sensor model have to be run for each particle at each step of the algorithm, all while respecting the desired output frequency. Therefore, efficient implementations of the sensor and motion model are crucial for improving the performance of the algorithm.

To this end, other than the explicit programmatic improvements detailed below, all calculations over the set of particles were *vectorized* in the *Numpy Python* library. This allows calculations to be performed in parallel using Numpy's optimized routines. The source code can be found online.

4.1. Motion Model Improvements

As previously discussed in section 3.2, there are various possible motion model formulations. Two algorithms were discussed, a naive one with fixed noise (algorithm 2), and a more sophisticated one based on a differential drive model (algorithm 3).

Improving the motion model is useful because the noise applied to each particle more closely reflects uncertainties in real quantities associated with the car's motion. This improves the efficiency of each particle, as each particle matches up more closely with possible motions in real life. Hence, there are less likely to be "obviously wrong" particles in the particle set.

However, the diff-drive motion model still has some limitations, notably at high longitudinal velocities. This was highlighted by Stahl et. al. [13], which proposed an improvement to the model. As the authors are from the Technical University of Munich (TUM), this improved motion model is henceforth called "TUM".

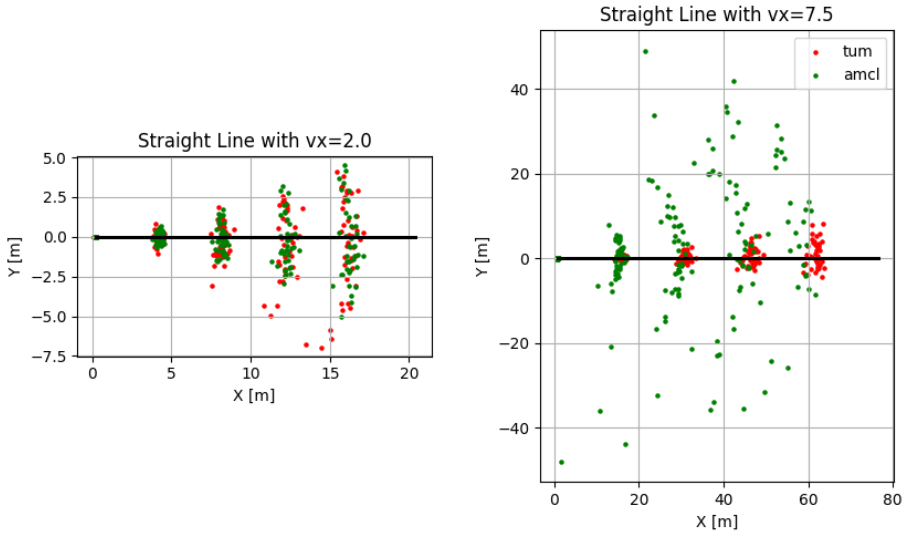

Figure 4.1.: A comparison of the diff-drive and *TUM* motion models.

Figure 4.1 illustrates this issue. Two trajectories are compared, one at a lower velocity of $2ms^{-1}$ on the left, and one at a higher $7.5ms^{-1}$ on the right. The spread of particles from different motion models is also compared, where particles from the diff-drive motion model (labelled `amcl`) are in green, and the improvement from TUM (labelled `tum`) are in red.

At high velocities, for instance when travelling down the straight sections of a race track, the car's angular uncertainty should not be very high, as the steering inputs would be relatively small. However, due to the terms $\alpha_2\delta_{trans}$ in line 5 and 7 of algorithm 3, the large positional displacement between adjacent poses results in a large value of σ_{rot}. This is reflected with the wide lateral spread of green particles in fig. 4.1. The majority of these particles are then obviously wrong, reducing the efficiency of particles.

The key improvement implemented is the replacement of line 5 and 7 of algorithm 3 from eq. (4.1) to eq. (4.2).

$$\sigma_{rot} \leftarrow \alpha_1\delta_{rot} + \alpha_2\delta_{trans} \tag{4.1}$$

$$\sigma_{rot} \leftarrow \alpha_1\delta_{rot} + \frac{\alpha_2}{\max(\delta_{trans}, \gamma_{th})} \tag{4.2}$$

The interpretation is then that angular uncertainty *decreases* as positional displacement increases. This reflects the intuition that it is much harder for a car to rapidly change its orientation as its linear velocity increases.

A scaling term, γ_{th}, is added as a threshold for this mechanism to become active in affecting the rotation variance. In addition, the variable name α_2 is kept, although its meaning is different.

4.2. Sensor Model Implementation

4.2.1. Boxed Lidar scan

The LiDAR used is a Hokuyo UST-10LX [14], which offers 0.25° angular resolution in a 270° field of view. This gives 1081 range measurements ("scanlines") to evaluate. Including more scanlines results in more information about the environment.

However, the sensor model has to be evaluated for each scanline, which quickly becomes computationally infeasible. To gain more information while keeping the number of scanlines constant, the orientation of the scanlines can be optimized.

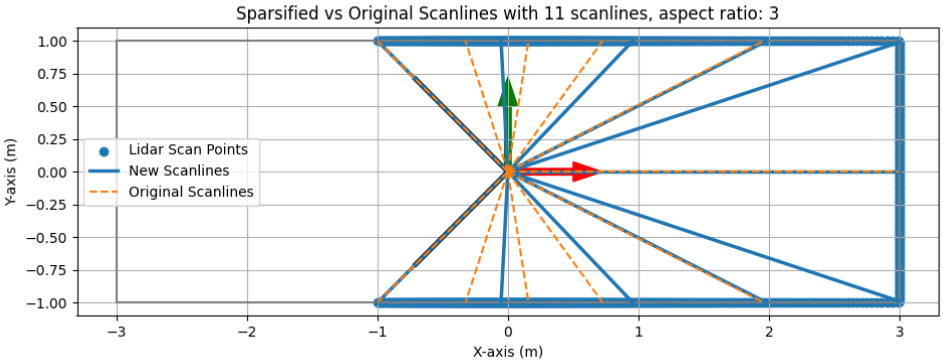

Figure 4.2.: Boxed vs Uniform Lidar patterns

Stahl et. al. [13] also proposed a *boxed* LiDAR layout, illustrated in fig. 4.2. The boxed scanlines are in solid blue, while the baseline uniformly-sampled scanlines are in dotted orange.

With a boxed layout, scanlines are chosen such that their intersections with a "corridor" of configurable aspect ratio are uniformly spaced. This is motivated by the observation that race tracks are corridor-like environments. By spacing the scanlines in this way, the boxed scanlines point further ahead down the racetrack, giving more information on

the geometry of the racetrack further ahead. This results in more information with a constant number of scanlines.

The aspect ratio is a tunable parameter. Having a low aspect ratio, as illustrated in fig. 4.3, does not yield much of a benefit over using uniformly-spaced scanlines. On the other hand, having an aspect ratio too high as illustrated in fig. 4.4 may discard scanlines that point towards the sides of the car, which would be important information to localize the car laterally. In practice, an aspect ratio of 3 to 3.5 is used.

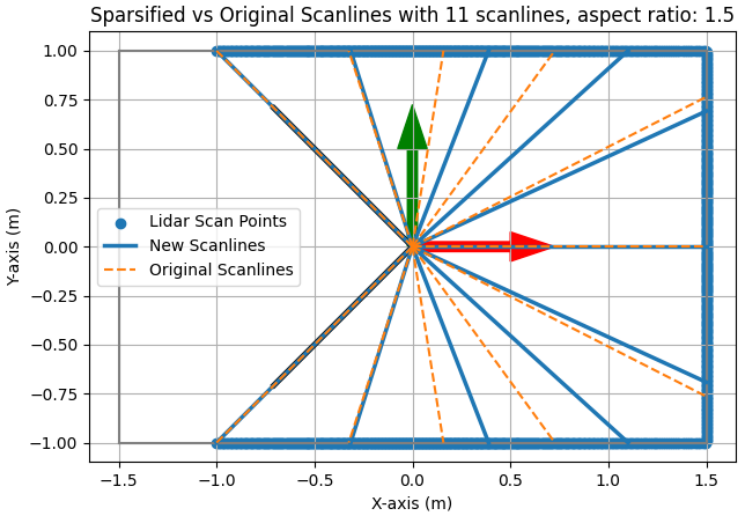

Figure 4.3.: Too-low aspect ratio in Boxed Lidar Implementation.

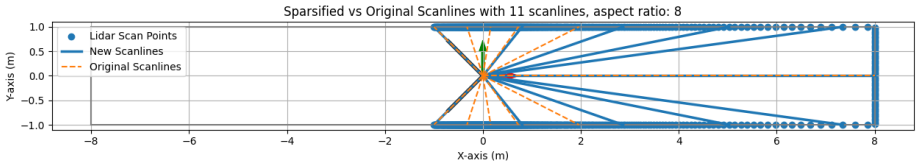

Figure 4.4.: Too-high aspect ratio in Boxed Lidar Implementation.

4.2.2. `rangelibc` Mode Selection

A large majority of the computation effort in MCL methods is evaluating the expected range at a given pose, z_t^{k*}. This is handled using the `rangelibc` [12] library. The library offers three modes of interest. Firstly, it allows the use of an available GPU to parallelize the ray-casting operation over all selected scanlines. This vastly speeds up the calculation of the expected ranges.

The second mode is called the Compressed Directional Distance Transform (CDDT), which is the authors' data structure to accelerate such queries to nearly constant time, regardless of the map size. This is unlike other algorithms like *ray-marching* and *Bresenham's line*, which scale linearly with map size.

Lastly, `rangelibc` offers a lookup table option to pre-calculate all expected ranges for a discretized set of poses in the map. A 3-D array is constructed, with entries for each possible x, y position of the LiDAR and orientation θ of the scanline. This increases the query speed to constant time at the expense of memory usage.

`rangelibc` Mode	Query Time (ms)	Memory Usage (MB)
Pruned CDDT	4.01	∼**95**
Lookup Table	**1.25**	∼160
Ray-Marching GPU	**0.59**	∼206

Table 4.1.: Timing and Memory characteristics of different `rangelibc` modes

Table 4.1 shows an evaluation of the computational requirements of the mentioned `rangelibc` modes. For evaluation, 21 scanlines were used. The author's laptop was used with an Intel i7-8750H CPU and Nvidia GTX1050Ti Mobile GPU.

A profiler was used to determine the average duration of the call to the sensor model over 1000 iterations. Memory usage was determined by observing the `htop` Linux utility.

The GPU-based method is almost **7x** faster than CPU-based Pruned CDDT method. In practice, the compute unit used in the F1TENTH car is a powerful CPU. In this case, the Lookup Table mode is **3.2x** faster while consuming a relatively small amount of memory. Therefore, the Lookup Table was used to evaluate the sensor model in this MCL implementation.

4.2.3. Pre-calculation of Measurement Probabilities

Similarly to the Lookup Table method above, the measurement probabilities detailed in eq. (3.1) to eq. (3.7) can be pre-calculated and looked up at run time.

A 2-D array P_{meas} is constructed with the expected ranges z_t^{k*} on one axis and the actual ranges z_t^k on the other. As the resolution of the expected range reading is linked

to the grid resolution M_{res} of the occupancy grid map, there is no noise added with this discretization. The array is then sized for distances corresponding to 0 to the maximum measurable distance, z_{max}. Furthermore, this discretization makes the computation of the normalizing terms η_{hit} in eq. (3.2) and η_{short} in eq. (3.5) convenient.

Equation (3.1) to 3.7 are then evaluated for each grid cell. Then, the resultant table is normalized over each value of z_t^{k*} to give a valid overall probability distribution, as detailed in the explanation of eq. (3.7). This algorithm is detailed in algorithm 4.

4.3. Calculation of empirical covariance

It is useful to compute the covariance of the inferred pose, either as a confidence metric or to feed downstream algorithms that require uncertainty estimates. The covariance Σ is calculated in eq. (4.3).

$$\Sigma \leftarrow \sum_{i=1}^{N} w_t^{[i]} s_t^{[i]} (s_t^{[i]})^T \tag{4.3}$$

Here, $w_t^{[i]}$ are the weights of each particle, and $s_t^{[i]}$ represent the deviation of the particle from the inferred pose. That is, the difference in x, y, θ.

Algorithm 4: Computation of measurement lookup table

Function *init_measurement_model()*
  $i_{max} \leftarrow z_{max}/M_{res}$ // M_{res} is the map resolution
  init$(P_{meas}, i_{max}, i_{max})$
  // Initialize P_{meas} as a square 2D array with side i_{max}
  init(η_{hit}, i_{max})
  init(η_{short}, i_{max})
  // Initialize arrays of length i_{max} to hold normalizer values at each z_{pred}
  // Compute normalizers
  for $i \in \{0, \dots i_{max}\}$ **do**
    $z_{pred} \leftarrow i * M_{res}$ // Expected distance measured (z_t^{k*})
    $s_{hit} \leftarrow 0$
    $s_{short} \leftarrow 0$
    for $j \in \{0, \dots p_{max}\}$ **do**
      $z_{meas} \leftarrow j * M_{res}$ // Actual distance measured (z_t^k)
      $s_{hit} \leftarrow s_{hit} + \mathcal{N}(z_{meas}; z_{hit}, \sigma_{hit}^2)$
      if $z_{meas} \leq z_{pred}$ **then**
        $s_{short} \leftarrow s_{short} + \lambda_{short} \exp(-\lambda_{short} z_{meas})$
      end
    end
    $\eta_{hit}[i] \leftarrow 1/s_{hit}$
    $\eta_{short}[i] \leftarrow 1/s_{short}$
  end
  for $i \in \{0, \dots i_{max}\}$ **do**
    $z_{pred} \leftarrow i * M_{res}$ // Expected distance measured (z_t^{k*})
    $n \leftarrow 0$ // normalizing sum
    for $j \in \{0, \dots p_{max}\}$ **do**
      $z_{meas} \leftarrow j * M_{res}$ // Actual distance measured (z_t^k)
      $p \leftarrow \eta_{hit}[i] z_{hit} p_{hit}(z_{meas}, z_{pred})$
      $p \leftarrow p + z_{max} p_{max}(z_{meas})$
      $p \leftarrow p + \eta_{short}[i] z_{short} p_{short}(z_{meas}, z_{pred})$
      $p \leftarrow p + z_{random} p_{random}(z_{meas})$
      $P_{meas}[i, j] \leftarrow p$
      $n \leftarrow n + p$
    end
    for $j \in \{0, \dots p_{max}\}$ **do**
      $P_{meas}[i, j] \leftarrow P_{meas}[i, j] \div n$ // normalize over the row
    end
  end
  return P_{meas}
end

Chapter 5

Results

5.1. Evaluation Metrics

To compare the performance of Cartographer vs PF2, qualitative and quantitative metrics were devised.

Firstly, **localization robustness** was to be determined *qualitatively*. This would be done by running the two algorithms offline on recorded data in various conditions, and visually inspecting the localization performance of the algorithms. This is possible because each localization algorithm has visually distinguishable failure cases. The quality of conditions would be varied to better answer the research objective in section 1.3.

Secondly, **localization accuracy** would be determined *quantitatively* by running each localization algorithm in-the-loop and comparing their fastest lap times. Lap time would be used as a proxy for localization accuracy because it was not possible to measure localization against a ground-truth directly. While it is an abstraction, both localization accuracy and lap time are factors in the speed of the vehicle, which is what matters at competition time.

Therefore, lap time is reported as a proxy for localization accuracy. In addition, the average lateral tracking error from the race line is reported, along with the 1-minute load average from the `htop` Linux utility.

5.2. Qualitative Localization Robustness

To determine localization failure, the two algorithms were run offline on recorded data. They were then visually inspected in *RViz*, a ROS graphical application. To be repre-

sentative, the data recordings were selected from laps where the car was running at a realistic race pace.

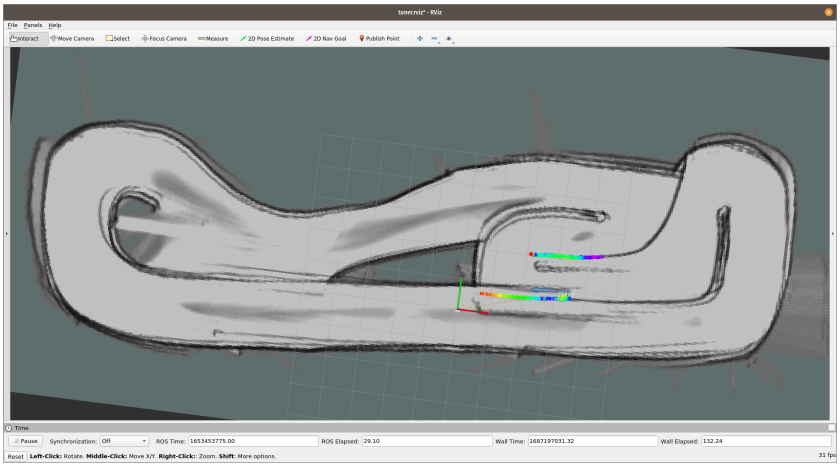

Figure 5.1.: An illustration of Cartographer localization failure. Pose estimate is the blue arrow and LiDAR scans are the rainbow dots.

Figure 5.1 shows an instance of Cartographer failing to localize. It can be seen that the map that Cartographer has internally has shifted, as the incoming scan and odometry data are unable to be matched to previously-mapped locations. The LiDAR scanlines also do not line up with the existing map, a sign of mis-localization.

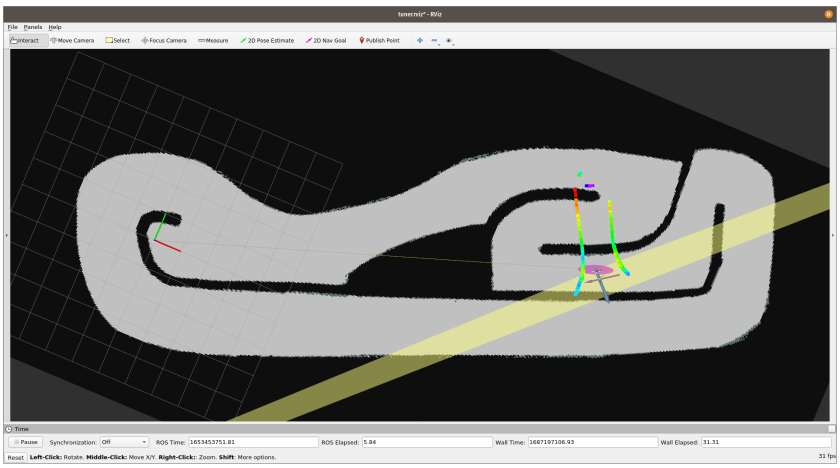

Figure 5.2.: An illustration of PF2 localization failure. Pose estimate is the blue arrow and LiDAR scans are the rainbow dots.

Figure 5.2 shows an instance of PF2 failing to localize. Not only is the angular covariance very large, the LiDAR scanlines also do not line up with the existing map.

5.2.1. Good Conditions (ICRA'23)

Firstly, the two algorithms were compared with conditions that were favorable for localization.

Figure 5.3.: Conditions at the F1TENTH Grand Prix at ICRA 2023.

Figure 5.3 shows conditions at the F1TENTH Grand Prix at ICRA'23. The track boundaries are made of white tubes, meaning that LiDAR can obtain low-noise distance measurements from large distances. Furthermore, the floor was made of grippy material, and thus odometry data was very reliable.

Results

Both Cartographer and PF2 are able to keep up in localization. They do not exhibit any signs of mis-localization.

Figure 5.4 shows trajectory histories for Cartographer (`icra23_slam`) and PF2 (`icra23_pf2`). The trajectory histories are slightly divergent, especially in the curves of the track. However, they exhibit similar smoothness properties in position.

5.2.2. Adverse Conditions (ICRA'22)

Next, the algorithms were compared in adverse conditions.

Figure 5.5 shows conditions at the F1TENTH Grand Prix at ICRA in 2022. As mentioned in section 1.2, the black tubes used were absorbent to the lasers used by LiDAR sensors, resulting in noisy readings beyond short ranges. Furthermore, the floor was slippery, causing odometry measurements to be unreliable due to wheel-spin.

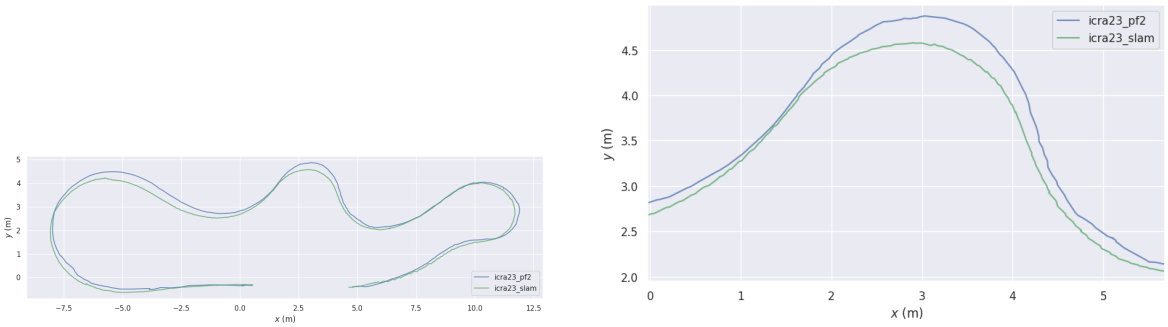

Figure 5.4.: Full (left) and zoomed (right) trajectory histories for Cartographer and PF2 on the ICRA'23 track.

Figure 5.5.: Conditions at the F1TENTH Grand Prix at ICRA 2022.

Results

Figure 5.6 shows the output map from Cartographer while running on ICRA'22 recorded data. The map quality is very poor, with Cartographer thinking that the straight section curves upwards and intersects the curvy section. This would clearly lead to fatal issues with the rest of the software stack if ran in-the-loop.

On the other hand, PF2 is able to cope with the poor measurements. A notable section of output is displayed in fig. 5.7. These screenshots are taken at 1 second intervals as the car approaches the end of the straight section.

Initially, the odometry readings have high variance, and the car is unable to localize itself longitudinally, even though the lateral localization is good. This is reflected in the long, purple oval in the first image, representing the particles' positional covariance.

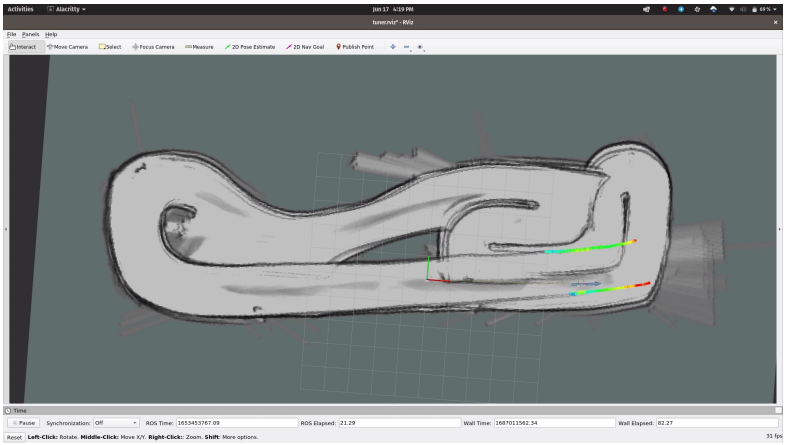

Figure 5.6.: Cartographer localization failure with ICRA'22 recorded data.

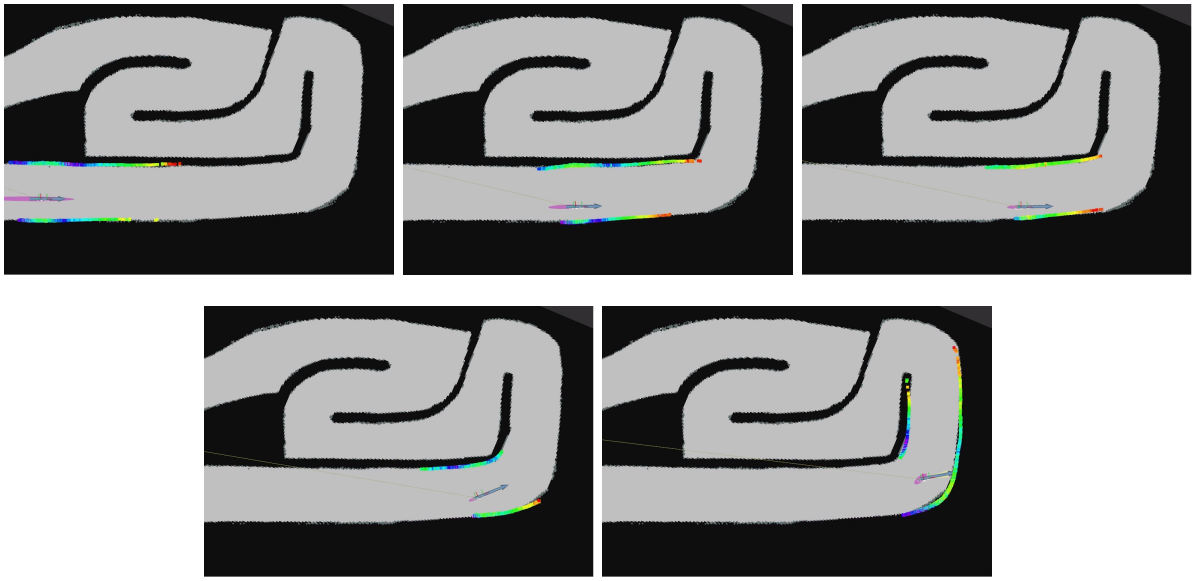

Figure 5.7.: Time evolution of Particle Filter reported pose with ICRA'22 recorded data. Left to right from top to bottom.

Subsequently, the car approaches the end of the straight. It turns out that due to the poor odometry, the car had not continued as far down the straight as it had thought. However, PF2 is robust to this. Based on the most current sensor measurements, the pose of the car does not move as much in the second and third image as the car corrects its pose estimate.

Subsequently, in the last two images, the car has detected the end of the straight and is able to localize itself well longitudinally and laterally. The LiDAR scan readings (rainbow dots) line up well with the map boundaries.

5.2.3. Poor Odometry Measurements (Test Track 1)

The above two recordings were conducted at two extremes of conditions – one almost perfect, while another extremely harsh. To have some balance in conditions, another lap was performed in a testing facility. The layout of this test track was similar to the track layout at ICRA 2022. This track will be referred to as *Test Track 1*.

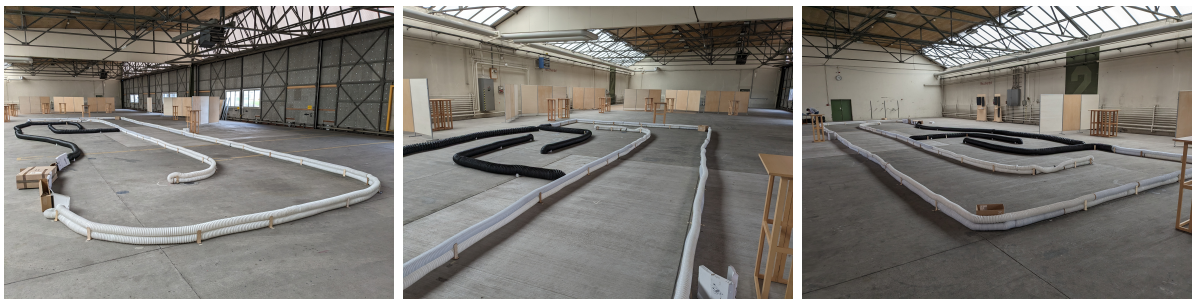

Figure 5.8.: Different views of Test Track 1.

The track would have white tubes in most sections, notably in the long straight where long-range LiDAR readings would be the most needed. Due to a shortage of white tubes, black tubes were used in the twisty sections, as the expected sensor ranges for those sections would be short. In essence, LiDAR scans for this test track were expected to be of good quality.

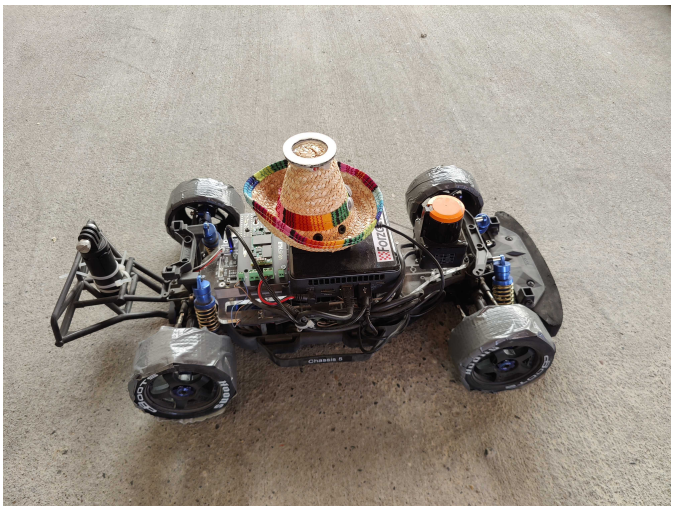

Figure 5.9.: Slippery tyre setup with taped tyres.

However, to investigate the effect of poor odometry, tape was applied to the tyres of the F1TENTH car to reduce the amount of available grip. The hope of this was to induce wheel-spin and simulate floor conditions that did not offer as much grip as in the testing facility. The taped tyres can be seen in fig. 5.9.

Results

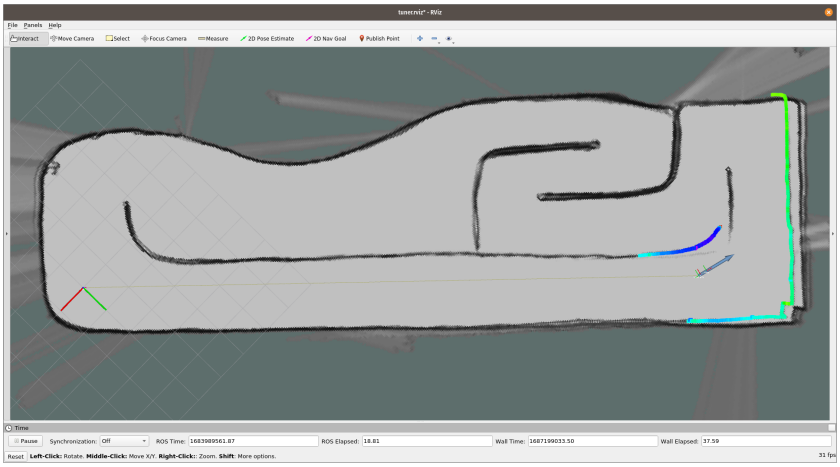

Figure 5.10.: Cartographer suffers a bit of map-shifting.

Figure 5.10 shows the output for Cartographer. While most of the map is consistent, the wall section near the end of the straight section has shifted. While this error is tolerable in this case, it could be more severe if, for example, the car was attempting an overtake and thought that there was more space than in reality.

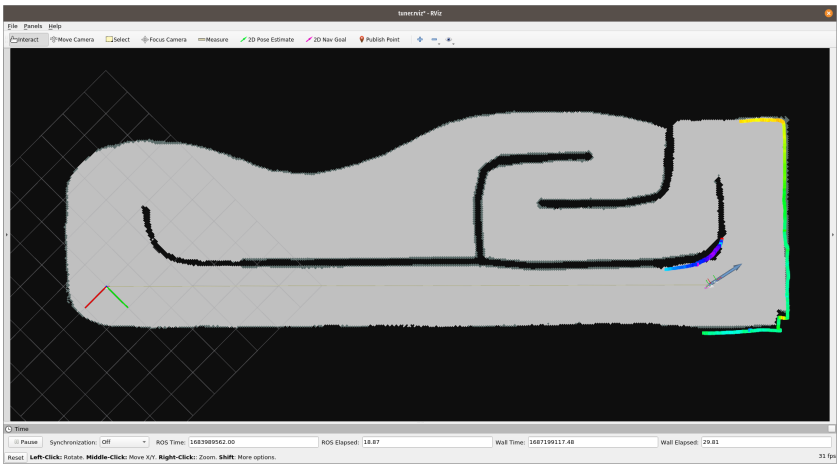

Figure 5.11.: PF fits the data well.

On the other hand, fig. 5.11 shows that in the same situation, PF2 is robust to the poor odometry.

Figure 5.12 shows trajectory histories for Cartographer (`hangar_slip_slam`) and PF2 (`hangar_slip_pf2`). Like the trajectories on the ICRA'23 track in fig. 5.4, the trajectory histories are slightly divergent, especially in the curves of the track.

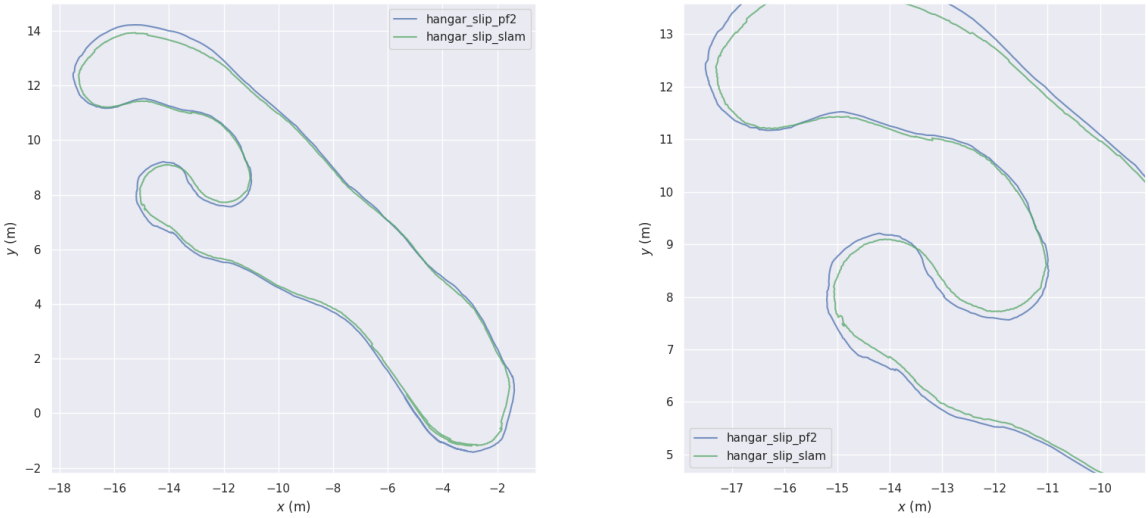

Figure 5.12.: Full (left) and zoomed (right) trajectory histories for Cartographer and PF2 on the track.

5.3. Quantitative Lap Time Analysis

Two tracks were set up in the testing facility with different characteristics, to test the two localization algorithms' performance in a variety of scenarios. For both test tracks, 10 flying laps were done with the average lap time taken.

5.3.1. Test Track 2 – Poor LiDAR, Poor Odometry

Test Track 2, shown in fig. 5.13, was an oval-shaped track. On the near side, black tubes were used. The intent of putting the black tubes at the end of a long straight section was to provide a challenging scenario for LiDAR detection, where long-range scans would be the most affected.

On the far side, the floor material was more slippery than in the grey concrete in the foreground. Along with the regular tyres used on the car, odometry from this setup was worse overall compared to Test Track 3.

Table 5.1 shows the lap time and compute results over 10 laps on Test Track 1. In this scenario, PF2 outperforms Cartographer.

Figure 5.13.: Left: A view of Test Track 2. Right: The regular tyres used on the car.

Table 5.1.: Lap Time and Compute results on Test Track 2

Localization Method	**Lap Time (s)**		**Average Lateral Error from raceline (m)**		`htop` **compute**
	Mean	Std Dev	Mean	Std Dev	
Cartographer	7.6680	0.0285	0.1592	0.0082	4.9
PF2	**7.4017**	0.0829	0.2172	0.0061	2.1

5.3.2. Test Track 3 – Poor LiDAR, Good Odometry

Figure 5.14 shows Test Track 3. It has a larger variety of turn types than Test Track 2. Black tubes are also placed at the end of straight sections, with similar rationale as Test Track 2. On the other hand, racing tyres were used on the car, which provide much more grip.

Table 5.2 shows the lap time and compute results over 10 laps on Test Track 3. In this scenario, Cartographer outperforms PF2.

Figure 5.14.: Left, Middle: Different views of Test Track 3. Right: The racing tyres used on the car.

Table 5.2.: Lap Time and Compute results on Test Track 2

Localization Method	**Lap Time (s)**		**Average Lateral Error from raceline (m)**		`htop` **compute**
	Mean	Std Dev	Mean	Std Dev	
Cartographer	**7.6453**	0.0675	0.078	0.0085	3.5
PF2	7.8858	0.1843	0.1565	0.0379	2.24

Chapter 6

Discussion

In this section, the Research Objective is answered. In addition, qualitative comments are made in comparing Cartographer and PF2.

6.1. Answering the Research Objective

As a reminder, the research objective in section 1.3 was: "Determine the localization algorithm to be used in adverse environments".

To cover different levels of "adversity", the experiments in chapter 5 can be broken into four categories. The more performant algorithm for each is also listed.

- good LiDAR, good odometry (eg ICRA'23 – fig. 5.3) : Cartographer
- good LiDAR, poor odometry (eg Test Track 1 – fig. 5.8) : PF2
- poor LiDAR, good odometry (eg Test Track 3 – fig. 5.14) : Cartographer
- poor LiDAR, poor odometry (eg. ICRA'22 – fig. 5.5) : PF2

Regardless of the quality of the LiDAR input, the quality of odometry decides between the better-performing algorithm. This shows that the **odometry quality** is more important that LiDAR quality in determining the difficulty of localization with Cartographer.

This is well illustrated with the data recording from ICRA'22 in section 5.2.2, where PF2 was able to overcome the large amount of wheel spin introduced by the slippery floor, while Cartographer was not.

However, it should be noted that Cartographer is more performant and consistent than PF2 in conditions where it can perform. For instance, in the results from Test Track 3 in table 5.2, Cartographer has a much lower Lap Time variance (0.0675s vs 0.1843 – **2.7x**

less) than PF2. This is linked to its ability to track the race line better, with a mean lateral error of 0.078m vs 0.1565m (**2x** better).

Hence, to answer the research objective: In a new racing environment, Cartographer should be trialled first. If it is unable to perform, PF2 is a reliable, robust backup option, especially in scenarios with poor odometry.

6.2. Qualitative Discussion

Answering the research objective was a binary decision between two algorithms. However, some qualitative aspects of the algorithm merit further discussion. While they may not affect the conclusion above, these factors are of interest when operating the algorithms in a race scenario, and perhaps for future extensions of the ForzaETH F1TENTH project. Of particular interest may be section 6.2.4, where the performance gap between PF2 and Cartographer in nominal conditions is discussed.

6.2.1. Computational Requirements

As noted in table 5.1 and table 5.2, the compute requirements for Cartographer are higher than that of PF2. These figures represent a 1-minute load average, during which the car was actively performing laps at race pace. While a more advanced profiler could give some further insight into the exact computational load for each algorithm without including other processes, this qualitative comparison still stands.

This point is particularly interesting because of the different compute units the ForzaETH team uses. In previous years of competition (eg ICRA'22), computational units were restricted, so the team used an Intel i3 processor. However, recently (ICRA'23), this restriction was lifted, and the team used a more powerful Intel i5 processor.

Due to the optimization-based nature of the scan-matching algorithm in Cartographer, obtaining localization estimates is not a constant-time process. Therefore, in more challenging environments where it is harder to find a minimum-cost solution, Cartographer's computational requirements *increase*. It has also been noted that the i3 processors suffer more from map-shifting when run on test tracks, especially with poorer sensor measurements coupled with running the full software stack.

Hence, in a scenario where computation is restricted, it is worth being aware of how Cartographer's computational requirements may affect its performance.

6.2.2. Sensitivity to Physical Map Changes

As discussed in section 6.1, Cartographer is more consistent than PF2 in nominal conditions, as it has lower variance in lap times and results in lower tracking errors to the race line.

However, it was observed at the ICRA'23 race that Cartographer is sensitive to physical changes in the map. These changes arise in the practice sessions of a competition, when many cars share the track. Cars often crash into the boundaries as teams try to find the limits, and if the track is not repaired often, these track limits often shift.

It is not surprising that Cartographer attempts to adapt to the changes in the map, as it is a mapping algorithm. However, this costs some reliability in localization, especially at high speeds.

Figure 6.1.: Comparison between Cartographer (purple) and PF2 (red) poses when map physically shifts.

Figure 6.1 shows a example of this. The purple arrow is the Cartographer pose estimate after the map had been physically changed in an off-screen area, while the red arrow is the PF2 pose estimate. While Cartographer erroneously reports that the car is in free space, PF2 correctly reports that a collision has occurred.

This scenario is only likely to occur in practice sessions during competitions, as in actual competition, the track will be free of most other cars. However, it is worthy to note, as team members will need to be vigilant in repairing the track when it is shifted if Cartographer is used.

6.2.3. Covariance Measurements

PF2 allows for the calculation of an empirical covariance through eq. (4.3). This is useful for downstream applications that require a covariance. An example of this is a *late-fusion* smoothing Kalman Filter described in [15] that fuses the localization pose estimate with a velocity estimate from odometry. Kalman Filters require a covariance estimate to weight the terms when fusing measurements from different sources, so having such an option is useful to improve the accuracy of such an estimate.

Another use-case of covariance measurements would be a "localization confidence" metric. This is described in [13], where the quality of localization is determined in part by the spread of the particle cloud. This would give an additional measure of autonomy and safety to the car, whereby it could take recovery actions in response to poor localization confidence, eg. driving slower or taking an alternative race line. This measure of localization confidence is not as straightforward to obtain in Cartographer.

6.2.4. Bridging the Performance Gap

Given that PF2 offers a number of qualitative benefits over Cartographer, it would be beneficial to improve its quantitative performance to match that of Cartographer. Some hypotheses of where PF2 lags behind are discussed here.

Pose Smoothness

PF2 outputs poses that are less smooth than Cartographer's. Figure 6.2 and fig. 6.3 show trajectories in individual axes for Test Track 1 (shown in fig. 5.8). Figure 6.4 and fig. 6.5 show trajectories in individual axes for the ICRA'23 track (shown in fig. 5.3). Blue lines are from PF2 and green lines are from Cartographer.

As can be seen, the outputs for PF2 are more jittery than Cartographer's. This is likely because the MCL algorithm does not place any knowledge on how the inferred poses from one time-step to the next vary. Therefore, the output poses end up jumping around. This ends up being bad for the rest of the rest of the software stack to follow.

This could probably be improved by incorporating some sort of smoothness prior to the particles, in addition to the likelihood scores from the sensor model. Alternatively, the highest-weighted particle could be used as the inferred pose, as each particle obeys the motion model.

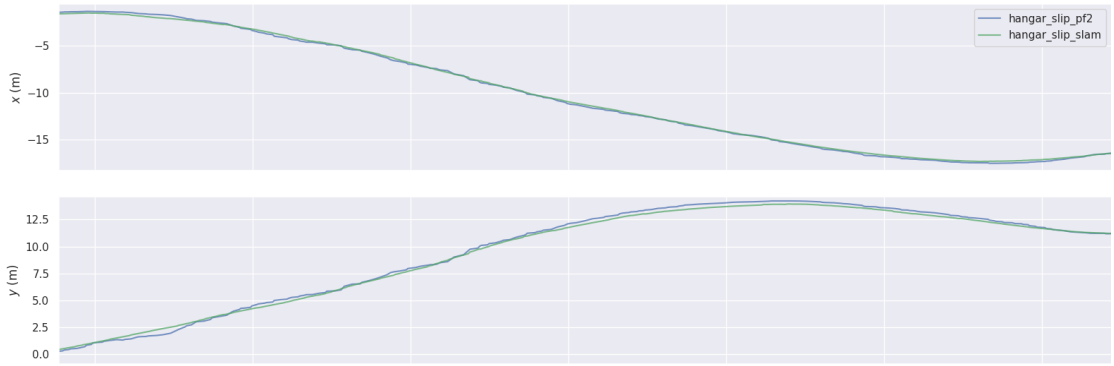

Figure 6.2.: Plots of Cartographer vs PF2 position output on Test Track 1.

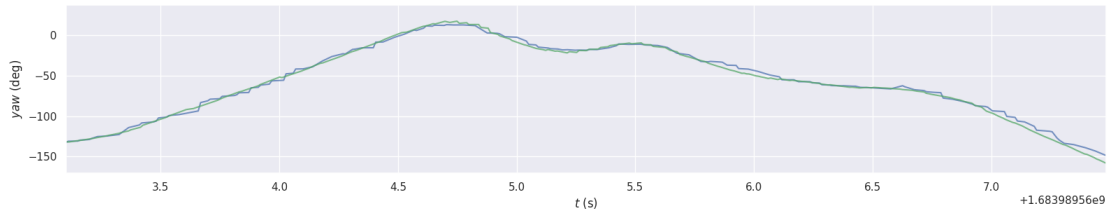

Figure 6.3.: Plots of Cartographer vs PF2 orientation output on Test Track 1.

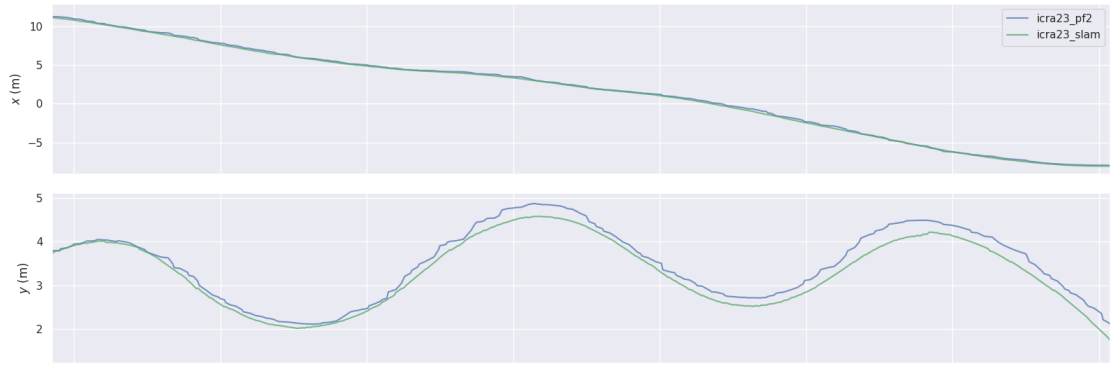

Figure 6.4.: Plots of Cartographer vs PF2 position output on the ICRA'23 track.

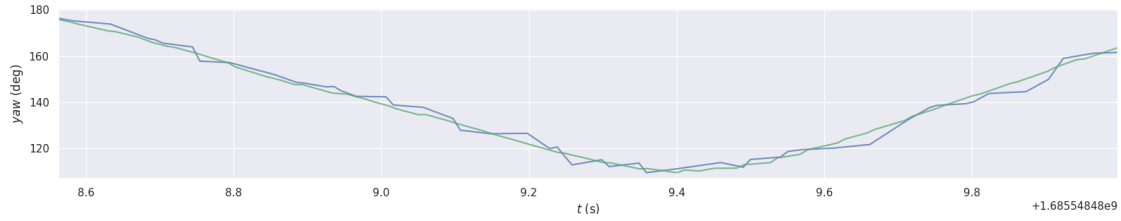

Figure 6.5.: Plots of Cartographer vs PF2 orientation output on the ICRA'23 track.

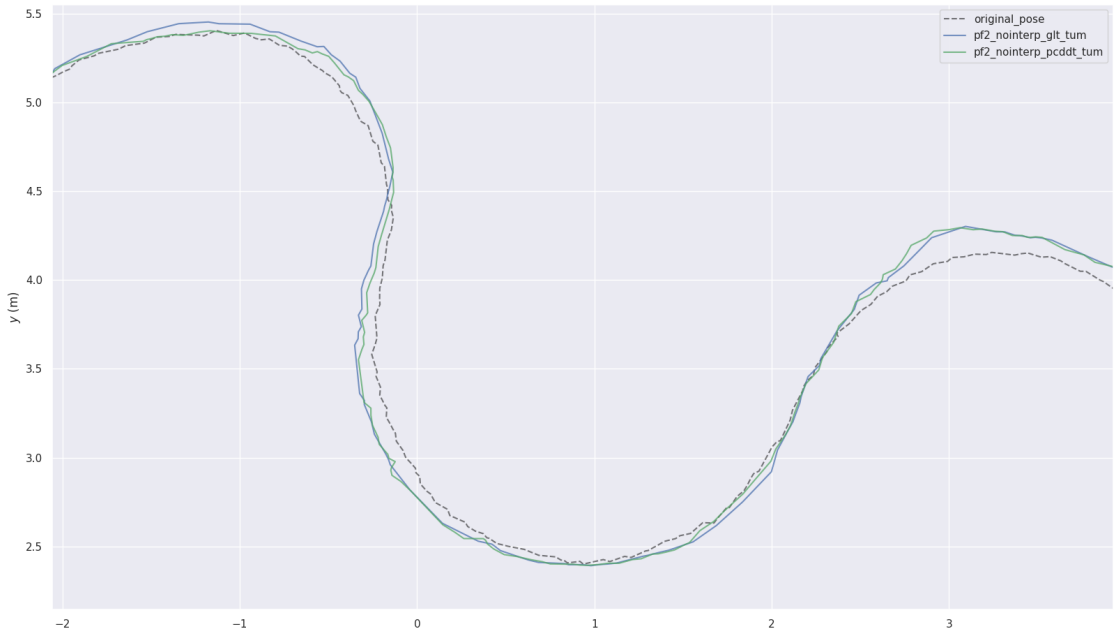

Figure 6.6.: Plots of Cartographer vs PF2 performance.

Localization Accuracy in Corners

Another probable cause of poor performance of PF2 could be because of its different localization estimates in corners.

Figure 6.6 shows the trajectory histories of Cartographer (dotted), PF2 run with Pruned CDDT (blue), and PF2 run with Lookup Table (green). In both, PF2 seems to lag Cartographer's pose output in the corners. This may be a source of PF2's worse performance in terms of tracking the race line.

This may be a result of several factors. Currently, each particle represents a hypothesis of the pose of the car's LiDAR sensor. However, it may be wrong to apply a motion model to these poses, as some motion models assume that the car moves around the location of its rear axle. Perhaps this implementation makes the motion model inaccurate, and places a majority of particles far away from the actual motion of the car.

Another possibility is that the sensor model is not discriminative enough and assigns high-enough probability to particles that are "close enough" in the map, leading to this inaccuracy.

Publish Frequency

Cartographer is able to publish a pose estimate at 200Hz. However, PF2's publish frequency is limited to the odometry input frequency of 50Hz. It was hypothesized that

a higher publishing frequency could improve the performance of PF2. However, this is unlikely, as the car's control stack also operates in the 40-50Hz range.

However, an attempt to improve publishing frequency by *dead-reckoning* revealed a possible source of jitteriness, especially in the MIT PF implementation. Dead reckoning works by selecting a fixed output frequency, and extrapolating pose estimates by integrating velocity from odometry when pose measurements are not available.

However, in the case of biased or poor odometry, it is possible that the dead-reckoned poses overshoot the actual distance travelled. This can be seen in fig. 6.7. Blue and green poses are with and without dead-reckoning, respectively. As can be seen, dead-reckoned poses are jittery as they travel too far forward in between sensor measurements, and 'snap back' to the correct positions when new measurements are obtained.

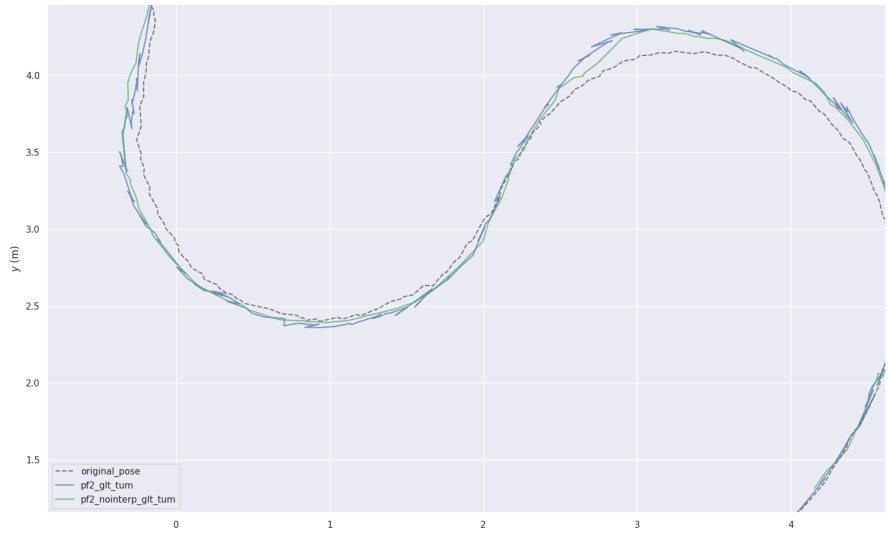

Figure 6.7.: Dead-Reckoning causing output jitteriness.

Chapter 7

Conclusion and Future Work

In conclusion, a performant and robust MCL algorithm, PF2, was developed and evaluated against the existing SLAM based localization method, Cartographer.

Qualitative and quantitative comparisons were done, varying the quality of LiDAR and odometry sensor input. It was found that Cartographer is especially reliant on good odometry input. On the other hand, PF2 is a robust alternative to Cartographer in challenging scenarios. However, Cartographer was found to be more consistent and performant than PF2 in cases where it could perform.

Some competition-specific, qualitative comparisons were also noted between PF2 and Cartographer. Particularly, section 6.2.4 details some hypotheses of the performance gap between the two.

Some ideas for future work are as follows.

Chiefly, an interesting direction is the continued investigation of a localization algorithm that is equally performant as Cartographer while having the positive characteristics (lower compute, higher robustness) of PF2. One option is investigating Pose Graph Optimization (PGO) methods that do not build a map.

Another option is to further improve PF2. The hypotheses in section 6.2.4 could be investigated. For instance, adding smoothness priors to PF2 to be evaluated in conjunction with sensor likelihood in order to induce a less jittery output.

In addition, given that some `rangelibc` modes are discretized (eg. Lookup Table), it would be useful to determine the discretization noise induced by such methods.

Lastly, it would be interesting to include some form of automatic parameter tuning of localization parameters to suit different track conditions, as the process is currently heavily based on operator intuition. Options like Bayesian Optimization could be considered in this case.

Appendix A

Task Description

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CENTER FOR PROJECT-BASED LEARNING

Task Description for a Semester Thesis on

Investigation of Localisation Techniques in Autonomous Racing in the Context of F1TENTH

at the Departement of Information Technology and Electrical Engineering

for

Tian Yi Lim
tialim@student.ethz.ch

Advisors:	Nicolas Baumann, nicolas.baumann@pbl.ee.ethz.ch Edoardo Ghignone, edoardo.ghignone@pbl.ee.ethz.ch
Professor:	Prof. Dr. Florian Dörfler, dorfler@ethz.ch
Handout Date:	14.03.2023
Due Date:	20.06.2023

Project Goals

The F1TENTH Autonomous Vehicle System is a powerful and versatile open-source platform for autonomous systems research and education on a 1:10 scale. The vehicle also defines the baseline for a competition that in the last years has been held during renowned robotics conventions (e.g. IROS, ICRA). The race consists of:

- A free practice slot, where the teams are allowed to map the environment and get used to the track.
- During the **Time trial - Qualification**, cars are challenged to race as quickly as possible, in a known environment without touching the walls. The fastest lap counts.
- During the **Head to head** race, two cars are challenged to race at the same time. The race is won by the car that completes 10 laps first. The previous qualification phase determines the racing bracket for a knock-out tournament.

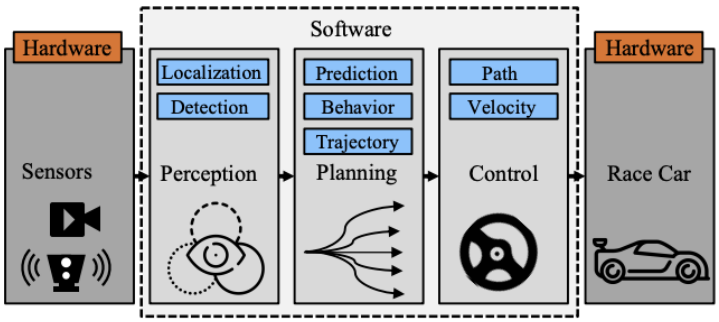

Figure 1: A classic autonomous driving stack including both hardware and software as well as the categorisation of perception, planning, and control from [1].

This thesis focuses on the localisation of the racing stack depicted Fig. 1. The current racing stack of the *ForzaETH* racing team utilises a graph-based Simultaneous Localisation and Mapping (SLAM) method (namely the *Google Cartographer* SLAM [2]) for its localisation task. This method has proven to be effective and performant [1], by supplying the subsequent planning and controls tasks with sufficiently accurate and low-latency localisation state information, enabling high performance racing capabilities.

However, it has been observed that when the track boundaries consist of highly light absorbent material (e.g. diffuse black tubes), the *Cartographer* SLAM method does not

[1] https://ee.ethz.ch/news-and-events/d-itet-news-channel/2022/08/forzaeth-team-wins-gold.html

perform well enough to ensure racing performance, as the range measurement qualities of the LiDAR sensor break down. Due to the sheer amount of tuning parameters and the intricacies of the algorithm, tuning *Cartographer* SLAM to the new surroundings is not always straightforward. If a race is being held under the aforementioned conditions, the current racing stack will not perform.

Therefore the goal of this thesis is to investigate different localisation algorithms in terms of localisation quality, computational efficiency and robustness to sensor quality. Examples to be investigated and compared to the current *Cartographer* SLAM technique could be: *MIT Racecar Particle Filter* [3] which is a commonly used localisation algorithm in F1TENTH [4]; Adaptive Monte Carlo Localization (AMCL) [5] a frequently used Particle Filter (PF) in the Robot Operating System (ROS) navigation stack [6]; other slam algorithms [7, 8].

Tasks

The project will be split into three phases, as described below:

Phase 1 (Week 1-4)

1. Study and get used to the F1TENTH environment.
2. Build a literature overview of possible localisation algorithms.
3. Define metrics (e.g. latency, accuracy) and experimental procedures to determine the racing performance to robustness to sensor quality (e.g. ill-reflective boundary material).

Phase 2 (Week 5-11)

1. Implement and integrate the previously identified localisation algorithms in the racing stack.
2. Perform the experiments with the previously identified metrics.
3. Evaluate the localisation algorithms based on the experimental outcome.

Phase 3 (Week 12-14)

1. Cleanup of code and documentation
2. Writing the report and preparing the presentation

Milestones

The following milestones need to be reached during the thesis:

- Identify localisation algorithms to test.
- Implement and integrate the different localisation algorithms into the race stack.
- Benchmark the different algorithms on the identified metrics.
- Evaluate the best-suiting localisation technique with respect to robustness to sensor quality (e.g. ill-reflective boundary material).
- Final report and presentation

Project Organization

During the thesis, students will gain experience in the independent solution of a technical-scientific problem by applying the acquired specialist and social skills. The grade is based on the following: Student effort, thoroughness and learning curve; Results in terms of quality and quantity; final presentation and report; documentation and reproducibility. All theses include an oral presentation, a written report and are graded. Before starting, the project must be registered in myStudies and all required documents need to be handed in for archiving by Center for Project Based Learning (PBL).

Weekly Report

There will be a weekly report/meeting held between the student and the assistants. The exact time and location of these meetings will be determined within the first week of the project in order to fit the students and the assistants' schedule. These meetings will be used to evaluate the status and document the progress of the project (required to be done by the student). Besides these regular meetings, additional meetings can be organized to address urgent issues as well. The weekly report, along with all other relevant documents (source code, datasheets, papers, etc), should be uploaded to a clouding service, such as Polybox and shared with the assistants.

Project Plan

Within the first month of the project, you will be asked to prepare a project plan. This plan should identify the tasks to be performed during the project and set deadlines for those tasks. The prepared plan will be a topic of discussion in the first week's meeting between you and your assistants. Note that the project plan should be updated constantly depending on the project's status.

Final Report and Paper

PDF copies of the final report written in English are to be turned in. Basic references will be provided by the supervisors by mail and at the meetings during the whole project, but the students are expected to add a considerable amount of their own literature research to the project ("state of the art").

Final Presentation

There will be a presentation (15 min presentation and 5 min Q&A for BT/ST and 20 min presentation and 10 min Q&A for MT) at the end of this project in order to present your results to a wider audience. The exact date will be determined towards the end of the work.

References

Will be provided by the supervisors by mail and at the meetings during the whole project.

Tran Yi Lin

Place and Date Zürich, 20/01/2023 Signature Student

Bibliography

[1] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022. [Online]. Available: https://doi.org/10.1109%2Fojits.2022.3181510

[2] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[3] C. Walsh and S. Karaman, "Cddt: Fast approximate 2d ray casting for accelerated localization," vol. abs/1705.01167, 2017. [Online]. Available: http://arxiv.org/abs/1705.01167

[4] "Mit racecar particle filter," https://github.com/mit-racecar/particle_filter, accessed: 2023-01-05.

[5] S. Xu and W. Chou, "An improved indoor localization method for mobile robot based on wifi fingerprint and amcl," in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, vol. 1, 2017, pp. 324–329.

[6] "Ros navigation stack amcl," http://wiki.ros.org/amcl, accessed: 2023-01-05.

[7] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[8] S. Macenski and I. Jambrecic, "SLAM toolbox: SLAM for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, May 2021. [Online]. Available: https://doi.org/10.21105/joss.02783

Appendix B

Declaration of Originality

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Investigation of Localisation Techniques in Autonomous Racing in the Context of F1TENTH

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Lim	TianYi

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 20/06/2023	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Bibliography

[1] "F1tenth - course documentation," https://f1tenth.org/learn.html, accessed: 2023-06-16.

[2] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022.

[3] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278.

[4] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[5] P. Biber and W. Strasser, "The normal distributions transform: a new approach to laser scan matching," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 3, 2003, pp. 2743–2748 vol.3.

[6] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[7] 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," *Journal of Machine Learning Research*, vol. 4, 05 2004.

[8] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[9] S. Macenski and I. Jambrecic, "Slam toolbox: Slam for the dynamic world," *Journal of Open Source Software*, vol. 6, no. 61, p. 2783, 2021. [Online]. Available: https://doi.org/10.21105/joss.02783

[10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[11] B. Gerkey, "amcl - ros wiki," 2023, [Online; accessed 19-June-2023]. [Online]. Available: http://wiki.ros.org/amcl

[12] C. H. Walsh and S. Karaman, "Cddt: Fast approximate 2d ray casting for accelerated localization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3677–3684.

[13] T. Stahl, A. Wischnewski, J. Betz, and M. Lienkamp, "Ros-based localization of a race vehicle at high-speed using lidar," *E3S Web of Conferences*, vol. 95, p. 04002, 01 2019.

[14] *Scanning Laser Range Finder Smart-URG mini UST-10LX Specification*, Hokuyo Automatic Co Ltd, 2015, rS-00554. [Online]. Available: https://www.hokuyo-aut.jp/dl/UST-10LX_Specification.pdf

[15] F. Massa, L. Bonamini, A. Settimi, L. Pallottino, and D. Caporale, "Lidar-based gnss denied localization for autonomous racing cars," *Sensors*, vol. 20, no. 14, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/14/3992