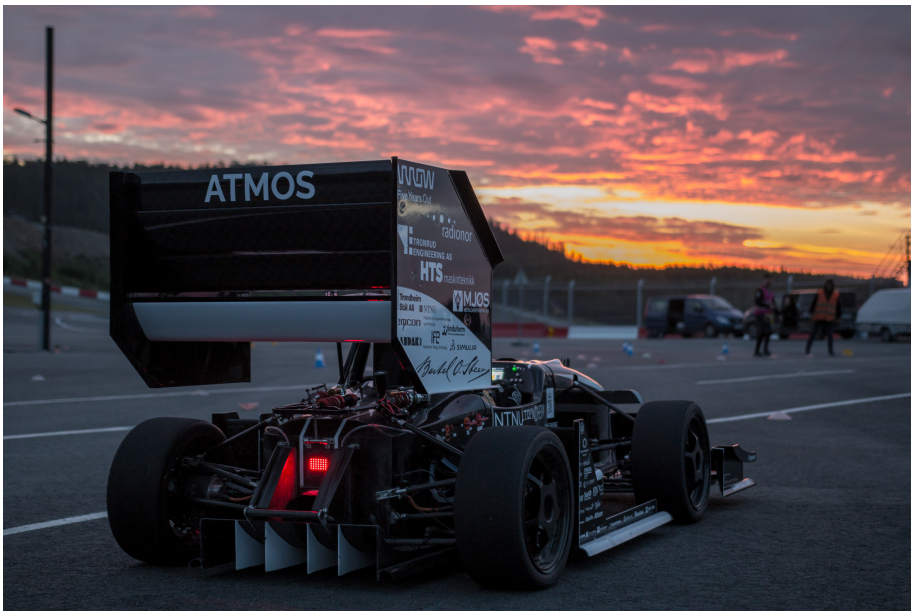
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Per Kvinnesland Omvik

Data Association and Simultaneous Localization and Mapping for an Autonomous Racecar

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke
June 2020



NTNU
Norwegian University of Science and Technology

revolve

Per Kvinnesland Omvik

Data Association and Simultaneous Localization and Mapping for an Autonomous Racecar

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU

Kunnskap for en bedre verden

Abstract

Autonomous racing is a relatively new addition to the field of robotics and autonomous mobility. Pioneered by the Formula Student (FS) competitions, it is a challenge tackled by student teams across the globe. As racetracks are limited in size and shape, racecars are constructed to perform at their best given these constraints. This principle is just as well applied to the development of software in an autonomous racecar.

Just as in regular racing, the involved systems in an autonomous racecar has to be able to react and make choices quickly based on external input. For this reason, the work presented in this thesis incorporates a state-of-the-art Simultaneous Localization and Mapping (SLAM) algorithm in iSAM2 to estimate the vehicle pose (position and orientation) as well as the locations of cones that make up the racetrack.

Based on measurements originating from visual sensors, this thesis goes into detail on the implementations of different means of data association in SLAM: the problem of associating measurements to cones. A total of four methods are considered, including the robust joint compatibility branch and bound (JCBB) algorithm, and tested in different scenarios relevant for FS competitions.

The proposed implementations allow the racecar to accurately build the map of cones and estimate its position on a winding track at speeds of at least 40 km/h. The introduction of probabilistic association schemes such as JCBB and maximum likelihood (ML)-data association allows the vehicle to better correct its *uncertain* initial pose within a given map, increasing the robustness of vehicle pose estimation.

Sammendrag

Autonom racing er et ganske nytt tilskudd til emnene robotikk og autonom mobilitet. Konseptet ble startet av Formula Student (FS)-konkurransene, og er en utfordring som studentlag over hele verden bryner seg på. Ettersom racerbaner er begrenset i størrelse og form, er racerbiler konstruert for å prestere på sitt beste gitt disse begrensningene. Dette prinsippet er like relevant for utviklingen av programvaren i en autonom racerbil.

Som i vanlig racing, må de involverte systemene i en autonom racerbil kunne reagere fort og ta hurtige valg basert på eksterne hendelser. Av den grunn innlemmer det arbeidet som er presentert i denne oppgaven en avansert Samtidig Lokalisering og Kartlegging (SLAM) algoritme i iSAM2 for å estimere kjøretøyets stilling, samt plasseringene til kjegler som utgjør racerbanen.

Gjennom bruk av målinger fra visuelle sensorer går denne avhandlingen i detalj om implementasjonene av forskjellige metoder for datatilknytning i SLAM: problemet med å knytte målinger til kjeglene. Totalt fire metoder ble vurdert, inkludert den robuste joint compatibility branch and bound (JCBB) algoritmen, og testet i forskjellige scenarier som er relevante for FS-konkurranser.

De foreslåtte implementasjonene lar racerbilen nøyaktig bygge kartet med kjegler og estimere sin posisjon på en svingete bane ved hastigheter på minst 40 km/t. Innføringen av sannsynlighetsbaserte datatilknytningsmetoder som JCBB og maksimal sannsynlighet (ML) lar kjøretøyet i større grad korrigere den *usikre* startstillingen sin på et gitt kart, noe som øker robustheten til estimeringen av bilens stilling.

Preface

This thesis concludes my time as a member of Revolve and as a student at NTNU. The previous year has been special in many ways, but I will especially appreciate the time I spent in Revolve and seeing my friends and classmates again after my year-long visit to the US.

The project I have been a part of in the last year was supposed to culminate in the participation in Formula Student events in Barcelona and Hockenheim this summer. I know that the team and I had big hopes for this year, but knowing the persistence and motivation of my fellow team members, I think 2021 is going to be exciting.

I give my sincere appreciations to all members of Revolve for their passion and interest, and especially the members of the Perception & Navigation group in the driverless project team for the good memories they have given me. Good luck next year! Also, thank you to my supervisors Edmund Brekke and Rudolf Mester for your feedback, guidance and interest throughout this project.

I want to thank Andrea for being a part of this journey with me, while also congratulating her for also becoming a Master of Science in Engineering Cybernetics. Thank you to my sisters and parents for your unconditional love and support.

Acronyms

$\sqrt{SAM}$ square root smoothing and mapping. 7

AWGN additive white Gaussian noise. 32, 80

BB branch and bound. 48

BoW bag of words. 42

CCOLAMD constrained COLAMD. 39

CDF cumulative distribution function. 22, 45

CG center of gravity. 29, 30

DAG directed acyclic graph. 25

DOF degrees of freedom. 30, 58, 77

DV driverless vehicle. 1, 54, 77, 101

EKF extended Kalman filter. 6, 33, 100, 101

EV electric vehicle. 11, 12, 101

FN false negative. 43

FP false positive. 43–45, 63

FS Formula Student. i, 1, 5, 9, 11, 12, 15, 43, 99

FSD Formula Student Driverless. 5, 11, 13, 100, 101

FSG Formula Student Germany. 5, 12, 77, 78, 87

GMRF Gaussian Markov random field. 34

GN Gauss-Newton. 36, 37, 71, 74

GNN global nearest neighbor. 43, 49

GTSAM Georgia Tech Smoothing and Mapping. 37, 57, 63, 65, 71, 72, 74, 85, 89

IC individual compatibility. 45–49, 64–67, 70, 85, 86, 90

ICP iterative closest point. 8

IMU inertial measurement unit. 54

INS inertial navigation system. 54

iSAM incremental smoothing and mapping. 7, 39, 49

iSAM2 incremental smoothing and mapping v2. 6–8, 34, 37, 39, 49, 64–66, 71, 75, 80, 99, 100

JC joint compatibility. 46–48, 67, 70, 90

JCBB joint compatibility branch and bound. i, 8, 48, 49, 67, 84, 85, 90, 94, 99, 100

JVC Jonker, Volgenant and Castanon. 62

LIFO last in, first out. 61

MAP maximum a posteriori. 6, 33–36, 38, 41, 44, 57, 58, 61, 62, 71

ML maximum likelihood. i, 8, 33, 44, 45, 49, 65, 67, 80, 81, 85, 86, 100

NLLSQ non-linear least squares. 36, 38

NN nearest neighbor. 43, 45, 46

PCL Point Cloud Library. 53

PDF probability density function. 6, 20, 21, 25–27, 32

RMSE root mean square error. 82

ROS Robot Operating System. 8, 53, 57–60

SCNN sequential compatibility nearest neighbor. 65, 85, 100

SLAM Simultaneous Localization and Mapping. i, 3–9, 13, 15, 26, 29–31, 33, 34, 39, 41, 42, 44, 49, 51, 53, 55, 57–59, 64, 69–71, 73, 74, 77–79, 82, 83, 89, 92, 99–101

SNN sequential nearest neighbor. 62, 63, 65, 69, 78, 79, 82, 86, 90, 94, 99, 100

SRIM square root information matrix. 37–39, 49

Contents

Chapter 1

Introduction

This thesis describes the author's work and contributions to the driverless vehicle (DV) team at Revolve NTNU.

Revolve NTNU is a voluntary, student-run organization at NTNU in Trondheim. Every year, students from several different fields of study come together to design and build an electrical racecar. Additionally, a Revolve-built car from a prior year is re-purposed to enable driverless operation. In the summer, both cars compete against other student teams from across the world at FS competitions organized in Europe. The competition winners are usually the teams with cars able to drive around the racetracks in the shortest amount of time.

Throughout the year, the DV team designs and implements autonomous software and hardware with a set of performance and competitive goals in mind. These goals are high-level and common for all vehicle systems. Consequently, they play a significant part in the problem formulation of this thesis, along with the decisions in design and application.

Figure 1.1: Revolve's driverless car Atmos driving on a typical Formula Student racetrack

1.1 Problem description

The specific performance goals are set by the organization at the beginning of the season (start of the academic year) and are based on the levels of ambition and current system capabilities. The goals are related to the *dynamic events* at the competitions and the updated goals from Revolve team 2019 (R19) to R20 are listed below:

- 4.3 s → 3.75 s on *Acceleration*
- 6.0 s → 5.5 s on *Skidpad*
- 5 m/s → 10 m/s avg on *Autocross*
- 10 m/s → 17 m/s avg on *Trackdrive*

In-depth explanations of the individual events are found in the chapter 2.

These goals, together with the constraints set by the competition rules and the organization itself, express the general task this thesis aims to solve: Design and

implement a SLAM system that allows Revolve's driverless car to drive as fast as possible. The SLAM module is responsible for iteratively mapping the stationary cones while concurrently localizing the car in relation to its surroundings. Cones constitute the racetrack delimiters, and the list of tracked cones will be referred to as a *map* throughout this thesis. As the involved systems need to act quickly and perform at high speeds, the module has to be designed with real-time operation and constraints are taken into consideration.

As the overarching problem is by design vaguely defined, an essential part of the project was to narrow the scope of the problem and identify focus areas that benefit performance in a significant way. This *concept* phase took place in the fall of 2019 and is documented in the author's project report [43]. In that time, it was decided that the SLAM *frontend* was to be developed further. Specifically, past members that were a part of the testing crew in the summer of 2019 suggested that the system was especially prone to errors during *loop closure*: When associating current spatial input data to a previously visited location. The suggested solution in [43] to this particular problem was to introduce a more robust method for *data association*. The reader is referred to chapter 5 for more details on this challenge.

Furthermore, as testing time was scarce in the summer, the team had mainly focused on optimizing the system for being able to build a *new* map for every time the system started up. This essentially meant that the possibility for the vehicle to use prior information was forfeited in exchange for *theoretically* being able to operate on any racetrack. As some of the events in the competitions (see section 2.2) have predetermined track layouts and also allow for multiple attempts, the ability to preload the track in memory is a valuable one. For one, the computational overhead is greatly reduced because neither track-finding or mapping is needed. In the case for SLAM, the problem is then reduced to *localization* only. Secondly, because the drivable area is known beforehand, the vehicle will not need to take the same precautions as when it explores unknown territory, resulting in higher speeds.

The demands of the SLAM system in Revolve's autonomous pipeline is finally summarized:

1. The design should allow the car to go as fast as possible. This is the overarching

goal.

2. The SLAM module should ideally keep track of all the cones in the track without introducing spurious cones.

3. The location and orientation of the vehicle should correctly be estimated using input from the *odometry* and *detection* modules.

4. The system should perform subject to real-time constraints such as *execution time* or computational resource usage.

5. Errors in loop closure should be minimized and preferably eliminated with a robust data association scheme.

6. The system should be able to localize the vehicle within a given racetrack to facilitate map preloading.

1.2 Project motivation

The voluntary student culture at NTNU is a proud tradition, and social groups have been a part of campus life for over a century. Although not a new concept, the student-run technical organizations such as Revolve, Ascend, Vortex and Propulse, all founded after 2013, have become a valuable part of what NTNU has to offer their students. In the case for Revolve, members range from being in the first year of their studies to being final year masters candidates. From the author's perspective, being a member of, and writing a thesis for Revolve carries a two-way benefit: The organization can apply the knowledge and experience of a fifth-year student to challenging tasks that demand time and research, and offers a unique and engaging experience in return.

Formula Student is one of the largest student engineering competitions in the world, with events organized in the USA, UK, Germany, Australia and more. Originally, competitions only included combustion engine vehicles, but later expanded to include electric-, and most recently, driverless racecars. Although the scope of SLAM in autonomous racing is constrained in the uniformity of tracks and environments, there

are some features of FS competitions that make the challenge unique. First of all, the SLAM module in a FS racecar is part of an extensive, interconnected system with components ranging from high-level software to high-powered electric motors. For this reason, the design has to be compliant with a complete, physical system operating in strenuous conditions. Moreover, the relative performance of the design, as part of the complete vehicle, compared to other teams is very tangible because of the shared ruleset and scoring in competitions.

1.3 Related work

Autonomous racing

Within the Revolve organization, there have been written two project reports [43][51], one bachelor thesis [16], and one master thesis [26] mainly or partially focusing on the SLAM module in the driverless pipeline. The subject of autonomous racing is rather specialized, and there is limited research published on SLAM and data association specific to the field. A notable entry is the published works of the Formula Student Driverless (FSD) team AMZ of ETH Zürich, where they present the concepts of their autonomous pipeline from 2018 [28] and 2019 [2]. In both seasons, they clinched the number one placement at Formula Student Germany (FSG). Another competition similar to FSD is Roborace, where in contrast to FS, the tracks are known to the traversing vehicle beforehand. Here, the focus lies on high speed localization and control at target speeds exceeding 150 km/h [53].

Simulataneous localization and mapping (SLAM)

Research on SLAM as a general problem is plentiful, and is a highly relevant topic in the impending age of autonomous mobility. The two-part introductory papers by Durrant-Whyte and Bailey [15][4] and the subject survey conducted by Cadena et al. [6] give substantial insight into the past, current and future state of the field. A common theme, as pointed out in the latter is that further development of recursive estimation techniques involving variants and improvements of the *Kalman filter* have mainly

been abandoned for batch optimization techniques based on maximum a posteriori (MAP) estimation.

Heavily motivated by the recursive pattern exhibited by the SLAM problem, the commonly proposed solutions for the problem used to involve *bayesian filtering*. The two main types of such filters used in SLAM are the aforementioned Kalman filter and *particle filters*. The latter iteratively estimates the posterior distribution by a set of weighted particles sampled from previous estimates. In contrast to Kalman filters, particle filters are not tied to assuming Gaussian models, as they can approximate any probability density function (PDF). Examples of particle filter-based methods include the works of Montemerlo et al. [39, 40] in FastSLAM 1.0 & 2.0.

Alternatively, implementations of the *extended Kalman filter (EKF)* and variations thereof, has long been the standard in SLAM applications in what is called EKF-SLAM [13, 52, 14]. The EKF linearizes the models describing motion and measurements at each time step, which are then used to predict and update the probabilistic models of the involved states. Thus, the EKF acts in an iterative manner, where the current state estimates are used as the linearization point, and is propagated as new information arrives. As a consequence, the *covariance matrix* maintained by the EKF will be dense, as all past information is marginalized and incorporated into the current estimates. This leads to a complete update of the system states and covariances at every iteration where new information is added, for which the computational effort is quadratic in the number of system states [15].

Batch estimation, long considered as an infeasible solution for on-line SLAM applications, refers to estimating the complete state trajectory using the full set of measurements [24]. In other words, batch estimation a form of *Bayes smoothing*, and aims to solve what is often referred to as *full SLAM problem*. Intuitively, batch estimation or smoothing appears to be the more demanding practice, as there is more data to process. Yet, in recent years, it is considered to be more capable, including in real-time applications, compared to recursive filters. State-of-the-art implementations of SLAM such as ORB-SLAM [41], LSD-SLAM [17] and iSAM2 [32] are examples of highly regarded batch estimation-based systems. For these methods, the most important trait of the smoothing covariance matrix is that it no longer has a dense

structure because states are not marginalized (removed) from the state vector. This fact allows for both incremental and batch updates of state and covariance estimates [11]. More importantly, the inverse of the covariance matrix, referred to as the *information matrix* in the literature, will be sparse. With the help of conscious state variable ordering, this sparsity can be used for efficiently finding optimal solutions. Chapter 4 in this thesis goes into more detail on this topic.

A relevant example of batch estimation in the realm of SLAM is the incremental smoothing and mapping v2 (iSAM2) optimizer introduced by Kaess et al. [31]. As given in its title, iSAM2 proposes an *incremental* solution to the SLAM smoothing problem. Comparing iSAM2 to its immediate predecessor in [33], the introduction of the *Bayes tree*, first presented in [30] improves real-time performance with fluid state linearization and independence of *periodic* batch updates of linearization points and variable ordering. Going even further back, the original version of incremental smoothing and mapping (iSAM) improved on the ideas from square root smoothing and mapping ($\sqrt{SAM}$) [11] by allowing incremental additions to the state vector without any considerable performance penalties. In the same article, the authors present a method for efficient recovery of uncertainty estimates from the *square root information matrix*. A collection of this work is compiled in [12].

Data association

The data association problem is nontrivial and essential in the field of SLAM. For any recursive or batch estimating SLAM system to operate, observations of the environment are necessarily assumed to be consistent with the autonomous agent's actual surroundings. The ability to accurately associate external measurements to elements of the environment, and thereby establish this consistency, is the essence of the data association problem in SLAM. Data association was previously a term mostly used in the target tracking domain, exemplified by Bar-Shalom et al. [5], and is where the *validation gating* approach to determining correspondences was introduced. Here, the *Mahalanobis distance* is used to quantify the similarity of measurements, with their associated uncertainties, with the probability distributions of target tracks. Neira

and Tardós [42] expand on this individual compatibility between measurements and environmental features by considering the internal correlations of measurements and the features from which they originate with the *Joint compatibility test*. Recent work concerned with data association in SLAM includes methods using variations of the JCBB algorithm from Neira and Tardós [42] with faster execution [37, 50]. Applying methods in *scan matching*, such as the iterative closest point (ICP) algorithm, has also been explored [48].

1.4 Contributions

This thesis builds upon ideas in the previous work on SLAM in Revolve by Engebretsen [16], Gustavsen [26] and [51]. The main contributions for this project is:

- The implementation of four methods for data association in SLAM for an autonomous racecar, including JCBB and ML-data association.
- Utilization of the iSAM2 SLAM backend for efficient, real-time uncertainty estimate recoveries for use in data association.
- A reworked *modular* frontend implementation which allows for low-effort modifications and additions to the data association schemes.
- A system capable of robust handling of asynchronous input in a realistic Robot Operating System (ROS)-based environment.
- Extended the existing capabilities of the Revolve SLAM module to include the ability to accurately localize the vehicle in a given map.
- In-depth descriptions of the integral parts of the implementation employed in Revolve, useful as a reference for future members of the organization.

1.5 Outline

The report is organized as follows. Chapter 2 gives some extra background information on Revolve and the FS competitions. In chapter 3, some elementary theory relating to reference frames, dynamic models, probability, and probabilistic graphical models is given. The two following chapters (4 and 5) go more in-depth on the structure of the SLAM problem and splits the theory into two parts: The frontend, concerned with data association, and the backend, which performs the batch estimation on the system states. Chapter 6 touches briefly on the systems in Revolves driverless software pipeline that supply the SLAM with data. A detailed description of the implementation of SLAM is given in chapter 7. Finally, results and conclusion with remarks on further works is found in chapters 8 and 9.

Chapter 2

Revolve and Formula Student

2.1 Revolve

After its establishment in 2010, Revolve NTNU first attended an FS competition in 2012 at Silverstone in the UK. While the first two iterations of vehicles produced housed combustion engines, every vehicle since 2014 has been powered by electrical motors.

For 2018, the organization retrofitted its previous electric vehicle (EV) Eld with new sensors and actuators, to compete in the FSD class, first introduced at FS Germany

in 2017. In the inaugural season for Revolve Driverless, the team secured 2nd position at FS East in Hungary solely based on static events performance, as the car was never cleared for participating in the dynamic events. A few weeks later, the team finished 7th at FSG after completing one of the ten laps in the *Trackdrive* event.

In 2019, as well as this year, the Revolve designed EV Atmos from 2018 was retrofitted with a visual sensor system to allow autonomous operation. After a promising testing season, Atmos experienced issues with a safety component during electrical scrutineering at the competitions. As a result, the 2019 Driverless team were unable to compete in any dynamic events. Sadly, no FS competitions will be held in the summer of 2020 due to the COVID-19 pandemic.

2.2 Dynamic events at FS competitions

The rules and events in FS competitions Revolve planned to attend in 2020 are set by FSG and listed in [19]. The competitions are split into two main parts: *static* and *dynamic* events. Static events do not focus on the operation and performance of the vehicle itself but are designed to test the students' knowledge about their systems and their ability to justify design choices. Dynamic events on the other hand, are all about the performance and safety of the vehicle itself. Before any car is placed on the track, extensive tests of the vehicle's low- and high-voltage electrics, hydraulic systems, structure integrity, internal state logic and general rule compliance all have to be passed. This is to ensure the safety of all participating teams and officials, and also to make sure that everyone is competing on a level playing field. When the car is cleared, it may participate in four scored dynamic events: acceleration, skidpad, autocross and trackdrive.

Acceleration

This event is a straight line race of 75 m where teams are scored on the elapsed time from a standing start to the finish line. At the start, the frontmost part of the car should be 0.3 m behind the starting line. In every event, including acceleration, blue and yellow cones make up the left and right sides of the track respectively, while large orange cones mark the start and finish lines. As per the rules, the minimum track width is 3 m and no two cones on one side are further than 5 m apart.

Skidpad

The layout of the skidpad track is displayed in figure 2.1. In this event, the car starts by entering the figure eight in the center. Then, it laps the right circle twice, in which the second lap is timed, before continuing around the left circle. Again, the second lap of the circle is timed. Lastly, the car exits the track on the opposite side of where it entered and comes to a stop. The average of the two recorded times is the base for the score on the event. Some unique aspects of the skidpad event are the fact that the number and location of cones in the inner and outer diameter of the circles are known, and that the track width is a constant 3 m. For SLAM, this means that no mapping is needed for this event, and localizing the car within the predetermined map is the principal task.

Autocross

The Autocross embodies the problem SLAM aims to solve. No prior knowledge about the map is allowed, and the vehicle has to build the map as it localizes itself within it. A run in this event is successfully completed when the car has completed one full lap and is aware of this fact. Thus, the car should stop by itself after the lap is completed. Each team is allotted at least two runs each, but data gathered from a previous run cannot be carried over to another. Judging from the scores of previous iterations of FSD competitions, the main focus in this event should be to finish without taking too many risks. This is also reflected in the dynamic goals listed in section 1.1. Some constraints on the track from the rules include no longer straights than 80 m, a minimum track

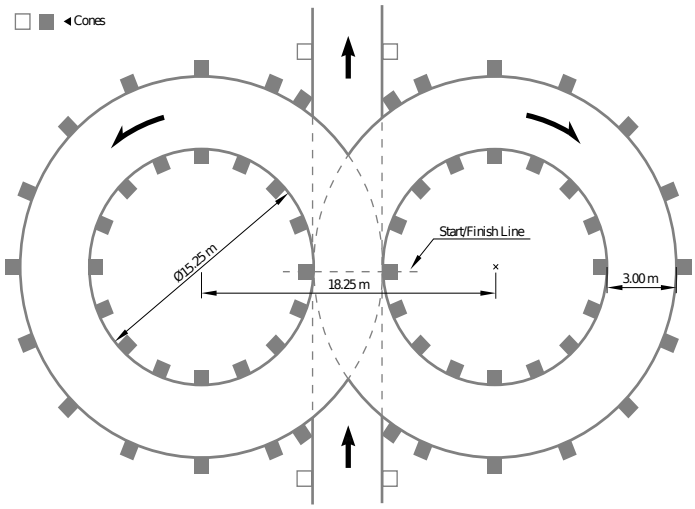


Figure 2.1: Skidpad track layout

width of 3 m and an approximate total lap length of 200 m to 500 m. The car also starts 6 m behind the timekeeping line which is indicated by a pair of large orange cones on each side.

Trackdrive

The trackdrive event takes place on the same track as autocross, only this time, a run consists of ten laps. In the rules regarding the trackdrive event, there is no statement saying that prior data cannot be used. This means that if the autocross event is held before trackdrive, and the saved map from the first event is of high quality, no mapping is needed. Hence, the car can drive a much faster first lap than it otherwise could have as the path is already set, and the confidence of cone locations is much higher. Either way, consequent laps will always be quicker than the mapping lap, which is also why the target average speed for trackdrive is considerably faster than for autocross (see sec. 1.1).

Chapter 3

Background theory

3.1 Reference frames

In the standard SLAM situation, an autonomous agent explores an environment where it is given the task of generating a map that is consistent with *stationary* surroundings. This is the case when mapping the interior and exterior of buildings, cities, road networks, etc. Because we consider all these stationary, we would like their mappings to be defined in an *inertial* frame of reference, meaning that the buildings, cities and roads do not move relative to an earth-fixed location. The location of the origin in this type of a coordinate system is in many cases arbitrary, and best chosen based on intuition or for simplicity. Where should the origin of a building be set, for instance? The same goes for the racetracks in FS competitions, where cones are defined in an earth-fixed inertial frame in which the origin is usually set as the starting location of the vehicle.

The case for all exteroceptive sensors such as lidars or cameras is that all output measurements are relative to the sensors themselves. If the location of a sensor is unknown relative to a predetermined inertial frame of reference, then the measurements cannot simply be expressed in this frame unambiguously. This is one of the problemsthat SLAM ultimately aims to solve.

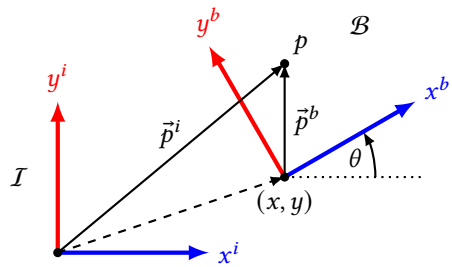


Figure 3.1: The point p is expressed in two different reference frames. The axes in the body frame $\mathcal{B}$ are shifted with (x, y) and rotated with θ in relation to the inertial frame $\mathcal{I}$

.

3.1.1 Coordinate systems

Reference frames, as discussed above, are accompanied with coordinate systems that allow unique descriptions of objects and their locations within the frames. The Cartesian and polar coordinate frames are the most commonly used when expressing two-dimensional geometry. The vector $\vec{p} \in \mathbb{R}^2$ is expressed equivalently in both systems:

$$\vec{p}_{cartesian} = \begin{pmatrix} x \\ y \end{pmatrix}, \quad \vec{p}_{polar} = \begin{pmatrix} r \\ \theta \end{pmatrix} \tag{3.1}$$

The connection between the variables is made in (3.2).

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix} \Leftrightarrow \begin{pmatrix} r \\ \theta \end{pmatrix} = \begin{pmatrix} \sqrt{x^2 + y^2} \\ \text{atan2}(y, x) \end{pmatrix} \tag{3.2}$$

3.1.2 Homogeneous transformations

The two reference frames $\mathcal{I}$ and $\mathcal{B}$, as shown in figure 3.1, are related through the counter-clockwise rotation θ and translation (x, y) of $\mathcal{B}$. The rotation of the x and y

axes from (x^i, y^i) to (x^b, y^b) is formulated by the *rotation matrix*:

$$\boldsymbol{R}_b^i(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{3.3}$$

The translation of the origin of $\mathcal{B}$ is denoted as $\vec{t}^i = (x, y)^\mathsf{T}$. Consequently, relating the vector $\vec{p}$ between $\mathcal{I}$ and $\mathcal{B}$ is straightforward:

$$\vec{p}^i = \boldsymbol{R}_b^i(\theta)\vec{p}^b + \vec{t}^i \tag{3.4}$$

In computer vision and robotics there is motivation in augmenting point coordinates by introducing a third entry z', generating *homogenous coordinates*. Tilde ($\sim$) notation is often used to signify vectors in the homogeneous coordinate space:

$$\tilde{\vec{p}} = \begin{pmatrix} \vec{p}' \\ z' \end{pmatrix} \in \mathbb{P}^2, \tag{3.5}$$

and is defined in the *projective plane*. Projective points are determined in Euclidean space as

$$\vec{p} = \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x'/z' \\ y'/z' \end{pmatrix} \in \mathbb{R}^2. \tag{3.6}$$

Motivated by how cameras and vision perceive depth (objects appear smaller as they are further away), this notation allows the variable z' to be used for representing points at infinity using finite coordinates by setting $z' = 0$. Furthermore, in camera-modeling, homogeneous coordinates enable the intrinsic and extrinsic parameters of cameras and setups to be directly applied to the linear projective transformations that map 3D world coordinates onto the 2D image plane [54].

In robot applications, it is desired to know or have the ability to estimate the relative positions of on-board sensors and actuators. As previously discussed, the location and orientation of the robot in an inertial frame of reference are also of interest. The *pose* of a robot in 2D space is defined by the compound vector $\vec{x} = (x, y, \theta)^\mathsf{T}$, and thus the

frame of reference of this robot is equivalent to $\mathcal{B}$. By defining the augmented matrix,

$$\boldsymbol{T}_b^i = \boldsymbol{T}(\vec{x}) = \left(\begin{array}{c|c} \boldsymbol{R}_b^i(\theta) & \vec{t}^{\,i} \\ \hline \vec{0}^{\mathsf{T}} & 1 \end{array}\right), \tag{3.7}$$

the transformation in (3.4) is condensed by setting

$$\tilde{\vec{p}}^{\,i} = \boldsymbol{T}_b^i \tilde{\vec{p}}^{\,b}, \tag{3.8}$$

with the homogeneous coordinates $\tilde{\vec{p}}^{\,i}$ and $\tilde{\vec{p}}^{\,b}$ in the form of (3.5) with $z' = 1$. As long as the conversion in (3.6) holds, the last coordinate in the homogeneous vector can be set arbitrarily. That being said, common practice suggests that this coordinate is set equal to one. The added benefit of this structure is that any number of transformations can be applied consecutively by simple matrix multiplication. As an example, in figure 3.2, assuming that the transformation matrices $\boldsymbol{T}_b^i$ and $\boldsymbol{T}_{l1}^b$ are known, the transformation from $\mathcal{L}_1$ to $\mathcal{I}$ is found by:

$$\boldsymbol{T}_{l1}^i = \boldsymbol{T}_b^i \boldsymbol{T}_{l1}^b \tag{3.9}$$

To extend this notation to robot or sensor poses directly, we can define the *compose* operation on a pair of poses as

$$\vec{x}_{l1}^{\,i} = \vec{x}_b^{\,i} \oplus \vec{x}_{l1}^{\,b}. \tag{3.10}$$

The pose $\vec{x}_{l1}^{\,i}$ is equivalent to

$$\vec{x}_{l1}^{\,i} = \vec{x}(\boldsymbol{T}_{l1}^i) = \begin{pmatrix} \boldsymbol{T}_{l1,13}^i \\ \boldsymbol{T}_{l1,23}^i \\ \operatorname{atan2}(\boldsymbol{T}_{l1,11}^i, \boldsymbol{T}_{l1,21}^i) \end{pmatrix}. \tag{3.11}$$

Transformation matrices on this form describe *rigid body displacements* and com-

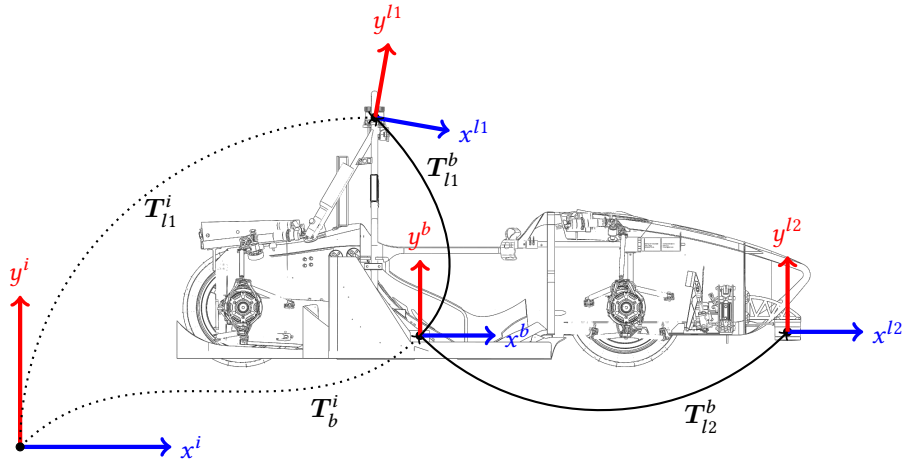


Figure 3.2: Transformations relevant for the car. Dotted lines suggest that the transformations are not constant. Used for illustration: $\mathcal{L}_1$ and $\mathcal{L}_2$ are the frames of the two lidar sensors. $\mathcal{B}$ is the body frame of the vehicle.

prise the *special Euclidean group* $SE(2)$. Formally, $SE(2)$ is defined as

$$SE(2) = \left\{ \boldsymbol{T} \,\middle|\, \boldsymbol{T} = \left(\begin{array}{c|c} \boldsymbol{R} & \vec{t} \\ \hline \vec{0}^{\mathsf{T}} & 1 \end{array} \right), \boldsymbol{R} \in \mathbb{R}^{2\times 2}, \vec{t} \in \mathbb{R}^{2} \right\} \tag{3.12}$$

The two dimensional rotation matrix $\boldsymbol{R}$ satisfy the criteria of the *special orthogonal group* $SO(2)$:

$$SO(2) = \left\{ \boldsymbol{R} \,\middle|\, \boldsymbol{R} \in \mathbb{R}^{2\times 2},\ \boldsymbol{R}^{\mathsf{T}}\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^{\mathsf{T}} = \boldsymbol{I}^{2\times 2}, |\boldsymbol{R}| = 1 \right\} \tag{3.13}$$

The $SE(2)$ group can be show to be *algebraic* [3], meaning that the inverse exist and allows the reversing of the order in (3.9) to find

$$\boldsymbol{T}_{l2}^{b} = (\boldsymbol{T}_{b}^{i})^{-1}\boldsymbol{T}_{l1}^{i} \tag{3.14}$$

The special Euclidean group is a subgroup of the *affine* group of transformations only where scaling, shearing and reflections are not applied.

The concepts of homogeneous coordinates and transformations can be extended to three dimensions. Vectors are simply extended to $\tilde{\vec{p}} = (x, y, z, 1)^\mathsf{T}$, while the rotation matrices are subject to a larger change. The counter-clockwise rotations of a reference frame $\mathcal{F}$ in relation to an inertial frame $\mathcal{I}$ about the x, y, z-axes are quantified as ϕ, θ, ψ respectively. The 3D rotation matrix applied to the coordinate frame, expressed with rotations about the individual axes is given as follows:

$$\boldsymbol{R}_f^i = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_x(\phi) \tag{3.15}$$

Because the work in this thesis is applied in the two-dimensional plane, no further discussion of three-dimensional motion will be supplied. Additionally, all coordinates are assumed to be homogeneous with the scaling variable set to 1. The implicit omission of this variable is also done to improve readability.

3.2 Probability theory

3.2.1 Random vectors

The random vector $\vec{X} = (X_1, \ldots, X_n)^\mathsf{T}$ is associated with a *joint PDF* $p_{\vec{X}}(x_1, \ldots, x_n)$, where the joint probability of the vector being sampled as any n-dimensional point $\vec{x} = (x_1, \ldots, x_n)^\mathsf{T}$ within the domain $\mathcal{A}$, is specified by

$$P(X \in \mathcal{A}) = \int_{\mathcal{A}} p_{\vec{X}}(x_1, \ldots, x_n) dx_1 \ldots dx_n. \tag{3.16}$$

For the PDF of a subset of the elements in the vector, the *marginal* distribution is found by integrating the joint PDF over all other variables in the vector not part of this subset. The *expected value*, or mean of $\vec{X}$ is the n-dimensional vector

$$\mathbb{E}[\vec{X}] = (\mathbb{E}[X_1], \ldots, \mathbb{E}[X_n])^\mathsf{T}, \tag{3.17}$$

with *variance* of $\vec{X}$ being the $n \times n$ matrix

$$\Sigma_{XX} = Var[\vec{X}] = \mathbb{E}[(\vec{X} - \mathbb{E}[\vec{X}])(\vec{X} - \mathbb{E}[\vec{X}])^{\mathsf{T}}]. \tag{3.18}$$

The (i, j) element of Σ_{XX} is the covariance between the pair: (X_i, X_j). When introducing a second n-dimensional random vector $\vec{Y} = (Y_1, \ldots, Y_n)^{\mathsf{T}}$, the cross-covariance matrix involving $\vec{X}$ and $\vec{Y}$ is defined:

$$\Sigma_{XY} = Cov(\vec{X}, \vec{Y}) = \mathbb{E}[(\vec{X} - \mathbb{E}[\vec{X}])(\vec{Y} - \mathbb{E}[\vec{Y}])^{\mathsf{T}}]. \tag{3.19}$$

3.2.2 Normal distribution

Throughout the work outlined in this thesis, the normal distribution is considered when approximating PDFs for stochastic quantities. Many phenomena in nature closely follow the normal (or Gaussian) distribution, and it is therefore commonly employed in the field of probabilistic robotics[55]. In the multidimensional case, normally distributed random variables are modeled as vectors that constitute the sample space of the n-dimensional multivariate normal distribution:

$$p_{\vec{X}}(x_1, \ldots, x_n) = \frac{\exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^{\mathsf{T}}\Sigma^{-1}(\vec{x} - \vec{\mu})\right)}{\sqrt{(2\pi)^k\,|\Sigma|}} \tag{3.20}$$

Noting that normal distributions are parameterized by their means and covariances, a compact notation for the normal random vector $\vec{X}$ is written

$$\vec{X} \sim \mathcal{N}(\vec{\mu}, \Sigma). \tag{3.21}$$

A convenient feature of multivariate normal distributions is that finding the marginal densities of the elements of a random vector, is just a matter of dropping the rows and columns containing the other variables from the joint covariance matrix and the corresponding rows from the mean[56]. As an example, the n-dimensional

random vector $\vec{X}$ contains three subvectors:

$$\vec{X} = \begin{pmatrix} \vec{X}_1 \\ \vec{X}_2 \\ \vec{X}_3 \end{pmatrix} \sim \mathcal{N}(\vec{\mu}, \Sigma),$$

with

$$\vec{\mu} = \begin{pmatrix} \vec{\mu}_1 \\ \vec{\mu}_2 \\ \vec{\mu}_3 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} & \Sigma_{13} \\ \Sigma_{21} & \Sigma_{22} & \Sigma_{23} \\ \Sigma_{31} & \Sigma_{32} & \Sigma_{33} \end{pmatrix}.$$

The marginal distribution $p_{X_1,X_3}(x_1, x_3)$ is normally distributed and parameterized as $\mathcal{N}(\vec{\mu}_{1,3}, \Sigma_{1,3})$, with

$$\vec{\mu}_{1,3} = \begin{pmatrix} \vec{\mu}_1 \\ \vec{\mu}_3 \end{pmatrix}, \quad \Sigma_{1,3} = \begin{pmatrix} \Sigma_{11} & \Sigma_{13} \\ \Sigma_{31} & \Sigma_{33} \end{pmatrix}. \tag{3.22}$$

3.2.3 Mahalanobis distance

The *Mahalanobis distance* is a distance measure weighted by the covariance matrix Σ. The similarity between sample points $\vec{x}$ and distributions with mean $\vec{\mu}$ and covariance Σ is expressed through the Mahalanobis distance function

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^\top \Sigma^{-1} (\vec{x} - \vec{\mu})}. \tag{3.23}$$

For a normal distribution with mean $\vec{\mu} \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n \times n}$ the Mahalanobis distance metric is *Chi-squared* distributed with n degrees of freedom (χ^2_n). The probability p for $D_M(\vec{x})$ to be smaller than some critical value c, is defined by the cumulative distribution function (CDF) and *Gamma function* $\Gamma(\cdot)$:

$$p = P_{\chi^2_n}(X < c) = \int_0^c \frac{t^{(n-2)/2} e^{-t/2}}{2^{n/2}\Gamma(n/2)}\, dt. \tag{3.24}$$

The inverse calculation $c = \{c : P_{\chi^2_n}(X < c) = p\}$ has no closed form solution when $n \neq 2$, and is found using approximation or lookup tables [18].

3.3 Dynamic models

A dynamic model describes a system of states and their behavior over time. Using *state-space representation*, dynamic models are described by a state vector $\vec{x} \in \mathbb{R}^n$, a vector of input variables $\vec{u} \in \mathbb{R}^m$, a vector of output variables $\vec{z} \in \mathbb{R}^d$, and a set of matrices relating the variables. The system, input and output matrices are $\boldsymbol{A}_{n\times n}$, $\boldsymbol{B}_{n\times m}$ and $\boldsymbol{C}_{d\times n}$ respectively, leading to the *discrete time-variant* linear state-space model at time step k:

$$\vec{x}_{k+1} = \boldsymbol{A}_k\vec{x}_k + \boldsymbol{B}_k\vec{u}_k \tag{3.25}$$

$$\vec{z}_k = \boldsymbol{C}_k\vec{x}_k. \tag{3.26}$$

This is a special case of the dynamic model, where $\boldsymbol{f} : \mathbb{R}^{n+m} \to \mathbb{R}^n$ and $\boldsymbol{h} : \mathbb{R}^n \to \mathbb{R}^d$ may be nonlinear:

$$\vec{x}_{k+1} = \boldsymbol{f}(\vec{x}_k, \vec{u}_k) \tag{3.27}$$

$$\vec{z}_k = \boldsymbol{h}(\vec{x}_k) \tag{3.28}$$

As working with linear systems is principally preferred over their nonlinear counterparts, the *first order Taylor expansion* of (3.27) create a linear approximation around the points $\vec{x}^*_k, \vec{u}^*_k$:

$$\vec{x}_{k+1} = f(\vec{x}_k, \vec{u}_k) \approx f(\vec{x}^*_k, \vec{u}^*_k) + \boldsymbol{J}_x(\vec{x}_k - \vec{x}^*_k) + \boldsymbol{J}_u(\vec{u}_k - \vec{u}^*_k) \tag{3.29}$$

where $\boldsymbol{J}_x$ and $\boldsymbol{J}_u$ are the *Jacobians* of $\boldsymbol{f}$ wrt. $\vec{x}$ and $\vec{u}$ given by

$$\boldsymbol{J}_x = \frac{\partial \boldsymbol{f}}{\partial \vec{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}, \quad \boldsymbol{J}_u = \frac{\partial \boldsymbol{f}}{\partial \vec{u}} = \begin{pmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial u_m} \end{pmatrix}, \tag{3.30}$$

evaluated at the linearization points.

3.4 Uncertainty propagation

If the state vector $\vec{x}$ from the previous section is not directly known, but is associated with some uncertainty it is treated as a random vector. The uncertainty of $\vec{x}$ is quantified by the covariance matrix Σ_{xx}. From (3.26), $\vec{z}$ and $\vec{x}$ are related through $\boldsymbol{C}$, and accordingly, the variance matrix of $\vec{z}$ is found:

$$Var(\vec{z}) = \Sigma_{yy} = \boldsymbol{C}\Sigma_{xx}\boldsymbol{C}^{\mathsf{T}} \tag{3.31}$$

By linearizing the nonlinear function $\boldsymbol{h}(\cdot)$, again using the first order Taylor expansion about the mean $\vec{x}_k^*$:

$$\vec{z}_k = \boldsymbol{h}(\vec{x}_k) \approx \boldsymbol{h}(\vec{x}_k^*) + \boldsymbol{H}_x(\vec{x}_k - \vec{x}_k^*), \tag{3.32}$$

a similar result can be found for the nonlinear relation in (3.28):

$$Var(z) = \Sigma_{zz} = \boldsymbol{H}_x\Sigma_{xx}\boldsymbol{H}_x. \tag{3.33}$$

Similar to the $\boldsymbol{J}_x$ matrix from (3.29), $\boldsymbol{H}_x$ is the Jacobian of $\boldsymbol{h}$ wrt. $\vec{x}$ evaluated at $\vec{x}_k^*$.

3.5 Probabilistic graphical models

3.5.1 Bayesian networks

Bayesian networks, or Bayes nets are graphical models that represent the joint probability of random variables in the form of a *directed acyclic graph (DAG)*. Bayes nets are useful in that they give an intuitive representation the involved variables and the marginal and conditional dependencies between them, characterized by nodes and edges in the graph. Because the nets are acyclic, any variable in the network is only affected by its predecessors. The notion of independence between variables follows the concept of *d-separation* [21]. In the example shown in figure 3.3, the variable pair (x_0, l_1) are *marginally independent* because there is no directed path connecting the pair, but conditioned on $z_{0,1}$ they become *conditionally dependent*. To further illustrate the Bayes net, the variable x_1 and its marginal is only affected by its parent nodes x_0 and u_1, while the PDF of x_1 conditioned on $z_{1,1}$ and $z_{1,2}$ is influenced by l_1 and l_2, i.e.

$$p(x_1|l_1, l_2) = p(x_1) = \iint_{x_0,u_1} p(x_1, x_0, u_1)\, du_1\, dx_0,$$

but

$$p(x_1|l_1, l_2, z_{1,1}, z_{1,2}) \neq p(x_1)$$

In summary, the complete joint PDF of the entire set of variables in figure 3.3 is given as

$$p(x_{0:2}, l_{1:2}, u_{1:2}, z_{0,1:2,2}).$$

By applying the chain rule

$$p(x, y) = p(y|x)p(x) = p(x|y)p(y), \tag{3.34}$$

and imposing the conditional and marginal independencies indicated by the graph edges, the joint probability is deconstructed to a product of prior and conditional

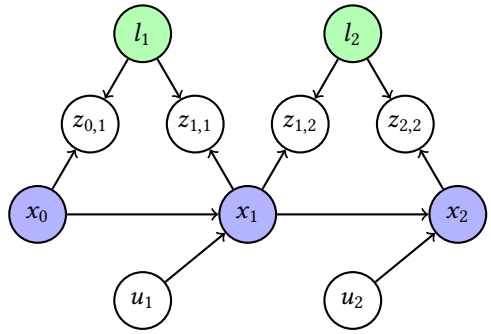


Figure 3.3: Bayes net with twelve variables. Variables are only directly affected by their immediate ancestor(s).

densities:

$$\begin{aligned} p(x_{0:2}, l_{1:2}, u_{1:2}, z_{0,1:2,2}) &= p(x_0) \cdot p(l_1) \cdot p(l_2) \cdot p(u_1) \cdot p(u_2) \\ &\times p(x_1|x_0, u_1) \cdot p(x_2|x_1, u_1) \\ &\times p(z_{0,1}|x_0, l_1) \cdot p(z_{1,1}|x_1, l_1) p(z_{1,2}|x_1, l_2) \cdot p(z_{2,2}|x_2, l_2) \end{aligned} \tag{3.35}$$

In practical applications such as SLAM, some variables in (3.35) are known, while others are not, and are sought to be estimated. In these instances, the full joint probability in (3.35) is reformulated as a *posterior* PDF. The posterior encapsulates the probability distribution of an event given some prior information. If for instance the z and u variables above are known, the posterior distribution $p(x_{0:2}, l_{1:2}|u_{1:2}, z_{0,1:2,2})$ is found using *Bayes theorem*:

$$p(x_{0:2}, l_{1:2}|u_{1:2}, z_{0,1:2,2}) = \frac{p(u_{1:2}, z_{0,1:2,2}|x_{0:2}, l_{1:2}) p(x_{0:2}, l_{1:2})}{p(u_{1:2}, z_{0,1:2,2})}. \tag{3.36}$$

3.5.2 Factor graphs

An alternative to Bayes nets are *factor graphs*. A factor graph is formally defined as a *bipartite* graph $\mathcal{G} = (\Theta, \mathcal{F}, \mathcal{E})$ with two sets of nodes: variables $\theta_i \in \Theta$, and factors

$f_j \in \mathcal{F}$. Because the graph is bipartite, the edges $e_{ij} \in \mathcal{E}$ always connect one factor and one variable. The edges are also undirected. A factor graph graphically expresses the factorization of a function

$$f(\Theta) = \prod_j f_j(\Theta_j), \tag{3.37}$$

where Θ_j denotes the variable set with edges connecting the variables to the factor f_j. One major difference between Bayes nets and factor graphs is that Bayes nets are tied to probability densities, and cannot inherently factor in other types of functions, such as likelihoods (unnormalized PDFs). As we are not always interested in the posterior probabilities directly, but rather seek to find the values of certain variables, the factor graph formulation is favorable. Based on the example Bayes net in figure 3.3 and the associated joint PDF (3.35), the variable set $\Theta = \{x_0, x_1, x_2, l_1, l_2\}$ is represented using a factor graph in figure 3.4. The graph provides a formulation of the function

$$\begin{aligned} f(\Theta) = & f_1(x_0) \cdot f_2(l_1) \cdot f_3(l_3) \\ & \times f_4(x_0, x_1) \cdot f_5(x_1, x2) \\ & \times f_6(x_0, l_1) \cdot f_7(x_1, l_1) \cdot f_8(x_1, l_2) \cdot f_9(x_2, l_2). \end{aligned} \tag{3.38}$$

in which $f(\Theta) \propto p(x_{0:2}, l_{1:2}, u_{1:2}, z_{0,1:2,2})$. The single variable functions in (3.38) are proportional to the first three prior densities from (3.35), while $p(u_1)$ and $p(u_2)$ are integrated into the factors connecting x_0, x_1 and x_2: $f_4(\cdot)$ and $f_5(\cdot)$. The conditional densities $p(x_1|x_0, u_1)$ and $p(x_2|x_1, u_2)$ are also included here. As briefly discussed in the previous section, if the variables u and z are known, the expression in (3.38) readily expresses the posterior distribution in (3.36), as the factors implicitly include any prior information about these variables. Lastly, the equivalence between the densities and factors relating the x-variables to the l-variables is established as $f_k(x_i, l_j) \propto p(z_{i,j}|x_i, l_j)$.

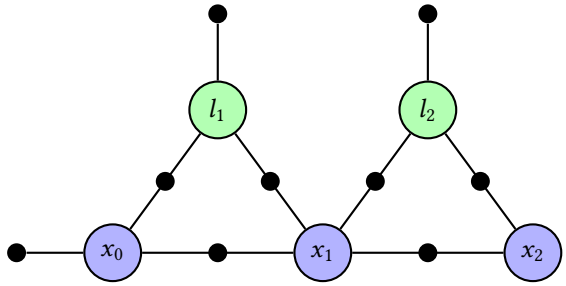


Figure 3.4: Factor graph representation of the Bayesian network in figure 3.3. Variables are labeled nodes, factors are black nodes that have edges to their affected variables.

Chapter 4

SLAM Backend - Optimization

4.1 SLAM system task

To establish a concrete frame for the SLAM system developed in this project, a more functional specification of its role in Revolves autonomous software pipeline is supplied at the beginning of this chapter. This section is placed here to specify the problem description from chapter 1, and to serve as a backdrop for the more general problem description and concepts that follow.

In essence, the input and output data related to the SLAM system at Revolve is quite simple, as there are mainly two types of data being received and passed on.

Input: From the detection systems, a list of points $\vec{p} = (p_1, \ldots, p_n)^\mathsf{T}, \quad \vec{p} \in \mathbb{R}^{n\times 2}$ is sent at every iteration. These are points in the horizontal plane which are *assumed* to be cones resulting from a *feature extraction* step performed on exteroceptive sensor input such as a camera or lidar (See section 6.1 for more specifics on this step as it is performed in Revolve driverless). The data is given relative to an origin fixed at the center of gravity (CG) of the vehicle, referred to as the *body frame* $\mathcal{B}$. The body frame is defined with the x-axis pointing out of the nose of the car. The y-axis points out of the left lateral side of the car.

Moreover, a stream with samples of the estimated inertial pose (position and

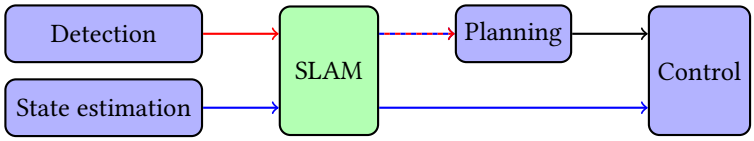


Figure 4.1: Flow and types of data between nodes adjacent to SLAM. Red arrows indicate point lists $\vec{p} \in \mathbb{R}^{n\times 2}$. Blue arrows indicate vehicle pose $\vec{x}$ in 3degrees of freedom (DOF)

.

orientation) $\vec{x}_b^i = (x, y, \theta)^\mathsf{T}$ of the vehicle's CG is sent from the *state estimation* module. Poses are given in the *map* frame, denoted $\mathcal{I}$. This module is also responsible for supplying the control systems with relevant information such as translational and rotational velocity and acceleration estimates. In the SLAM domain, such a system is often referred to as *odometry* to recognize its *internal* state estimation (acceleration etc.), as opposed to the *external* state estimation performed by SLAM. "External state" is here referring to the state, specifically the pose, of the vehicle in relation to its surroundings.

Output: Similar to the input, one of the two output data streams of the SLAM system is a list of points in the inertial map frame $\mathcal{I}$. These are, in the point of view of the autonomous system, the locations of actual cones in the track. That being said, there is no guarantee that all cones are found and placed in this list, nor that all points in the list truly are cones. The other output is the pose estimates originating from the state estimation module with applied corrective offsets. This is the vehicle pose estimate calculated in the SLAM module.

In Revolve's autonomous software pipeline, the SLAM module is subsequent to the detection and state estimation modules. The output SLAM data is utilized by the *path planning* and control systems. The flow of data involving the SLAM module and the adjacent systems is shown in figure 4.1.

4.2 Problem formulation

This section and subsequent sections in this chapter build further on the author's writing in the specialization project [43], as the relevant theory is largely the same. That being said, the material has been reconsidered and rewritten, and new theoretical concepts are added for completeness.

All autonomous agents operating in the physical world are exposed to a number of unpredictable processes. These include wind, waves, electromagnetic interference and more, and can impact system operation in multiple ways. Because of the inherent randomness in these processes, they are usually modeled as random variables. In any sensor that is designed to measure some external quantity, the output will always be some combination of external and internal random signals combined with the true value. This leads to the motivation for formulating the probabilistic structure of the SLAM problem. In SLAM, the ultimate goal is to maximize the posterior joint probability of the agent states $X = \{\vec{x}_1, \ldots, \vec{x}_k\}$ and map $L = \{\vec{l}_1, \ldots, \vec{l}_n\}$ with stationary landmarks, conditioned on set of measurements $Z = \{\vec{z}_1, \ldots, \vec{z}_k\}$ and motion inputs $U = \{\vec{u}_1, \ldots, \vec{u}_k\}$:

$$p(X, L|Z, U, \vec{x}_0)$$

This way, the estimated solution is the value of the states X and landmarks L that are the most probable given the collected information. The state $\vec{x}_0$ is also included to denote prior information about the initial state of the problem, e.g. the starting position of a robot in relation to an earth-fixed location. Expanding and rearranging terms of the posterior above, and applying Bayes theorem (3.36) yields

$$\begin{aligned} p(X, L|Z, U, \vec{x}_0) = {} & p(\vec{x}_k|X_{0:k-1}, L, Z, U) \\ & \times \frac{p(\vec{z}_k|X, L, Z_{1:k-1}, U) \cdot p(X_{0:k-1}, L|\vec{x}_k, Z_{1:k-1}, U)}{p(\vec{z}_k|Z_{1:k-1}, U, \vec{x}_0)}. \end{aligned} \tag{4.1}$$

Coincidentally, figure 3.3 is a graphical formulation of the SLAM problem, which means that the same assumptions of independent variables can be made. The resulting expression in (4.1) can be simplified using the *Markov assumption* that the agent state

$\vec{x}_k$ is only dependent on its previous state $\vec{x}_{k-1}$ and the motion input $\vec{u}_k$ applied between time steps $k-1$ and k. Furthermore, a measurement $\vec{z}_k$ is conditionally independent of the previous measurements given the landmarks, as well as all previous states $\vec{x}_{0:k-1}$. The simplified expression for the posterior probability in (4.1) becomes:

$$p(X, L|Z, U, \vec{x}_0) \propto p(\vec{z}_k|\vec{x}_k, L) \cdot p(\vec{x}_k|\vec{x}_{k-1}, \vec{u}_k) \cdot p(\vec{x}_{1:k-1}, L|\vec{z}_{1:k-1}, \vec{u}_{1:k-1}) \tag{4.2}$$

The first term on the right side of eq. (4.2) is usually termed the *measurement model* of the system. The measurement model indicates the probability of making an observation $\vec{z}_k$ given the current state of the map and robot. In terms of the nonlinear dynamic models discussed in section 3.3, the measurement model can be made more explicit by altering the expression in (3.28) with the addition of multivariate *additive white Gaussian noise (AWGN)* $\vec{v}_k$:

$$\vec{z}_k = \boldsymbol{h}(\vec{x}_k, L) + \vec{v}_k. \tag{4.3}$$

As the number of landmarks is usually larger than one, (4.3) will be a vector of size $n \times d$ where d is the dimension of a single measurement taken of a landmark. As an example, for measurements taken directly by a camera, $d = 2$ while landmarks are in $\mathbb{R}^3$ because the three-dimensional scene is flattened onto the image plane.

The second term on the right hand side of (4.2) is termed the *motion model* of the robot. It models the PDF of the predicted state $\vec{x}_k$ as a function of the previous state $\vec{x}_{k-1}$ and the applied input $\vec{u}_k$. Again based on the nonlinear models in section 3.3, we can introduce a AWGN noise parameter $\vec{w}_k$ to get an expression for the general nonlinear motion model with the random vector $\vec{x}_{k+1}$:

$$\vec{x}_{k+1} = \boldsymbol{f}(\vec{x}_k, \vec{u}) + \vec{w}_k. \tag{4.4}$$

Finally, the rightmost expression of (4.2) indicates the recursive Bayesian estimation pattern as it is similar to expression for the full posterior above, but does not include variables from the k^{th} time step. This pattern lends itself to an incremental filtering approach, as each iteration can initialize using results from the previous. Repeating

the operation recursively from $k-1$ to 0, we get the complete expression for all k time steps:

$$p(X, L|Z, U, \vec{x}_0) \propto \prod_{i=1}^{k} p(\vec{x}_i|\vec{x}_{i-1}, \vec{u}_i) \cdot p(\vec{z}_i|\vec{x}_i, L) \tag{4.5}$$

4.3 Maximum a posteriori estimation

The concept of MAP estimation is based on estimating values that best explain the given data. As an approach, it is therefore similar to ML estimation, but is expanded to include any prior information that may be available [38]. When considering the SLAM problem as it is stated in eq. (4.5), we can use MAP estimation to find a solution: Given the set of measurements, and any prior knowledge about the system states (vehicle pose or landmark locations), the MAP estimate is the set of state values that maximizes this posterior distribution. Because MAP estimation is not limited to any amount of data, full trajectory batch estimation is usually formulated as a MAP estimation problem. Also, in contrast to EKF-SLAM, MAP estimation is not directly coupled with the formulation of motion and measurement models, and can include arbitrary probabilistic relationships in the solution[6]. MAP estimation is, for this reason, a powerful tool that can incorporate many types of data and constraints relevant to the specific applications.

As the posterior describing the SLAM problem is already formulated in eq. (4.5), the MAP framework can be directly applied. For simplicity, we define the augmented state vector $\Theta = (X, L)$ containing the full trajectory of robot states as well as the landmarks in the map. The estimate of Θ that maximizes (4.5) is the MAP estimate, and is defined as

$$\begin{aligned} \Theta^{MAP} &= \arg\max_{\Theta} p(\Theta|Z, U) \\ &= \arg\min_{\Theta} \big(-log\left[p(\Theta|Z, U)\right] \big). \end{aligned} \tag{4.6}$$

The noise parameters in equations (4.4) and (4.3) are assumed to be zero-mean Gaussian: $\vec{v}_k \sim \mathcal{N}(\vec{0}, \Lambda_k), \vec{w}_k \sim \mathcal{N}(\vec{0}, \Sigma_k)$, which results in the Gaussian formulations of the

motion and measurement models:

$$\boldsymbol{f}(\vec{x}_k, \vec{u}) + \vec{w}_k \sim \mathcal{N}(\boldsymbol{f}(\vec{x}_k, \vec{u}_k), \Sigma_k) \tag{4.7}$$

$$\boldsymbol{h}(\vec{x}_k, L) + \vec{v}_k \sim \mathcal{N}(\boldsymbol{h}(\vec{x}_k, L), \Lambda_k). \tag{4.8}$$

Lastly, the MAP estimate is found by inserting the right hand side of (4.5) into (4.6) and applying the definitions above:

$$\Theta^{MAP} = \arg\min_{\Theta} \left\{ -log \left[\prod_{i=1}^{k} p(\vec{x}_i|\vec{x}_{i-1}, \vec{u}_i) p(\vec{z}_i|\vec{x}_i, L) \right] \right\} \tag{4.9}$$

$$= \arg\min_{\Theta} \left\{ \sum_{i=1}^{k} \|\boldsymbol{f}(\vec{x}_{i-1}, \vec{u}_i) - \vec{x}_i\|^2_{\Sigma_i} + \|\boldsymbol{h}(\vec{x}_i, L) - \vec{z}_i\|^2_{\Lambda_i} \right\} \tag{4.10}$$

4.4 Factor graphs in SLAM

In recent years, the shift from recursive filtering to smoothing based SLAM methods has fueled the need for involving data structures that make use of the sparsity of the smoothing information matrix. Examples of such structures include the *Treemap* introduced in [20], the Gaussian Markov random field (GMRF) as used in [47], and notably the *Bayes tree* presented in [30] and its application in *iSAM2* [31, 32]. In any case, sparse linear algebra is employed to allow efficient operation on the information matrix, which in turn leads to accelerated inference.

The link between linear algebra and graph theory is strong. Problems in one domain can be transferred and manipulated in the other, resulting in an extensive framework for solving problems in computer science, mathematics, biology and more [34]. In SLAM, expressing the problem using *Bayes networks* or factor graphs is convenient because, especially in the case for Bayes nets, the connection between the graphical model and the real world is made clear: States are stochastically related through measurements, and conditional (in)dependencies are made explicit. On the other hand, factor graphs are more comfortable to work with from a mathematical point of view. In contrast to Bayes nets, they can describe any function (including likelihoods), not just

normalized probabilities. Further, factor graphs benefit from their close relationship to linear algebra.

4.4.1 MAP inference

Building on the theory supplied in section 3.5.2, the MAP estimation problem presented in (4.10) can equally be expressed using factor graphs. To start off, (4.5) is proportionally factorized as

$$f(X, L) = f_0(\vec{x}_0) \prod_{i=0}^{k} f_{(i-1)i}(\vec{x}_i, \vec{x}_{i-1}) \prod_{\{i,j\}} f_{ij}(\vec{x}_i, \vec{l}_j). \tag{4.11}$$

Or, in terms of the concatenated state variable $\Theta = \{\theta_0, \ldots, \theta_{k+n}\} = (X, L)$ (k time steps and n landmarks):

$$f(\Theta) = f_0(\theta_0) \prod_{i=0}^{k} f_{(i-1)i}(\theta_i, \theta_{i-1}) \prod_{\{i,j\}} f_{ij}(\theta_i, \theta_j). \tag{4.12}$$

The factors here are based on the motion and measurement models presented in section 4.2 with some slight modifications: Input vectors $\vec{u}_i$ and measurements $\vec{z}_i$ are encoded by factors connecting subsequent robot states and factors connecting states and landmarks. Moreover, the set $\{i, j\}$ in the second product is defined as correspondences between landmark j and robot state i established by measurements of the landmark $\vec{l}_j$ at state $\vec{x}_i$. Building this set is not trivial, but has been assumed known thus far.

By introducing the general prediction function $\boldsymbol{g}(\Theta_i)$ for the variable subset $\Theta_i \in \Theta$, related measurement $\vec{z}_i$ and associated covariance matrix Σ_i (4.12) is reformulated to

$$f(\Theta) = \prod_{i=0}^{N} \exp\left(-\frac{1}{2} \|g(\Theta_i) - \vec{z}_i\|^2_{\Sigma_i}\right) \tag{4.13}$$

The right-hand side of eq. (4.13) is a product of N Gaussian likelihood functions, and proportional to the posterior distribution in eq. (4.5). Hence, the optimal state estimates Θ^* are found by maximizing the expression in (4.13). The maximization can

be formulated as a non-linear least squares (NLLSQ) problem:

$$\begin{aligned}\Theta^* &= \arg\max_{\Theta} \ f(\Theta) \\ &= \arg\min_{\Theta} \{-\log f(\Theta)\} \\ &= \arg\min_{\Theta} \left\{ \sum_{i=0}^{N} \|g(\Theta_i) - \vec{z}_i\|^2_{\Sigma_i} \right\}, \end{aligned} \tag{4.14}$$

which is a generalized, but equivalent formula for the MAP estimate from eq. (4.10). An explicit distinction between the motion and measurement errors as they are presented in eq. (4.10), is not made here to emphasize the factor graph representation using variables and factors.

4.5 Non-linear least squares

In contrast to linear least squares problems where closed form solutions exist, NLLSQ problems such as the one presented in eq. (4.14) have to be numerically approximated. Common *iterative* methods for performing this task include Dogleg, Levenberg-Marquardt and *Gauss-Newton (GN)*. All these methods involve approximation by linearization of a nonlinear system of equations, but are differentiated by their *update* steps at each iteration. The primary steps of GN are applied to eq. (4.14) as follows:

1. Linearizing at point Θ^0 using the Jacobian $\boldsymbol{J}_\Theta$ of $g(\Theta)$ wrt. Θ:

$$f(\Theta) \approx \sum_{i=0}^{N} \left\| \boldsymbol{J}_{\Theta_i}(\Theta_i - \Theta_i^0) - (\vec{z}_i - g(\Theta_i^0)) \right\|^2_{\Sigma_i}. \tag{4.15}$$

2. Apply a *whitening transform* to the linearized expression above to express the regular linear least squares problem for finding the optimal update step $\Delta_\Theta = \Theta - \Theta^0$:

$$\Delta_\Theta = \arg\min_\Theta \sum_{i=0}^{N} \left\| \Sigma_i^{-1/2} \boldsymbol{J}_{\Theta_i} \Delta_{\Theta_i} - \Sigma_i^{-1/2} (\vec{z}_i - g(\Theta_i^0)) \right\|_2^2 \tag{4.16}$$

The whitened Jacobians and constants can then be aggregated into $\boldsymbol{A}$ and $\boldsymbol{b}$ respectively, resulting in the *linear* least squares formulation:

$$\Delta_\Theta = \arg\min_\Theta \| \boldsymbol{A} \Delta_\Theta - \boldsymbol{b} \|_2^2 \tag{4.17}$$

3. By matrix decomposition, such as through *Cholesky*[23], one can solve for Δ_Θ by factorizing the *information matrix* $\Omega = \boldsymbol{A}^\mathsf{T} \boldsymbol{A}$ in the *normal equation*:

$$(\boldsymbol{A}^\mathsf{T} \boldsymbol{A}) \Delta_\Theta = \boldsymbol{A}^\mathsf{T} \boldsymbol{b} \tag{4.18}$$

4. If the user-defined termination-criteria are not met, steps 1-3 are repeated with $\Theta_0 = \Theta_0 + \Delta_\Theta$. Otherwise, $\Theta_0 + \Delta_\Theta$ is the found approximate solution.

The iSAM2 algorithm as part of the Georgia Tech Smoothing and Mapping (GTSAM) C++ library [9], which is at the core of the work presented in this thesis, supports both Dogleg and GN-based optimization. The GN-based version is the one implemented in this work, and so only GN is presented here.

In step 3, the Cholesky factorization generates the upper triangular matrix

$$\boldsymbol{R} : \Omega = \boldsymbol{A}^\mathsf{T} \boldsymbol{A} = \boldsymbol{R}^\mathsf{T} \boldsymbol{R}, \tag{4.19}$$

which due to its structure is the key to efficiently calculate Δ_Θ using *back-substitution*. Back-substitution on $\boldsymbol{R}$ begins with the lower-right element on the matrix diagonal, which is already found as it is the only element in the row, and iteratively works its way up the rows until the full update vector is found. In the realm of batch optimization, its relation to the information matrix has led to $\boldsymbol{R}$ being referred to as the *square root information matrix (SRIM)*. Although not first introduced by Dellaert and Kaess [11], it is a crucial element in the efficiency of iSAM2 and its predecessors [29, 31, 32].

4.5.1 Factors and the square root information matrix

It can be shown that the whitened Jacobian matrix $\boldsymbol{A}$ represents the underlying factor graph used to model the original NLLSQ problem supplied in eq. (4.14). To exemplify, we can consider the factor graph in figure 3.4 where we evaluate the factor connecting the variable pair $\{x_0, l_1\}$. For continuity, x_0 is defined as the state of the robot, l_1 is a landmark, and $z_{0,1}$ is the measurement imposing dependence between the two variables. The linearization points are in this case the currently available estimates of the variables, indicated with hatˆ -notation. The corresponding linear least square residual r for this factor with the variable set $\Theta_i = (x_0, l_1)^\mathsf{T}$ and corresponding measurement Jacobian $\boldsymbol{H}_{x_0,l_1}$, is then found as

$$\begin{aligned} r &= \left\| h(x_0, l_1) - z_{0,1} \right\|_\Lambda^2 \\ &\approx \left\| \boldsymbol{\Lambda}^{-1/2} \boldsymbol{H}_{x_0,l_1} \Delta_{\Theta_i} - \boldsymbol{\Lambda}^{-1/2} (z_i - h(\hat{x}_0, \hat{l}_1)) \right\|^2 \end{aligned} \tag{4.20}$$

For the full state vector $\Theta = (x_0, x_1, x_2, l_1, l_2)$ and whitened Jacobian matrix $\boldsymbol{A}$, this residual corresponds to the i^{th} row of $\boldsymbol{A}$ with nonzero elements at columns 1 and 4 (x_0 and l_1 are the 1^{st} and 4^{th} elements of Θ). To further convey the correspondence between factors in the factor graph and elements of the whitened Jacobian matrix used to find the MAP estimate in eq. (4.18), the factor connecting the consecutive poses x_0 and x_1 is embedded in entries at column 1 and 2 in $\boldsymbol{A}$ at the row index of the factor.

Factorization of $\Omega = \boldsymbol{A}^\mathsf{T}\boldsymbol{A}$ into the SRIM is shown in [12] to be equivalent to performing the *variable elimination algorithm*[22] on the factor graph associated with $\boldsymbol{A}$. In fact, sparse linear algebra factorization such as Cholesky or QR are special cases of the elimination algorithm. The graph structure representing the SRIM $\boldsymbol{R}$ is, as a result, a Bayes net. This Bayes net encodes the joint distribution of the state variables: $p(\Theta)$.

Depending on the ordering of variables and factors in the original NLLSQ problem in eq. (4.14), the $\boldsymbol{A}$ matrix can take several different forms, which then causes $\boldsymbol{R}$ and the complementary Bayes net to change. The joint distribution $p(\Theta)$ remains unchanged, however. The computational complexity of performing back-substitution

on $\boldsymbol{R}$ is highly dependent on this variable ordering, and so it should be performed with care. In cases where the variable ordering is poor, the amount of nonzero off-diagonal elements in $\boldsymbol{R}$ will increase. This is referred to as *fill-in*, and should be avoided as much as possible to maintain efficient operations on the SRIM. Applying heuristic variable ordering schemes is a powerful tool to reduce fill-in, but can be a potentially expensive operation in itself, and so should be performed consciously also. In SLAM, a useful assumption is that recently seen landmarks are likely to be present in subsequent measurements. Heuristic variable ordering schemes such as constrained COLAMD (CCOLAMD) [10] can take advantage of these kinds of features of the specific problem and significantly reduce the computational effort at no loss of accuracy [1].

4.6 iSAM2

Regardless of how efficient back-substitution performed on $\boldsymbol{R}$ is, the standard procedure of calculating eq. (4.18) involves updating all linearization points using the update vector Δ_Θ every time either a new measurement of an existing variable enters or when a state variable is added to the state vector. The main features of the original version of iSAM were the incremental execution of these two actions. Instead of having to re-factorize the $\boldsymbol{R}$ matrix at every new incoming measurement, the introduction of *Givens rotations* allowed the incremental addition of new measurement rows into the SRIM [33]. In order to ensure efficient operation, the algorithm performed periodic batch updates of variable ordering to limit the cost of back-substitution.

The forming of iSAM2 in [31] followed from the introduction of the Bayes tree[30]. The Bayes tree is essentially a Bayes net where *chordal* parts of the net are contained in *cliques*. A key takeaway from the Bayes net is that new information (measurements) is propagated from the child nodes to the root. This means that descendants of the clique with the affected variables are not included in computations. In other words, "old" states at the bottom of the tree are not affected by changes in newer states which are situated further up in the tree, adhering to the CCOLAMD ordering. Other notable features of iSAM2 include:

- Fluid relinearization: The state update vector is changed when new data is added

to the Bayes tree. To avoid having to relinearize all variables, variables with the state update value smaller than a threshold $\Delta < \beta$ are not linearized. To relinearize variables, all their connected factors have to be extracted from the Bayes tree and linearized, which leads to more processing.

- Partial state update: Only when the calculated state update value increases sufficiently from one iteration to the next($\Delta_{k+1} > \Delta_k + \alpha$), should the value actually get updated for the particular variable. Because of the structure of the Bayes tree, it is guaranteed that variables further down than a variable whose state update is not updated, will see an even smaller update, and so are not considered.

Chapter 5

SLAM Frontend - Data association

In the previous chapter, the errors of the predicted landmark locations are minimized based on implicitly chosen measurements. For the MAP estimates to be consistent with the true state of the problem it is trying to solve, these measurements are therefore *assumed* to originate from the particular landmark in question. In practice, determining these correspondences is not trivial and has to be done with care, as making wrong ones can deteriorate the MAP estimates. This chapter presents some methods for tackling this challenge, termed the data association problem.

Data association in SLAM is usually presented as considering a moving agent with an unknown, estimated pose, in an environment with unknown stationary landmarks. The ultimate objective in data association is that the landmarks are consistently tracked throughout the operation, allowing for consistent pose estimates. In this sense, SLAM data association is contrary to the *tracking problem* where the observing agent is stationary, and the movement (track) of the features is to be estimated. Conceptually, the problem can be presented with a set of stationary landmarks $L = \{\vec{l}_i\}_{i=1}^{N}$ and a set of measurements $Z = \{\vec{z}_j\}_{j=1}^{M}$. The goal is then to establish a *hypothesis* with a set of

associations $\mathcal{H} = \{a_j\}_{j=1}^{M}$ where

$$a_j = \begin{cases} (j, i) & \text{if } \vec{z}_j \text{ is paired with } \vec{l}_i. \\ (j, 0) & \text{if } \vec{z}_j \text{ has no pairing.} \end{cases} \tag{5.1}$$

Because the application of SLAM usually involves operation over time, two fundamental abilities are needed in data association: short and long term tracking. These two challenges are equal in their formulation but often require different solutions. Because of the inherent pose uncertainty that increases over time in SLAM, the problem of data association becomes increasingly difficult when an area is being revisited after prolonged operation. Hence, in large-scale SLAM applications, additional methods such as the appearance-based bag of words (BoW) approaches described in [8] and [36], are required for closing wide loops. In such a system, a designated worker thread is responsible for comparing the current scene to a subset of the previously visited ones and searching for a potential match. In this thesis, no explicit loop closing algorithms such as these are employed, as all landmark tracking, both short- and long-term, is handled by the approaches in the following sections.

5.1 Assumptions

When operating in real-world scenarios, data association cannot be guaranteed to be perfect and unambiguous. The most common source of ambiguity is the fact that at the estimated pose of the vehicle can diverge from its true state. All data association choices will therefore be based on possibly inaccurate estimations, leading to even more errors. This poses a big challenge for loop closure. In short term tracking, data association is less affected by these types of uncertainties as they are less significant in a local frame of reference. In these instances, instantaneous uncertainties can be detrimental to the measurement-landmark associations. Data association uncertainty in this sense is comprised of three main factors:

1. The grade of accuracy and precision of the sensor supplying the measurements.

2. The complete lack of a measurement of a landmark within the field of view of the sensor, or false negative (FN).

3. Spurious measurements of non-existent landmarks, or false positives (FPs).

To reduce the scope of the assignment problem, and also to mitigate the issues above, domain-specific constraints can be applied. For instance are the cones in FS competitions situated with some space in between, and colored according to which side of the track they are on. Another commonly applied constraint is *mutual exclusion*: that only *one* measurement can originate from each landmark at every iteration.

5.2 Nearest Neighbor

The simplest form of data association is also the most intuitive one: match the measurements to the landmarks to which they are closest to in Euclidean space. As measurements $\vec{z}$ are related to the estimated position of landmarks through the predictive measurement function in eq. (4.3), the nearest neighbor (NN) association for each measurement is found as

$$a_j = (j, i) : i = \arg\min_i \left\| h(\vec{x}, \vec{l}_i) - \vec{z}_j \right\|_2^2 . \tag{5.2}$$

In the cases where two or more measurements are paired with the same landmark, which is probable to happen, three solutions are considered here:

1. Choose the measurement-landmark pairings with the shortest distance between them.

2. Make a random pick of all the associations involving the landmark.

3. Keep track of the distance between all pairs, and employ a global assignment method such as the *Hungarian method* [35] with the sum of distances as the cost to be minimized. This type of global assignment is fittingly referred to as global nearest neighbor (GNN).

To account for potential FPs, it is common to add an upper bound to the distance metric in (5.2) so that spurious detections outside a specific region are immediately discarded. In landmark-based SLAM applications, it is preferred to miss some associations over making wrong ones because the number of other tracked landmarks is in most cases sufficient. Wrong associations will equally be passed on to the MAP estimation performed in the backend, which can potentially lead to unrecoverable estimation errors.

5.3 Maximum likelihood

The pure nearest neighbor approach as given in (5.2) has an significant shortcoming: it does not utilize any information about the underlying distribution on errors of estimated landmarks. Instead of the squared 2-norm distance metric, ML data association determines the measurement-landmark pairing using the squared Mahalanobis distance (3.23):

$$a_j = (j, i) : i = \arg\min_i \left\| h(\vec{x}, \vec{l}_i) - \vec{z}_j \right\|^2_{\boldsymbol{S}_{ij}} . \tag{5.3}$$

The covariance matrix $\boldsymbol{S}$ is a parameter of the distribution of the error between the predicted landmark location and the measurement. By (3.22) and the results from section 3.4, we can propagate the joint covariance of predicted state $\vec{x}$ and landmark $\vec{l}_i$

$$\Sigma_i = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xl_i} \\ \Sigma_{l_i x} & \Sigma_{l_i l_i} \end{pmatrix} \tag{5.4}$$

through the nonlinear $h(\cdot)$ function with the Jacobian matrices as in (3.30):

$$\boldsymbol{H}_i = \begin{pmatrix} \boldsymbol{H}_x & \boldsymbol{H}_{l_i} \end{pmatrix}. \tag{5.5}$$

And finally, together with the measurement model, where we have assumed that measurements are subject to Gaussian noise $\mathcal{N}(\vec{0}, \Lambda_j)$, we find $\boldsymbol{S}_{ij}$ as

$$\boldsymbol{S}_{ij} = \boldsymbol{H}_i \Sigma_i \boldsymbol{H}_i^\top + \Lambda_j. \tag{5.6}$$

In the case of many associations to the same landmark, the same measures as described in the previous section can be applied.

5.4 Individual compatibility

As with the NN approach, limiting FPs is something that needs to be considered when performing the ML assignment in (5.3). Because the squared Mahalanobis distance is χ^2 distributed, certain pairs can be rejected according the desired critical value. To accept the matching of a measurement and a landmark, and thereby accept the association $a(i, j)$, the residual or *innovation* vector $\vec{v}_{ij} = h(\vec{x}, \vec{l}_i) - \vec{z}_j$ is subject to the individual compatibility (IC) test:

$$\vec{v}_{ij}^\top \boldsymbol{S}_{ij}^{-1} \vec{v}_{ij} \sim \chi_k^2 < \gamma. \tag{5.7}$$

The number of degrees of freedom (k) is equal to $dim_{\mathbb{R}}(v_{ij})$. For intuition, a threshold $p \in [0, 1]$ on the CDF of the χ^2 distribution is used as the tuning parameter to signify the probability of the IC test passing if the detection $\vec{z}_j$ is truly an observation of $\vec{l}_i$. The relationship between p and γ is shown in equation (3.24).

Using ML data association as described above, means that the covariance estimates have to be retrieved for every landmark in L at every incoming new set of measurements. For large environments especially, and depending on the structure of the information matrix, this can be an expensive operation. Additionally, the right-hand side of (5.3) is calculated $N \times M$ times, and includes the inversion of the $\boldsymbol{S}_{ij}$ matrix. To reduce this computational load, using NN assignments conditioned on the IC check is a possible solution.

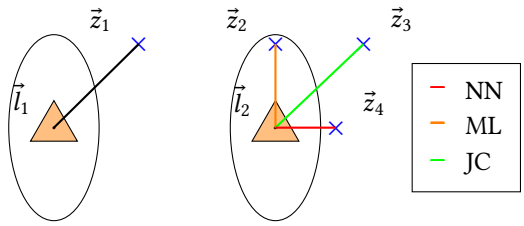


Figure 5.1: Four measurements including two spurious ones, and two landmarks with their respective distributions indicated with ellipses. All three association schemes correctly determine the association $a_1 = (1, 1)$, indicated with a black line. With landmark $\vec{l}_2$, the nearest neighbor method wrongly picks $a_4 = (4, 2)$, and the Mahalanobis distance-based maximum likelihood association $a_2 = (2, 2)$ is also wrong. joint compatibility (JC) incorporates the correlation between the pair of landmarks and applies it to the measurements, leading to the correct assignment $a_3 = (3, 2)$).

5.5 Joint Compatibility

Although IC accounts for the probability distributions of the individual landmarks, it does not incorporate information about the internal correlation between a *set* of landmarks to a *set* of measurements. To illustrate this point, figure 5.1 displays two landmarks and four detections where $(\vec{z}_2, \ldots, \vec{z}_4)$ are all IC with $\vec{l}_1$. Visually, one can deduce that the optimal hypothesis is $\mathcal{H}^* = \{a_1, a_2, a_3, a_4\} = \{(1, 1), (2, 0), (3, 2), (4, 0)\}$ because the topology of the measurements matches the topology of the landmarks. However, both the NN and IC associations fail because the implication that the error-causing shift affects all the measurements equally is not taken into consideration. That being said, if the measurement noise ($\mathbf{\Lambda}$ from (5.6)) is similar in scale to the pose error itself, the result in 5.1 is not as clear as the connection between $\vec{l}_2$ and $\vec{z}_3$ is weakened.

The concept of JC was first introduced by Neira and Tardós [42] and is designed to consider the correlations discussed above. To achieve this, JC evaluates a hypothesis of individually compatible pairings for their joint consistency. For the hypothesis $\mathcal{H} = \{a_j\}_{j=1}^{M}$, we have $a_j = (i, j)$ and $\vec{l}_{a_j} = \vec{l}_i$, where landmark $\vec{l}_i$ and measurement $\vec{z}_j$

are IC. The joint innovation vector of all associations in $\mathcal{H}$ is written as

$$\vec{v}_{\mathcal{H}} = \begin{pmatrix} \vec{v}_{a_1} \\ \vdots \\ \vec{v}_{a_M} \end{pmatrix} = \begin{pmatrix} h(\vec{x}, \vec{l}_{a_1}) - \vec{z}_1 \\ \vdots \\ h(\vec{x}, \vec{l}_{a_M}) - \vec{z}_M \end{pmatrix} = h_{\mathcal{H}}(\vec{x}, \vec{l}) - \vec{z}_{\mathcal{H}} \tag{5.8}$$

To find the joint covariance of this innovation vector, the Jacobian of the joint measurement function

$$\frac{\partial h_{\mathcal{H}}}{\partial(\vec{x}, \vec{l}_{\mathcal{H}})} = \boldsymbol{H}_{\mathcal{H}} = \begin{pmatrix} \boldsymbol{H}_x^{a_1} & \cdots & \boldsymbol{H}_{l_{a_M}}^{a_1} \\ \vdots & \ddots & \vdots \\ \boldsymbol{H}_x^{a_M} & \cdots & \boldsymbol{H}_{l_{a_M}}^{a_1} \end{pmatrix} \tag{5.9}$$

is needed together with the joint state covariance matrix

$$\Sigma_{\mathcal{H}} = \begin{pmatrix} \Sigma_{xx} & \Sigma_{xl_{a_1}} & \cdots & \Sigma_{xl_{a_M}} \\ \Sigma_{l_{a_1}x} & \Sigma_{l_{a_1}l_{a_1}} & \cdots & \Sigma_{l_{a_1}l_{a_M}} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{l_{a_M}x} & \Sigma_{l_{a_M}l_{a_1}} & \cdots & \Sigma_{l_{a_M}l_{a_M}} \end{pmatrix}. \tag{5.10}$$

The off-diagonal entries involving two different landmarks (e.g. $\Sigma_{l_{a_M}l_{a_1}}$) encode the correlation between them, and is what separates JC from IC. Including the definition of the concatenated measurement noise for the measurements in the hypothesis $\mathcal{H}$: $\Lambda_{\mathcal{H}}$, the resulting covariance matrix for the innovation vector (5.8) is then found to be

$$\boldsymbol{S}_{\mathcal{H}} = \boldsymbol{H}_{\mathcal{H}} \Sigma_{\mathcal{H}} \boldsymbol{H}_{\mathcal{H}}^{\mathsf{T}} + \boldsymbol{\Lambda}_{\mathcal{H}}. \tag{5.11}$$

In summary, the hypothesis $\mathcal{H}$ deemed jointly compatible if it passes the JC test

$$\vec{v}_{\mathcal{H}}^{\mathsf{T}} \boldsymbol{S}_{\mathcal{H}}^{-1} \vec{v}_{\mathcal{H}} \sim \chi_k^2 < \gamma, \tag{5.12}$$

where $k = dim(\vec{v}_{\mathcal{H}})$.

5.6 Joint compatibility branch and bound

The added features of JC come with a cost however, in addition to the potential $\Pi_{ic} = N \times M$ evaluations to find IC pairings in $O(NM)$, the number of potential evaluations for JC for is

$$\Pi_{jc} = \frac{(N+1)!}{(N+1-M)!} \tag{5.13}$$

which is of time complexity $O(N^M)$[29]. To reduce the number of evaluations, Neira and Tardós [42] applies a branch and bound (BB) search on the hypothesis space, in what they call the JCBB algorithm. To enable the BB search, the hypothesis space is structured in an *interpretation tree* (see figure 5.2). Each level of the tree corresponds to a measurement, and nodes $\mathcal{N}_j = \{n_i\}_{i=0}^{N}$ at level j signify the association hypothesis of measurement $\vec{z}_j$ to landmark $\vec{l}_i$. For $i = 0$, no landmark is selected, and so denotes the null-hypothesis from eq. (5.1). The objective of the search performed in JCBB is to end up with the hypothesis $\mathcal{H}^{best}$ with the highest score: the number of non-null jointly compatible pairings. At the start of the search, $\mathcal{H} = \emptyset$ and the node signifying the IC pairing between the measurement $\vec{z}_1$ and the landmarks with the lowest residual (5.3) is added to $\mathcal{H}$ first. This is also the case every time the search goes down one level in the tree. To *bound* the search, at each node the current hypothesis $\mathcal{H}$ is evaluated on two criteria. For one, the number of levels in the tree below the current node plus the number of non-null hypotheses in $\mathcal{H}$ has to be greater than the score of the current best hypothesis $\mathcal{H}^*$. Second, $\mathcal{H}$ has to be JC in accordance with (5.12). If either criterion is not met, the node and its children are pruned from the interpretation tree. If at any point the search discovers a hypothesis with the same score as the current best, the one with the smallest Mahalanobis distance (left-hand side of eq. (5.12)) is considered the best moving forward. When all nodes have either been visited or removed from the tree, the best hypothesis is found as $\mathcal{H}^{best} = \mathcal{H}^*$, and JCBB is done.

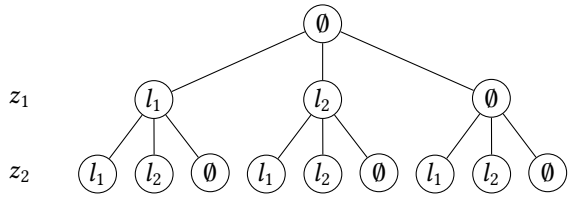


Figure 5.2: Interpretation tree of the JCBB solution space. In this example there are two detections, each corresponding to a level in the tree. Because there are two landmarks, each association hypothesis (node) is is a pairing between measurement z_j (at level j) to either l_1, l_2 or $\emptyset$ (null hypothesis).

5.7 Data association in iSAM2

It follows from equations (5.4) and (5.10) that it is integral to the ML, IC and JCBB data association schemes presented in this chapter to access to the covariances of the landmarks and poses in the state space. The authors of iSAM2 present an efficient solution to covariance recovery on the SRIM for their iSAM1 algorithm in [29]. In this article, GNN and a global ML association scheme is compared to JCBB on execution time, and the accuracy of JCBB on the Victoria park dataset [58] is validated. The described method in [29] does not consider the introduction of the Bayes tree on which iSAM2 is based, but as the SRIM is common in both algorithms, Kaess et al. [31] states that the means of covariance retrieval is conceptually equivalent. Wang and Englot [59], propose a solution for further reducing data association ambiguity by maintaining multiple JCBB-based hypotheses over time as the one with the highest score is chosen. Their implementation is using iSAM2 as the SLAM backend, from which joint covariance recovery is performed. The authors concede that the computational effort is substantial when compared to considering a single hypotheis at each iteration, which ruled out using a similar scheme for the task in Revolve. Still, ideas from their implementation of JCBB with iSAM2 has been welcome.

Chapter 6

Detection and odometry

This chapter briefly presents the methods used for supplying the data entering the SLAM module in the autonomous pipeline. These components acquire and process the raw data from relevant sensors on the vehicle, before passing condensed information on to SLAM.

Figure 6.1: ATMOS with the sensor package from 2019

6.1 Lidar detection

The *lidar detection* module is responsible for extracting cone candidates from a raw point cloud output from the lidar sensor(s). A lidar operates by emitting light beams and measuring the time, wavelength and intensity of the reflected signals. This way, the range to and relative reflectivity of the target can be identified. A common type of lidar used in the autonomous industry (and in Revolve) is the spinning "cylindrical" lidar. These types of lidars have multiple *channels* in the vertical direction that each corresponds to a recorded signal. Because the sensors spin, these vertical channels are sampled in a 360° field of view, with a sample rate or resolution dependent on the particular model. In any case, the sensor output is a dense point cloud which is generated at every sensor revolution. The lidar used in Revolve in 2019 was the Ouster OS-1 64-channel [45]. For 2020 the team had acquired the Hesai Pandar40 40-channel lidar [27] for use in development and competitions.

For finding potential cones in the point cloud, a detection algorithm is employed using a set of filtering techniques alongside clustering and outlier rejection logic. Fellow Revolve-member Benjamin Palerud has implemented the lidar detection algorithm, and the following section is an excerpt from his master's thesis [46].

The first step is to filter out irrelevant data. Because the point clouds are sparse at longer distances, the maximum possible detection range is limited. Also, as the topology of the racetrack is consistently flat, a filter is applied to remove any points outside user-supplied xyz-constraints. Secondly a *voxelgrid* downsampling is performed. This downsampling works by dividing the point cloud into boxes (voxels) with a set size. All points situated inside a voxel is represented by one point: the centroid. These two steps reduce the size of the point cloud, which results in faster processing time for the clustering.

The euclidean clustering is based on specific conditions to evaluate the point cloud. These include the maximum and minimum amount of points that can be in a cluster, the threshold d_{th}, and a condition that says that points from at least two different lidar channels have to be present in a cluster. The last condition is to ensure that a certain height is preserved in the clustering, and that clusters with only horizontal points are

Procedure	Tuning parameter
1. xyz-filter	(x, y, z) - constraints
2. Voxelgrid downsampling	Voxel size
3. Clustering	Min/max points, threshold d_{th}
4. Outlier removal	Radius

Table 6.1: Tuning parameters

not evaluated as candidates. When the clustering finds a candidate, the width and the height between the outermost points in the cluster is checked to ensure it is within a reasonable range based size of the cones. Finally, the set of points representing a cluster is represented by the centroid of the cluster.

Lastly, a filter performs outlier rejection to remove candidates that are too close to each other. The basis for imposing this filtering step is that cones are spaced out with a certain distance to each other on the racetrack. The filter checks how many candidates that are within a certain area and neglecting them if there are too many. When a cone candidate is found, the point representing a cluster is reconstructed by taking the raw data from the input point cloud within a radius of the candidate. The intensity pattern of the raw data related to this candidate can then be used to distinguish the color of the particular cone.

The performance of the detection algorithm is very dependent on the user-defined parameters that need to be tuned to ensure good candidate localization. The detection sequence and tuning parameters are summarized in table 6.1. The voxel size and the clustering parameters are particularly dependent on each other, and are the main parameters that define whether the cone candidate extraction is successful. For instance, by increasing the voxel size, which results in a greater distance between the points, the threshold d_{th} might need to be increased in order to find relevant clusters. The detection algorithm is implemented in C++ as a ROS node which subscribes to the raw point cloud output from the lidar ROS-driver, and publishes messages to the SLAM node. All the filtering techniques and the clustering is implemented by using the Point Cloud Library (PCL)[49].

6.2 Odometry

In Revolve, an extensive array of sensors are used to measure *internal states* of the car. Some sensors are used for their direct output data, such as thermometers and voltmeters used to monitor the batteries. Readings from potentiometers that measure the positioning of dampers and the acceleration and braking pedals are also directly used. The latter two, together with the data from a rotary encoder on the steering wheel are treated as measurements of the inputs to the system. In many cases, the states we wish to know about are not directly measurable and have to be *estimated*. Additionally, measurements from sensors are without exception prone to errors and noise that can impact estimates. As an example, a rotary encoder on a wheel outputs the angular position of the wheel. Differentiating this signal yields the angular velocity of the wheel, which together with the known wheel radius is used to find the wheel's translational velocity ($v = r\omega$). If the tire connecting the wheel to the ground is exposed to slip, the tangential velocity at the outer radius of the wheel is no longer equal to the translational velocity of the wheel hub, and the velocity estimates of whatever the hub is connected to (racecar!) are wrong. In 2018, the newly founded DV team at Revolve relied solely on the encoders at the wheels to estimate the velocity and heading of the vehicle. At low speeds and on dry tarmac, the slip issue is almost nonexistent, and so it was a viable option for them.

As ambition levels rose in the following year, the move towards integrating a state-of-the-art *inertial navigation system (INS)* was a priority. Accordingly, the Vectornav VN-300 [57] was fitted to Atmos, and is a GPS-aided INS that can output estimates of position, velocity, acceleration and orientation using an internal *inertial measurement unit (IMU)*. Last year's team experienced some issues with acquiring correct output from the device, while also wanting a system that could perform well in the absence of consistent GPS-signals. Thus, previous Revolve member Adrian Skibelid implemented a nonlinear *state observer* that generates longitudinal, lateral and yaw (x, y, θ) velocity estimates by combining measurements of translational acceleration and angular velocity from the INS with the data from the wheel encoders. A detailed description of this system is found in [51]. The velocity estimates generated by this observer are

integrated over time to output vehicle pose estimates. In [51], the observer is shown to be accurate over short periods, but as no GPS corrections are applied to the positional estimates, drift accumulates over time. The SLAM module combines the data from the detection and odometry modules to combat this drift, as well as to further improve the short term pose estimation of the vehicle.

Chapter 7

Implementation

7.1 Software

The autonomous pipeline in its entirety is deployed in ROS Kinetic Kame[44] on Ubuntu 16.04. Most of the code, including all SLAM specific code, is written in C++. The open-source Eigen library [25] is used for vectors, matrices and their related linear algebra. The ISAM2 backend, along with the probabilistic data associations schemes from chapter 5, is implemented using the factor graph framework supplied by GTSAM in [9].

7.2 System overview

The implementation of the SLAM module in Revolve follows the same two-component structure as presented in chapters 4 and 5. A graphical illustration of the structure of the SLAM implementation is displayed in figure 7.1 The frontend handles the incoming data from the detection and odometry modules, which then is processed and passed to the backend for calculating MAP estimates of the compound state vector $\Theta = (X, L)$. The odometry module outputs pose estimates at around 200 Hz, but only a fraction of these are incorporated into X. The reason for excluding some poses is that finding the

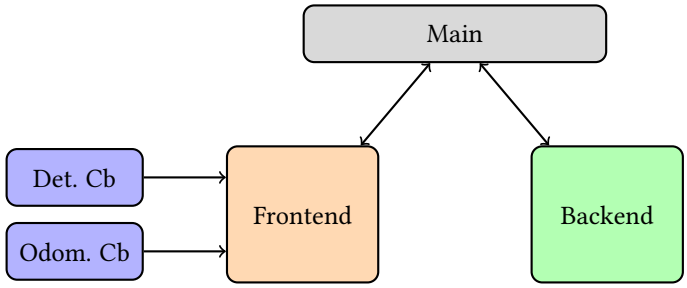


Figure 7.1: SLAM system overview. The blue boxes are the callbacks triggered by ROS messages. The "Main" component is responsible for initializing the ROS node, and giving the shared resources between the frontend and backend

MAP estimates for 200 states per second put an unnecessary strain on the backend. Keeping execution time low is a vital criteria in autonomous high-speed racing, and as detections enter at a much slower rate of 20 Hz, keeping the extra intermediate poses yield little if any, accuracy benefits. The states that become a part of X is commonly referred to as *keyframes*. The resulting MAP output is the set of estimated inertial poses in 3DOF

$$\hat{X} = \{\hat{\vec{x}}_k\}_{k=0}^{K} \in \mathcal{I}, \tag{7.1}$$

for time steps k, and estimated 2D locations of the cones in the map:

$$\hat{L} = \{\hat{\vec{l}}_i\}_{i=1}^{N} \in \mathcal{I}. \tag{7.2}$$

To make the distinction clear between the output pose estimates from SLAM and the input pose estimates from the odometry, tilde notation is used for the latter:

$$\tilde{X} = \{\tilde{\vec{x}}_t\}_{t=0}^{T} \in \mathcal{I}. \tag{7.3}$$

Both the frontend and backend operation run in parallel to ensure that both tasks are given the necessary computational resources. These processes are encapsulated by the SLAM ROS node which is also responsible for handling message passing to and from the SLAM system. In ROS, users define specific topics that each carry a certain type

of data. Nodes can then be configured to *subscribe* to relevant topics for input data and *publish* their output data to other topics. Figure 4.1 displays the topics involving the SLAM node as arrows, with the publisher-subscriber relationship indicated by the arrows' direction. For data being passed between the two parallel threads in the SLAM module, *race conditions* are avoided through the use of a shared set of data protected by mutual exclusion objects (mutexes).

The SLAM system operates in two modes: *mapping and localization*, and *localization* only. If the map is unknown before the run starts, such as in the autocross event (see section 2.2), mapping is naturally needed. When the vehicle completes a lap, i.e. closes the loop, the state changes to localization only. This is also the case when the map is known beforehand, as in the skidpad and (possibly) trackdrive events. In localization mode, the assumption that all cones have been found is made, but that their positions are still affiliated with some uncertainty. Because of this, measurements are continuously added to the optimization in the backend to update and decrease ambiguity in the cones' positions.

7.3 Frontend structure

7.3.1 Input - Messages

Odometry

The main structure in the frontend is comprised of two message-triggered callback functions (odometry and detection) and a continuously looping thread. The callback triggered by odometry estimates is the most straightforward of the two. In summary:

1. Message type: ROS Odometry[1]. Important contents: [timestamp, pose estimate]
2. Messages enter at a constant rate of 200 Hz.
3. The last second of messages ($\approx$ 200) is stored in memory (used for matching detections to closest pose).

[1]link: `http://docs.ros.org/kinetic/api/nav_msgs/html/msg/Odometry.html`

4. If the car pose has changed more than certain amounts in translation or heading angle since the previous keyframe, a new keyframe is generated.

5. A number of the most recent keyframes are stored in a buffer in the frontend (again for matching to detections).

6. Keyframes are passed to the backend, where they are added to the factor graph and optimized. See section 7.8 for details on this.

Detections

The detection callback is triggered by either the lidar or camera detection modules sending messages to the shared "detections" ROS topic.

1. Message type: Revolve custom "Obstacle" message. Contents: [timestamp, list of 2D points, sensor]

2. Messages enter asynchronously from either detection sensor node. The frequency is a tuning parameter, but the max is about 20 Hz for each.

3. By using the timestamps, detections are matched to the closest pose estimate in *time* and are then defined as originating from the vehicle at the given pose. The assumption made here is that pose estimates at 200 Hz are frequent enough that the difference between exteroceptive sensor timestamps and their closest pose estimates are sufficiently low and random to assume zero mean distributed errors. These uncertainties will also be implicitly accounted for in user-defined noise parameters discussed later.

4. When the closest pose estimate is found, the next step is to transform points in space to the frame of the closest pose keyframe. The closest pose and pose *keyframe* estimates of detection $\vec{z}_k$ are denoted $\tilde{\vec{x}}_k$ and $\tilde{\vec{x}}_k^{kf}$, respectively. The transformation between the poses is found as

$$\boldsymbol{T}_x^{x^{kf}} = \boldsymbol{T}\left(\tilde{\vec{x}}_k^{kf}\right)^{-1} \boldsymbol{T}\left(\tilde{\vec{x}}_k\right). \tag{7.4}$$

The detection is finally transformed to the closest keyframe:

$$\vec{z}_k^{kf} = \boldsymbol{T}_x^{x^{kf}} \vec{z}_k. \tag{7.5}$$

From this point forward, all detections are assumed given in the pose keyframes, and so the superscript kf is not explicitly written to reduce clutter.

5. The transformed points, together with the keyframe (id and pose) are then stored in a fixed-size last in, first out (LIFO) queue for processing by the frontend loop thread. The reason for using a LIFO queue is so that the most recent detection is prioritized, while still allowing for past data to be processed if there is available computational headroom.

7.3.2 Main thread

The looping thread is responsible for the majority of the tasks performed in the frontend. This includes data association to the preexisting map, the generation, maintenance and deletion of possible cone candidates, data association to candidates and tracking of cone types (colors). "Cone candidates" described in this section differ slightly from the definition in chapter 6 which are in this chapter here referred to as detections or measurements. Here, a cone candidate is an object that is kept in memory and has a confidence level associated with it. Once the confidence in the candidate reaches a specified threshold, it is considered to be a cone.

Below, the sequence of actions performed by the looping frontend thread will be presented, while the specifics of key elements such as the data association schemes come later.

1. Save the latest MAP estimates found in the backend.

2. Keep track of which lap the car is in, and update the vehicle state if necessary (mapping/localization only).

3. Perform data association on the most recent set of detections, as stored in the detection callback function, to the current map of cones already added as

variables in the backend optimization.

4. If the vehicle is in mapping mode: continue to step 5, otherwise: return to step 1.

5. For the detections that were not associated to any landmarks in step 3, attempt to associate them with the current set of stored cone candidates.

6. If a detection is associated to a candidate, the candidate's position is updated, and its confidence is increased.

7. All candidates not being associated to any detections in this iteration of the loop has their confidence decreased.

8. The remaining set of detections that are not yet associated to either the map or the cone candidates are initialized as new cone candidates.

7.4 Data association

In this section, the specific implementations of the data association schemes introduced in chapter 5 will be presented.

7.4.1 Sequential nearest neighbor (SNN)

To differentiate global association schemes using assignment algorithms such as the Hungarian method [35] or Jonker, Volgenant and Castanon (JVC) [7] assignment, with the *sequential* approach taken here, this method is termed sequential nearest neighbor (SNN). The procedure is summarized in Listing 7.1. The reason for randomizing the order of the input is to eliminate any potential bias in the ordering of incoming detections. When finding the closest landmark, the detection is momentarily transformed from the body to the inertial frame (superscript notation) using the MAP estimated pose of the keyframe from which it was captured:

$$\vec{z}_k^{\,i} = \boldsymbol{T}(\hat{\vec{x}}_k)\vec{z}_k^{\,b}. \tag{7.6}$$

On line 11 in Listing 7.1, the compatibility of a detection and a landmark is determined by the closeness of the two. Based on competition regulations and the layout of the track to be driven, this threshold can be tuned accordingly. Primarily, this threshold prevents associations between cones in the map and detections of cones that have not yet been added to the map, but also filters out FPs not belonging to any cone (random clutter) outside this radius. The problem with this approach is that when the magnitude of the accumulated drift after the first "mapping" lap is larger than the threshold value, making wrong associations can happen very easily. Also, as evident from the results presented in section 8.4, loading the map prior to a run can cause difficulties for SNN.

```
1  // Input : Detections
2  // Output: Associations between detections and landmarks
3
4  associations = []
5  randomize(detections);
6  for (auto z : detections) {
7      Landmark lm = findClosestLandmark(z, map);
8      if (lm.isAssociated = true) {
9          continue;
10     }
11     else {
12         if(distance(z, lm) < acceptanceRadius) {
13             lm.isAssociated = true;
14             associations.push_back(z, lm);
15         }
16     }
17 }
```

Listing 7.1: C++ pseudocode for SNN procedure

7.4.2 Individual compatibility (IC)

As detailed in sections 5.3 and 5.4, the compatibility of measurements to landmarks can be determined using the marginal distributions on the landmarks and pose. To recover these quantities, listed in eq. (5.4), the GTSAM library supplies the user

with a set of tools. The diagonal elements of eq. (5.4): marginal covariances of the pose and landmark ($\Sigma_{xx}, \Sigma_{l_i l_i}$), can be fetched directly using the iSAM2 specific function `ISAM2.marginalCovariance(variable)`, where the *variable* parameter is either the pose keyframe or the landmark. Retrieving the off-diagonal elements requires more processing, summarized in Listing 7.2. In the case for determining individual compatibility between pose $\vec{x}_k$ and landmark $\vec{l}_i$, the list of variables in Listing 7.2 would only include these two.

```
1  // Input : List of variable keys (x_1, l_1 etc.)
2  // Output: Full joint covariance matrix of variables
3
4  Matrix jointCovariance(keys) {
5      factorGraph = ISAM2.getFactors();
6      estimates = ISAM2.getLinearizationPoint();
7      // GTSAM Marginals class (intermediate step)
8      marginals = Marginals(factorGraph, estimates);
9
10     return marginals.jointCovariance(keys);
11 }
```

Listing 7.2: C++ pseudocode for recovering joint covariance matrices from ISAM2, including cross-covariances between different random variables

The joint covariance matrix returned from the procedure in Listing 7.2 includes the same diagonal elements as found using the `ISAM2.marginalCovariance()` function, meaning that performing both procedures is not necessary. In sections 8.2 and 8.3 it is shown that foregoing the off-diagonal elements is an acceptable measure for reducing computation times while maintaining performance.

In addition to the covariance matrix, the innovation vector (residual) and measurement Jacobian is needed to evaluate IC, as per equations (5.6) and (5.7). The implementation of SLAM in Revolve is confined to the two-dimensional plane, and so the measurement model for the single predicted *bearing-range* measurement $\hat{\vec{z}}_j$, given

estimated pose $\hat{\vec{x}}_k = (x, y, \theta)^\mathsf{T}$ and landmark $\hat{\vec{l}}_i = (l_x, l_y)^\mathsf{T}$ is given as:

$$\hat{\vec{z}}_j = h(\hat{\vec{x}}_k, \vec{l}_i) = \begin{pmatrix} \hat{\beta} \\ \hat{r} \end{pmatrix} = \begin{pmatrix} \text{atan2}\left(\frac{l_y - y}{l_x - x}\right) - \theta \\ \sqrt{(l_x - x)^2 + (l_y - y)^2} \end{pmatrix}. \tag{7.7}$$

The next step would then be to calculate the innovation vector $\vec{v} = \hat{\vec{z}}_j - \vec{z}_j$ and evaluate the Jacobian of eq. (7.7) at the estimated pose and landmark location as given above. However, when using GTSAM no explicit expression for either the measurement model, nor its Jacobian, is needed. By supplying the framework with a factor containing the type and linearization points of two variables (2D pose and point in this case), and a measurement connecting them, it can evaluate the residual and Jacobian for us. To summarize, Listing 7.3 proposes how determining IC between a landmark and a measurement using GTSAM and iSAM2 could be done. The measurement noise parameter in Listing 7.3 is discussed in detail in section 7.8.4.

7.4.3 Sequential compatibility nearest neighbor (SCNN)

The only separating factor between the implementations of SNN and sequential compatibility nearest neighbor (SCNN) is the replacement of line 11 in Listing 7.1 with the function presented in Listing 7.3. This method was implemented to bridge the gap between the basic SNN approach and the more comprehensive probabilistic methods described next. The desire to keep execution time low means that this method does not use the off-diagonal elements of the covariance matrix when performing the IC test.

7.4.4 Maximum likelihood (ML)

Instead of associating measurements to the landmarks to which they are closest to in 2D space, the ML approach is based on the Mahalanobis distance metric. In theory, this involves checking IC between every measurement and every landmark, a total of $N \times M$ times. To reduce the number of evaluations, the implementation of ML-data association only considers landmarks within a certain radius of measurements. If

```
// Input : Landmark lm, Pose x,
//         Detection z, Probability p
// Output: Boolean: Individually compatible?

bool IC(lm, x, z, p) {
    marginalPose = ISAM2.marginalCovariance(x);
    marginalLandmark = ISAM2.marginalCovariance(lm);
    // Covariance matrix
    Sigma = diag(marginalPose, marginalLandmark);

    factor = BearingRangeFactor(x, lm, z);
    // Innovation vector
    nu = factor.evaluateError();
    H = factor.Jacobian();
    Lambda = z.noise;
    S = H*Sigma*H.transpose() + Lambda;

    mahalanobis = nu.transpose()*S.inv()*nu;
    if (mahalanobis < chi2inv(p, dim(lm)) {
        return true;
    }
    else return false;
}
```

Listing 7.3: C++ pseudocode for IC test between landmark lm and measurement z made at pose x using GTSAM. The $chi2inv(\cdot)$ function returns the critical value c in eq. (3.24) for a given probability p and degrees of freedom $dim(lm)$

this radius is set larger than the upper bound of any potential drift the odometry estimates will experience in the first lap, accuracy will not be affected in any way. If the radius is sufficiently large, it is also likely that landmarks will fall within this region of multiple measurements simultaneously (see the bottom left landmark in Figure 7.2 for an example of this). Instead of then having to recover the marginal covariances (as in Listing 7.3) for a landmark and pose directly from iSAM2 multiple times, the matrices are stored temporarily in memory for subsequent look-ups. When all checks are done, the measurement-landmark pairings with the smallest Mahalanobis distance that satisfy IC are chosen and sent to the backend.

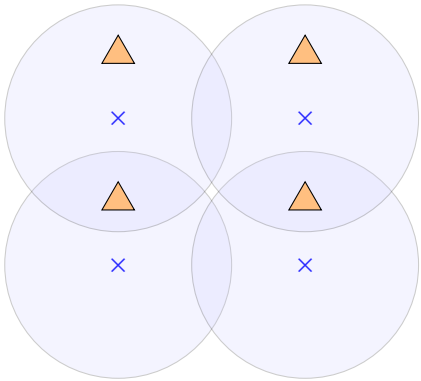


Figure 7.2: Orange triangles indicate landmarks, blue crosses are measurements, and the shaded blue circles are the evaluated region for measurements in ML and JCBB data association.

7.4.5 Joint compatibility branch and bound (JCBB)

A condition for the associations evaluated in JCBB is that they all satisfy IC. As with the ML-data association, landmark-measurement pairings that are too distant from each other are not evaluated. When the IC check is being performed, the residuals and evaluated Jacobians are stored, so that they can efficiently be combined into the joint innovation vector and Jacobian matrix used for JC checks (see equations (5.8) - (5.12)) during the branch and bound search. For the joint covariance matrix, the procedure in Listing 7.2 is performed with additional variables in the list of keys. The JC-check is done in a similar fashion to the IC-check in Listing 7.3, only with using *joint* covariances instead of *marginal* ones, and that the innovation and Jacobian matrices are not calculated but instead retrieved from memory.

The high-level version of the JCBB implementation is displayed in Listing 7.4. During testing, it was found that because the number of landmarks is somewhat limited, recovering the full covariance matrix for all landmarks once, instead of doing more frequent but smaller recoveries, was the optimal solution. The basis for making this choice is presented in the results in section 8.3. For details on how the branch and bound search is performed, see section 5.6.

```
1  // Input : Pose x, map with landmarks, set of measurements
2  // Output: Best hypothesis
3
4  Hypothesis JCBB(x, map, measurements) {
5      fullSigma = jointCovariance(x, map) // From Listing 7.2
6      associations = [];
7      for (auto z : measurements) {
8          for (auto landmark : map) {
9              if (distance(landmark, z) < evaluationRadius) {
10                 if (IC(landmark, z, x, fullSigma)) {
11                     associations.push_back(landmark, z);
12                 }
13             }
14         }
15     }
16     it = buildInterpretationTree(associations);
17     return BranchAndBound(it, fullSigma);
18 }
```

Listing 7.4: C++ pseudocode for the implementation of JCBB. The IC check here does not calculate the covariances as in listing 7.3, but performs look-ups in the full joint covariance matrix fullSigma.

7.5 Candidate management

The measurements that do not get associated to a landmark in the map are not immediately added to the map as new landmarks. The reason for this is the potential presence of spurious measurements, known as false positives. To account for this, detections that are not associated to either the map or the existing set of candidates, are initialized as *cone candidates*. The main components of a cone candidate are its estimated position, its confidence level and the set of measurements that has been associated to the candidate.

Whenever a cone candidate is re-observed, its confidence level is increased and the measurement together with the pose keyframe from which the measurement was made, is stored in the candidate. The candidate's position is also updated using a

moving average. The moving average positional updating implemented as

$$p_k = avg(p_{k-1}, z_k), \tag{7.8}$$

was chosen because it is simple, and it naturally weights recent measurements more. If a candidate is not observed in a set of detections, its confidence is lowered. For pairing measurements to candidates, the SNN approach is employed regardless of the data association scheme used for landmark associations. Because candidates do not stay in memory for very long, the impacts of positional estimation errors and drift are assumed insignificant. As a consequence, the acceptance radius is set lower than for measurement-landmark associations. The procedure for candidate management is presented in Listing 7.5. Here, the existing set of candidates is updated with new measurements, and the unassociated measurements are initialized as new cone candidates. The reason for accumulating measurements is so that when the candidate is passed to the backend, all measurements can be added to the factor graph for optimization.

The amount of false positives and false negatives arriving as input to the SLAM node is directly influenced by the type and tuning of the feature extraction. Because of this, the values for incrementing, decrementing, acceptance-/rejection thresholds as well as the starting confidence are tuned accordingly. The tuning of these parameters will also be based on the preferences of the path planning module. Adding candidates quicker to the backend allows for longer ranges, but the risk of adding false positives is larger. Typically, this is something that would be done more thoroughly in the summer testing, when the newly developed concepts for the different systems are tested together.

7.6 Color tracking

Although the driverless team has managed to have well-performing cars for two years without recognizing cone colors, keeping track of colors is in the best interest of the path planning module. Having access to this information allows the path planner to be

more confident that the planned path is within the track delimiters, as blue and yellow cones signal the left and right sides of the track respectively. For the lidar especially, the range at which color detection of cones is reliable is considerably lower than for detecting cone positions (see Palerud [46] for more in-depth material). Because of this, cones are added to the map before colors are determined accurately, and the color output by the lidar is set to "unknown" outside a given range. As the path planning prefers having no color instead of the wrong color, the SLAM module keeps track of all color estimations on each cone, and sets the color based on which is the most frequently occurring. Color detections originating from the camera is given more weight because cameras are better suited for telling which color a cone is, and is very unlikely to output a wrong color.

7.7 Frontend parameters

- **Keyframe generation distance:** The maximum distance between keyframe poses. Set to 0.5 m to balance granularity with limiting the number of states in the backend.

- **Keyframe generation rotation:** Same as above, but is a separate condition. In sharp turns, the heading angle will change fast, and pose keyframes are needed. Set to 2.5°.

- **ML/JCBB evaluation radius:** Landmarks that fall within a set radius of a measurement is considered for the IC-check. Imposed heuristic to reduce the amount of expensive covariance recoveries. Set to 3.0 m based on testing, but will need to be tuned according to the breadth of the track and the magnitude of drift on odometry estimates.

- **Chi-square probability threshold:** For which probability the critical value for the chi-squared distribution should be set. Relevant for the IC and JC checks. Set to 0.9 for both based on testing results.

7.8 Backend

The SLAM backend consists of a single thread iterating indefinitely. On each iteration, if new data is added to the factor graph, iSAM2 incorporates the new factors into the Bayes tree and calculates the MAP estimates of the state variables by performing the GN updating from section 4.5 a set amount of iterations. Data passed from the frontend is divided into three categories: Pose keyframes, new landmarks and measurements of existing landmarks.

7.8.1 Pose keyframes

When the most recent pose keyframe $\tilde{\vec{x}}_k$ is passed from the frontend, the relative pose between it and the previous keyframe $\tilde{\vec{x}}_{k-1}$ as defined in

$$\tilde{\vec{x}}_k = \tilde{\vec{x}}_{k-1} \oplus \delta\vec{x}, \tag{7.9}$$

is stored in a GTSAM `BetweenFactor()`. The initial estimate for the variable used as the starting linearization point in the optimization is set equal to the MAP estimate of the preceding vehicle pose composed with the same difference:

$$\hat{\vec{x}}_k = \hat{\vec{x}}_{k-1} \oplus \delta\vec{x}. \tag{7.10}$$

The use of hat-notation here signals that these estimates are processed by the MAP estimation in the SLAM backend, for which reason they can be expected to more closely follow the true vehicle pose $\vec{x}_k$. As with all factors, a noise model has to be imposed on the residual (Σ_i from eq. (4.14)). As the number of keyframes added to the backend is fewer than the odometry estimates input to the SLAM node, the propagation $\delta\vec{x}$ is a sum of differences from n subsequent odometry outputs. Noise parameters on the normally distributed data coming from the odometry can be assumed to be constant and diagonal ($\Sigma = diag(\sigma_x^2, \sigma_y^2, \sigma_\theta^2)$), which results in the naive model of the Gaussian distribution of $\delta\vec{x}$:

$$\delta\vec{x} \sim \mathcal{N}(\delta\vec{x}, n \cdot \Sigma). \tag{7.11}$$

In practice, an even simpler approach is implemented. Because keyframes are generated based on the traversed distance and change in orientation, and not based on the number of odometry estimates, the noise is assumed constant for all $\delta\vec{x}$ regardless of which number n is. Although mainly done for simplicity, this approach reduces the risk of overparameterizing the approach. And while this is something that can be made more advanced, extensive tuning needs to be done depending on the specific implementation of odometry.

7.8.2 New landmarks

The tracked cone candidates in the frontend have to have a certain amount of accumulated measurements for them to be sent to the backend. For each of the new landmarks to be added, a new variable is added to the variable set L with its estimated position set as the initial linearization point. Furthermore, all accumulated measurements for the particular landmark are added to the factor graph as GTSAM `BearingRangeFactors()`. As a result of eq. 7.5, the measurement points are given in the reference frame of their associated pose keyframe. The only processing needed is then to convert the points from Cartesian to polar coordinates using eq. (3.2), where r is range, and θ is bearing. For the new landmark and associated pose keyframe index k, the resulting measurement factor connects the variable pair $(\hat{\vec{x}}_k, \hat{\vec{l}}_i)$.

The uncertainties of measurements are encoded with a constant diagonal covariance matrix on the range and bearing values. An overview of important tuning parameters, including measurement and motion noise values, are listed in section 7.8.4.

7.8.3 Measurements

Similarly to adding new landmarks, including new measurements of existing landmarks entails the generation of `BearingRangeFactors()` between the landmark and the pose keyframe from which the measurement was made. The correspondence between the measurement and landmark is performed in the frontend by the data association schemes presented in section 7.4.

7.8.4 Parameters

Below is a list explaining notable tuning parameters related to the SLAM backend.

- **Motion noise:** Correlated with keyframe generation in the frontend, as fewer keyframes should result in larger uncertainties between them. By tuning this parameter in conjunction with the measurement noise, a weighting is performed on the system's trust in cone measurements vs motion inputs. In the current system, tuning was performed by visually validating the output from SLAM running together with the lidar detection and odometry modules processing actual sensor data. When generating pose keyframes with a 0.5 m distance and/or 2.5° heading between then, reasonable motion noise was found to be

$$\Sigma = diag((0.1\,\text{m})^2, (0.1\,\text{m})^2, (0.01\,\text{rad})^2).$$

- **Measurement noise:** Should encapsulate all unmodeled noise relating to measurements. For a stationary vehicle, this mostly reduces to the intrinsic noise parameters of the exteroceptive sensors. The experiment presented in section 8.5 was performed to estimate these parameters. However, the results from this experiment suggested parameters that were not transferable to the case of a moving vehicle. When the vehicle is no longer stationary, there are potentially many sources of noise that gets magnified. One example of this is the mismatch of pose timestamps to detection timestamps. From section 7.3, it was stated that because odometry estimates are much more frequent than detections, any time mismatches are disregarded. To illustrate the potential impact this may have on a measured cone position, consider a racecar driving 20 m/s on a straight. With pose estimates and detections entering at 200 Hz and 20 Hz respectively, the maximum time mismatch is 2.5 ms. This leads to a 5 cm error in the x-direction for all cone measurements. If angular velocity in yaw (heading) is included, even larger errors are introduced. To summarize, the noise parameters should encompass the potential errors happening under non-ideal conditions. Together with the motion noise parameters set above, a good approximation was found

to be

$$\Lambda = diag((0.05\,\text{m})^2, (0.15\,\text{rad})^2).$$

- **Prior pose noise:** When preloading maps, the only task of SLAM is to localize the car within the map. Competition rules limit the potential discrepancies in starting positions, but precisely setting the initial vehicle heading (yaw) is up to the team members. To account for a possible disparity between the map frame (in which the map is defined), and the initial body frame of the vehicle a prior noise is set on the GTSAM `PriorFactor()` defining the starting pose of the vehicle. If prior knowledge is included and the SLAM system is responsible for building the map, the prior noise is set to zero because the starting pose of the vehicle defines the map frame. No real-life testing was performed to set the prior noise, and it was deemed justifiable to set

$$\Sigma_0 = diag((0.5\,\text{m})^2, (0.5\,\text{m})^2, (0.26\,\text{rad})^2).$$

- **ISAM2 parameters:**
  - **Number of updates:** The number of iterations of GN optimization to perform each backend loop iteration. Set to **20**.
  - **Relinearizing threshold:** Variables that during GN updates have their Δ found to be of magnitude larger than this threshold, have their linearization points updated. Because relinearizing variables come at an extra computational cost, increasing this parameter will cause fewer variables in the Bayes tree to be updated. Set to the default **0.1**.
  - **Relinearize skip:** Defines the number of iterations of GN updates performed before relinearizing the variables (subject to relinearizing threshold). Set to the default **10**.
  - **Wildfire threshold:** If the update step Δ of a variable changes with magnitude less than this threshold from one iteration of optimization to the next, its descendants in the Bayes tree are not considered for updating.

Referred to as *partial state updates* in the iSAM2 article [31]. Set to the default value of **0.001**.

```
// Input : Pose x, detections, candidates
// Output: Updated candidate set

T = transformationMatrix(x);
for (auto z : detections) {
    // Transform z to map frame
    z_map = T*z;
    Candidate& cand = findClosestCandidate(z_map, candidates);
    if (cand.isAssociated = true) {
        Candidate newCand;
        newCand.pos = z_map
        newCand.measurements.push_back([z, x]);
        candidates.push_back(newCand);
    }
    else {
        if(distance(z, cand) < acceptanceRadius) {
            cand.isAssociated = true;
            cand.pos = avg(z_map, cand.pos);
            cand.measurements.push_back([z, x]);
            cand.confidence += increment;
            if (cand.confidenc > acceptanceThreshold) {
                sendCandidateToBackend(cand);
            }
        }
        else {
            Candidate newCand;
            newCand.pos = z_map
            newCand.measurements.push_back([z, x]);
            candidates.push_back(newCand);
        }
    }
}

for (auto cand: candidates) {
    if (cand.isAssociated == false) {
        cand.confidence -= decrement;
        if (cand.confidence < rejectionThreshold) {
            removeCandidate(cand);
        }
    }
}
```

Listing 7.5: Pseudocode procedure for maintaining cone candidates

Chapter 8

Testing and results

8.1 Test setup

The testing with ATMOS was planned to start in the middle of April. Seeing as the car was turned into a DV last year, testing time was according to the plan, going to be plentiful before the competitions in the summer. Due to the ongoing pandemic, all in-office and workshop Revolve operations halted, and so did hopes of performing any on-board testing of any part of the autonomous pipeline. Luckily, some testing was performed with the vehicle in its 2019 iteration during the fall of last year. The data from a full, ten-lap autonomous run is used throughout the results evaluations in this chapter. This dataset is originally a *rosbag* of raw sensor data, but after running the detection and odometry systems, the output is recorded in a new rosbag to use for direct input to SLAM. This dataset will be referred to as the Moholt data, because the testing was done on a parking lot at Moholt in Trondheim.

In addition to this, cone locations from the FSG 2018 trackdrive event is stored for use in a simulator. The simulator generates odometry with user-defined noise parameters in all 3DOFs of the vehicle: x, y and θ. Normally distributed noise can also be set on the bearing and range of all measurements.

Figure 8.1: Picture of ATMOS taken during the Moholt testing.

8.2 Performance

This section examines the accuracy of the SLAM system as a whole and compares it between the data association methods. As no ground truth is available for the Moholt data, the only *absolute* performance metric is the number of laps the system managed to do before it diverged. This data is presented for the Moholt run, as well as on the FSG simulation, in table 8.1. To simulate a vehicle driving faster, the Moholt bag was played back at 1.5× speed. The recorded top speed was with this modification around 40 km/h. The only method that managed to complete all ten laps is the SNN data association. The simple explanation for this is that it is the only method that keeps execution time low enough for the complete run. If the rosbag is played back at 0.5 speed, all methods complete ten laps. Execution time results are discussed in further detail in the next section.

When ground truth data is unavailable, comparing *relative* performance give an indication of the differences between the methods. Figure 8.2 shows the absolute value

Dataset	Number of laps completed			
	SNN	SCNN	ML	JCBB
Moholt	10	9	9	6
FSG	10	4	2	9

Table 8.1: These are the approximate averages over five runs for each method. The uncertainty of this number is about ±0.5 laps, but is difficult to pinpoint due to needing visual confirmation of divergence.

of the translational correction SLAM applies throughout the Moholt run. The point where SLAM diverged is also given for each method. Based on this data, it is hard to decide which method performes best in the middle of the run, but because divergence is something that gives zero points in the competitions, it is a deciding metric. In figure 8.3, the correction applied in the methods over the first three laps is displayed. Here, it is clear that all methods perform on a similar level.

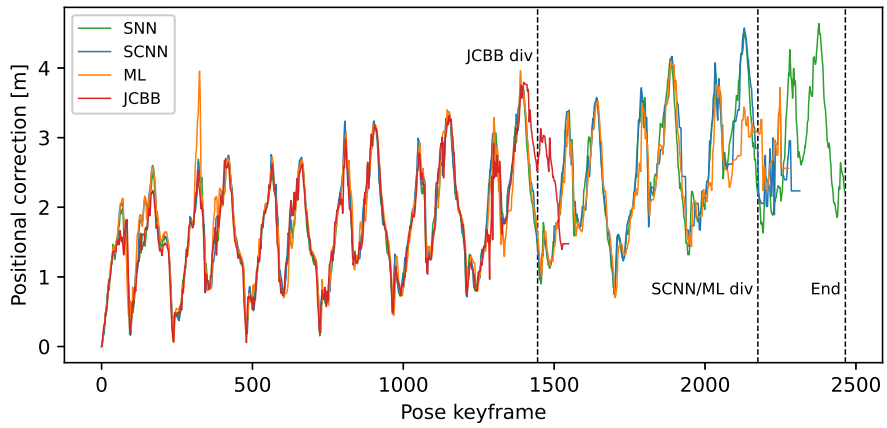


Figure 8.2: The total positional correction applied on the odometry input, as calculated by the SLAM backend. The point of divergence is indicated for the methods, exept for SNN which completed the full ten laps.

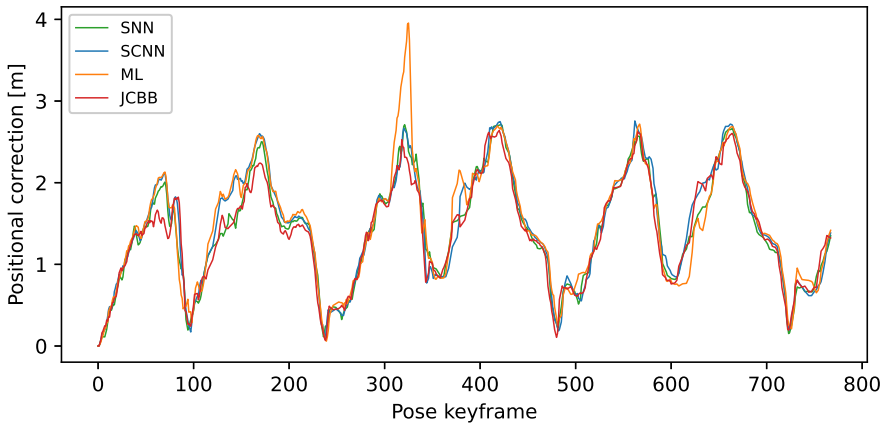


Figure 8.3: Truncated version of the full trajectory. Included to se the small spread between the methods.

As discussed in section 7.4.2, there are multiple ways to recover the marginal covariances from the iSAM2 backend. The choice of whether to include the off-diagonal elements of the covariance matrix in eq. (5.4) also has to be made. In figure 8.4, the ML data association is tested with three different means of covariance recovery. Only in "ML joint" are the off-diagonal cross-covariances between the pose and landmark recovered (again, see eq. (5.4)). The regular "ML" in the figure, uses the method described in 7.4.4, and "ML marginal" generates the same Marginals object as done on line 8 in Listing 7.2, but instead calls the `marginalCovariance(variable)` function on this object. As evident here, using the extra information from the joint covariance matrix does not yield better results, likely accredited to the increase in computation time (see figure 8.14. In summary, the best method for retrieving marginal covariances was found to be the one manipulating the iSAM2 object directly.

The last comparison was made in the FSG18 simulation. Here, the ground truth of cone locations and poses is known. The noise added to the pose is AWGN with simulated bias. Specifically, samples from the Gaussian distribution were modified and

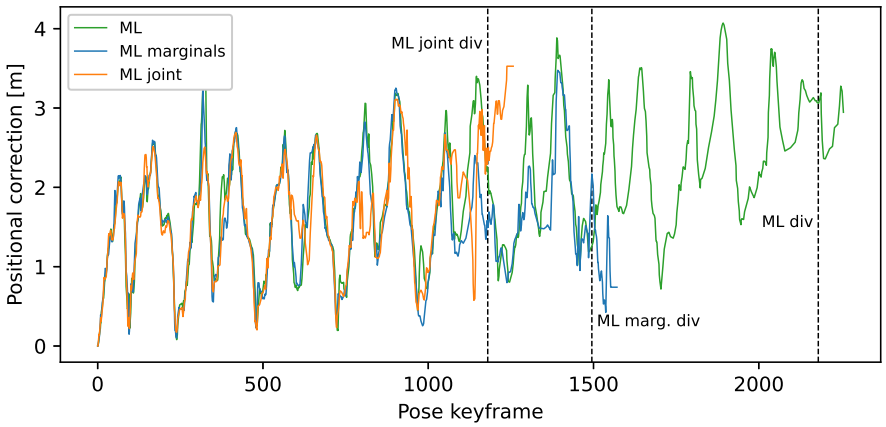


Figure 8.4: Positional correction applied when employing three different ML data association schemes. Point of divergence is indicated.

added separately to the x, y and θ odometry estimates entering slam as follows

$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{\theta} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + = \begin{pmatrix} w_x \\ w_y \\ w_\theta \end{pmatrix} \tag{8.1}$$
$$w_x, w_y, w_\theta = \frac{w + |w|}{2}, w \sim \mathcal{N}(0, 2.4 * 10^{-4}).$$

The odometry estimates enter at 200 Hz, which is why the standard deviation for the noise applied to each estimate is set seemingly low. Zero-mean Gaussian noise on the individual bearing-range measurements was also added as

$$\tilde{\vec{z}}_j = \vec{z}_j + \vec{v}_j, \quad \vec{v}_j \sim \mathcal{N}(0, diag(0.05^2, 0.1^2), \tag{8.2}$$

to simulate the uncertainties of the detection systems. Measurement clutter was simulated with five uniformly distributed false positives within the 30 m, 180° field of

view of the car.

To evaluate performance, the root mean square error (RMSE) of the cone placements after 50 runs of autocross (unknown map, one lap) was recorded for each method. Additionally, the mean and standard deviation in the number of cones in the map after a completed lap was estimated. Lastly, the failure rate for each method is estimated. "Failures" were classified as instances where the car did not end up within 3 meters of its starting position. Table 8.2 highlights the performance of each method. Clearly, the SNN is the best performer, with a zero failure rate. A possible explanation for this is that SNN is more detached from the probabilistic quantities than the other methods. Noise parameters are tuned based on the real data from Moholt, and so the probabilistic data association methods are penalized because they are not optimized for the simulation scenario.

The RMSE of poses is not included here because there were uneven numbers of ground truth poses and slam output poses. Looking back, timestamp matching could have been performed to establish two sets of even size, but was not considered further at the time because simulation data results were less prioritized than the Moholt data.

	SNN	SCNN	ML	JCBB
Cone pos. RMSE [m]	**0.07**	0.17	0.44	0.30
Mean num. cones	178	180	186	**179**
Std dev num. cones	**2**	7	16	9
Failure rate	**0%**	2%	10%	8%

Table 8.2: General performance parameters for the different DA-methods estimated on the FSG18 dataset over 50 one-lap runs. **Bold type** indicates the best performer for the particular parameter.

As a final visualization of the SLAM performance, figures 8.5 - 8.7 illustrate the effect SLAM has in the pipeline. The first figure plots all the incoming data from ten laps at Moholt, where the "smearing" of measurements is quite significant due to vehicle drift. Furthermore, in figure 8.6, all data is again plotted, but now detections are expressed in the frame of the SLAM-corrected vehicle trajectory. There is still a

certain smearing effect, but cones can, in general, be easily distinguished in the point cloud. Also, the vehicle trajectory is estimated within the track boundaries throughout the run. The final figure indicates all the data the SLAM module outputs during the ten laps. Even if some false positives are plotted outside the track, they are a lot less frequent when comparing to the input data. In this plot, the smearing is almost eliminated as the output positions of landmarks is almost constant during the run.

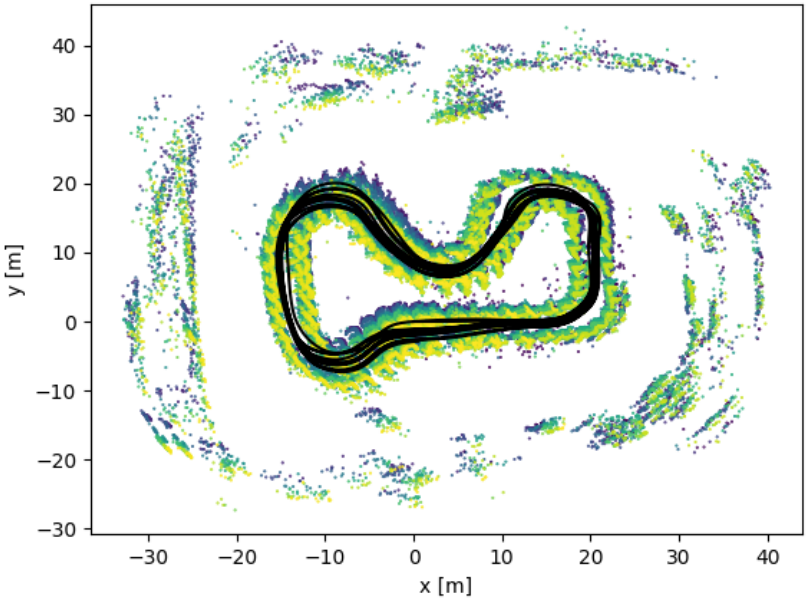


Figure 8.5: All detection data entering SLAM for ten laps at Moholt as seen from the *uncorrected* pose trajectory

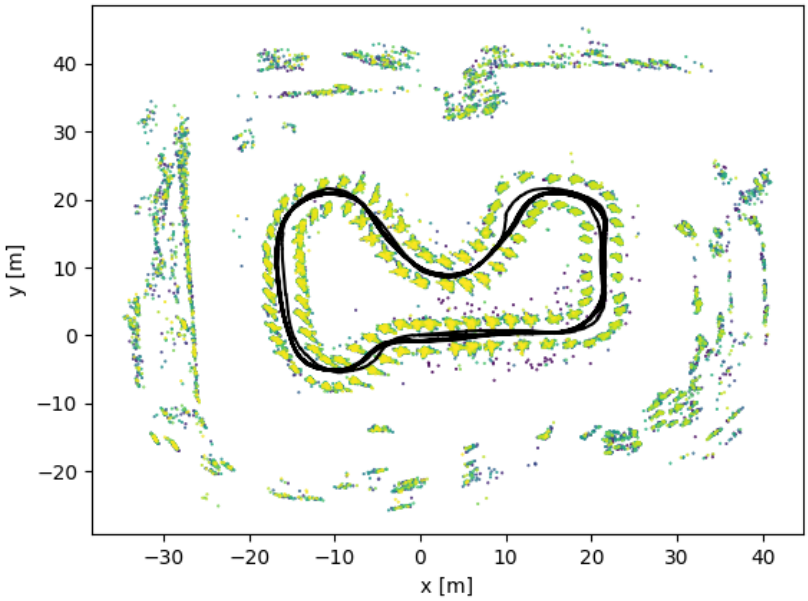


Figure 8.6: All detection data entering SLAM for ten laps at Moholt as seen from the *corrected* pose trajectory

8.3 Execution time

In any real-time system where reactive actions are taken based on the system input, keeping execution time down is always important. This allows incoming data to be processed as soon as they become available, and effectively reduce the reaction time of the system.

Substantial time was spent on optimizing each of the implemented data association schemes to operate as fast as possible. Most of the effort went into the JCBB algorithm. The first version of this implementation suffered large spikes in the iteration times. From figure 8.9, we can see that the main culprit is the repeated calls for the marginal

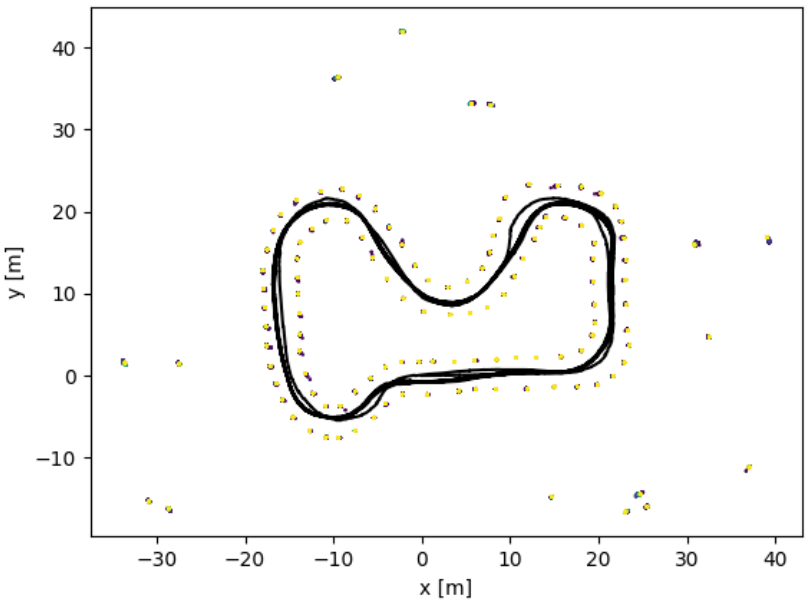


Figure 8.7: All data output from SLAM during ten laps. Cones are output at a rate of $\approx 20\,\text{Hz}$.

covariances in the IC-check. In this first version, these marginal covariances were recovered using the Marginals class in GTSAM, and performing key-by-key individual lookups. Contrary to the ML and SCNN schemes, which were found to perform better when off-diagonal elements of the covariance matrix involved in the IC-check were disregarded, these quantities and the correlation between landmarks is essential in JCBB. Because the joint covariance matrix is generated for a subset of variables anyways, the new version was altered to generate the joint covariance matrix for *all* the landmarks and the pose keyframe from which the current set of measurements were made, at each iteration. As shown in figure 8.11, iteration times became much more stable, but still increase with the number of pose keyframes in the variable set.

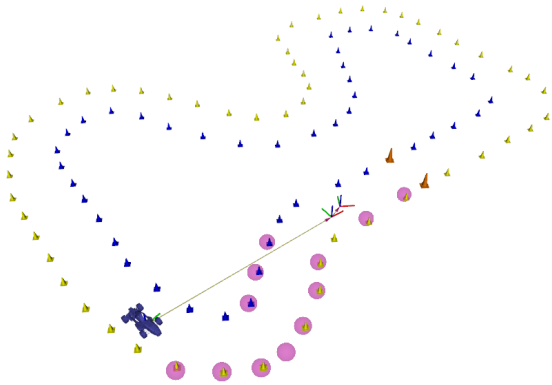


Figure 8.8: Visualization of the testing environment using ROS built-in visualization tool Rviz. This is the generated map of the Moholt test track with cone colors added on later.

Figure 8.10 shows the time taken for the components in the second version. As the marginal covariances on all variables are embedded on the diagonal in the joint covariance matrix, the marginal covariance recovery in the IC-checks are reduced to accessing a block in a stored matrix. The main reason for the feasibility of this solution is the limited number of landmarks. Because no new landmarks are added to the map after one lap is completed, the state vector only grows in the number of poses.

A similar comparison was done for the ML-data association scheme, already discussed in section 8.2. The difference in iteration times for the three methods for covariance retrieval is illustrated in figure 8.14.

Comparisons between the optimal versions of each data association scheme are found in figures 8.12 and 8.13. As expected, the SNN approach is extremely lightweight, and operates in a different timescale than the methods requiring uncertainty estimates. All these methods suffer from iteration times that grow in the number of poses until they diverge. A possible solution to this is to marginalize out older poses, similarly

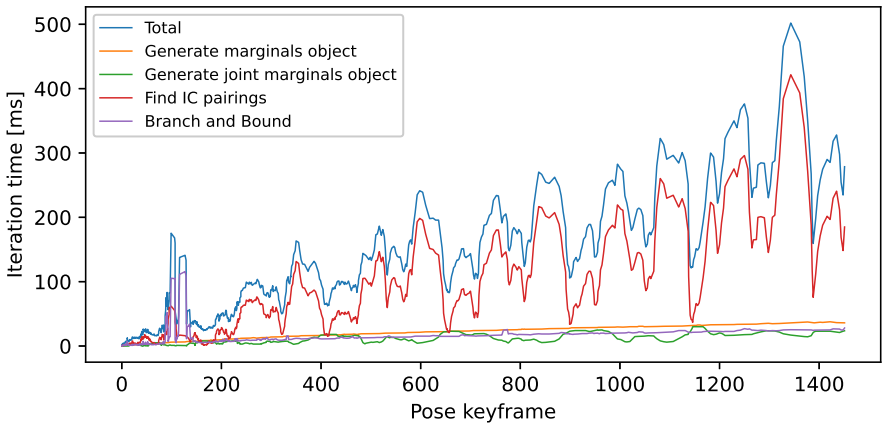


Figure 8.9: Detailed presentation of the time taken for each of the components of the *first* version of implemented JCBB. The object generation is the same as in Listing 7.2

to a *fixed-lag smoother*. This way, the state vector is kept of constant size. Due to constraints on time, this concept was not explored further.

8.4 Map preloading

Both the Moholt and FSG datasets were used in this test. To test how well each of the data association schemes performs in the start of a run where the map is stored beforehand, a positional offset is applied to all the cones to simulate the initial pose uncertainty. From the vehicle's perspective, the initial pose is always $\vec{x}_0 = (0, 0, 0)^\mathsf{T}$, but cannot be guaranteed to equal to the earth fixed initial pose of the vehicle from when the map was created. Also, for the skidpad and acceleration events (not tested here), the map is defined from the rules with the origin at the *expected* vehicle starting point with the x-axis parallel to the longitudinal axis of the vehicle and not the actual starting point. In summary:

1. Cones saved from a previous run are given in the inertial map frame $\mathcal{I}$.

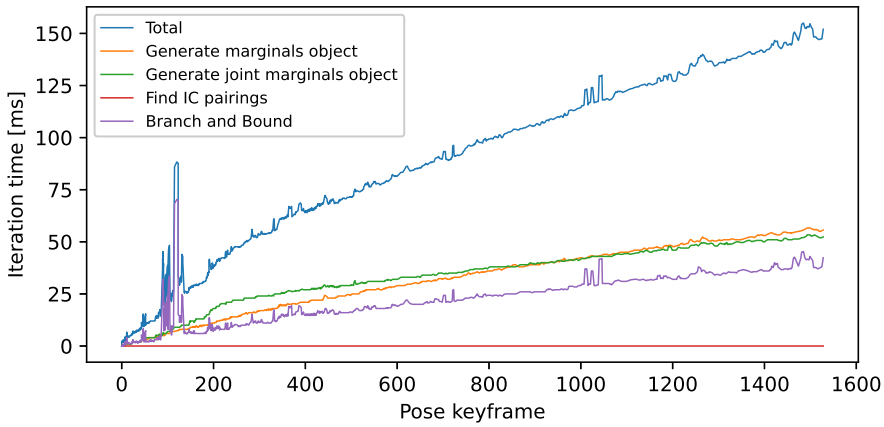


Figure 8.10: Detailed presentation of the time taken for each of the components of the *second* version of JCBB. The object generation is the same as in Listing 7.2

2. In practice, the vehicle cannot be guaranteed to start at the exact same spot with the exact same heading each time, meaning that the origin of subsequent runs will be defined as the map frame with an offset: $\mathcal{I}' = \mathcal{I} + \vec{\delta}$ where $\vec{\delta} = (\delta x, \delta y, \delta\theta)^{\mathsf{T}}$.

3. All cones in the map $L = \{\vec{l}_i\}_{i=1}^{N}$ are then subject to the transformation $\boldsymbol{T}(\delta)$:

$$L' = \{\boldsymbol{T}(\vec{\delta})\vec{l}_i\}_{i=1}^{N} \tag{8.3}$$

4. Detections will still enter in the body frame $\mathcal{B}$, which at time step 0 is equal to $\mathcal{I}'$, but because of the introduced offset they will not align perfectly as they would if $\mathcal{I}' = \mathcal{I}$.

If the data association schemes correctly pair detections to cones in the map, the added measurement factors are expected to be sufficient for the backend to estimate the error $\vec{\delta}$ imposed on the initial vehicle pose as defined in the "true" map frame $\mathcal{I}$. The result of this allows the vehicle to continue the lap as usual in the *localization* state. An important practical distinction to make here is that when performing full

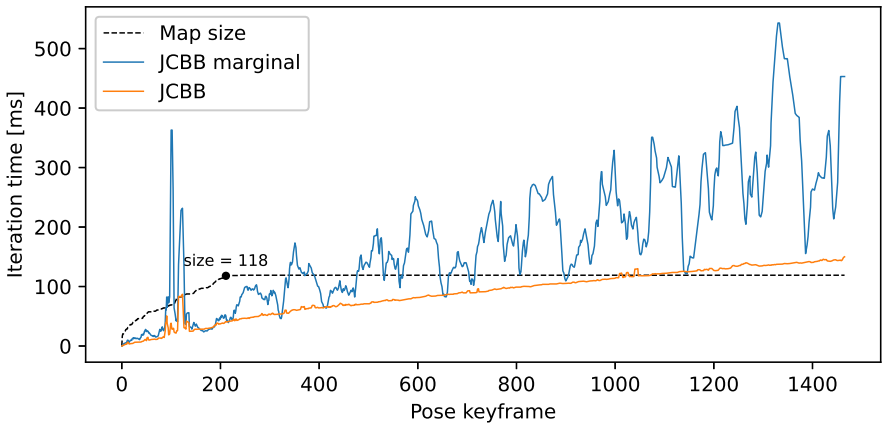


Figure 8.11: Comparison of iteration time between the two versions of implemented JCBB

SLAM (mapping and localization), the initial pose uncertainty is an irrelevant metric because we define the origin as equal to the initial pose. For localization only, the origin is already set from when the map was built, and so the initial pose uncertainty is a parameter that is nonzero and should be set based on a realistic setting. For this test, the initial uncertainty is modeled using a GTSAM *PriorFactor* with mean $\vec{\mu} = \vec{x}_0 = (0, 0, 0)^\mathsf{T}$ and *diagonal* covariance matrix with noise parameters (std devs) $\sigma_x = \sigma_y = 0.5\,\text{m}, \sigma_\theta = 15^\circ \approx 0.26\,\text{rad}$. The cones are also modeled with PriorFactors, where the mean of each cone is as given in (8.3) and with uncorrelated noise in the x- and y-directions with $\sigma_x = \sigma_y = 0.3m$. It can be argued that the ideal implementation of the noise estimates would involve the relevant indices from the covariance matrix recorded together with the map. But because the correctness of the saved map cannot be guaranteed and also to not *overconstrain* the problem, modeling with a moderately peaked Gaussian was assumed to be acceptable. This implementation is also applicable to the skidpad and acceleration events, where no noise estimates are available on the first run.

In this test, the evaluation is based on a binary *success* metric. If the SLAM system

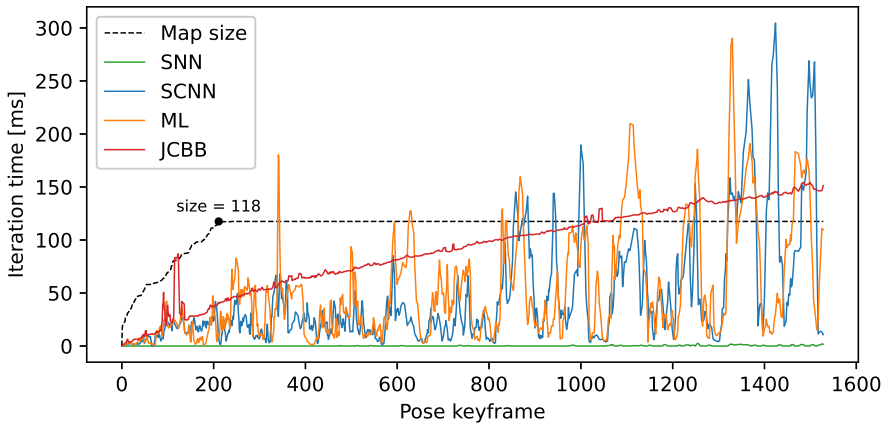


Figure 8.12: Iteration time tracked for all data association schemes on the Moholt dataset.

recovers after the initial pose offset (cone offset in practice) and continues the lap as usual, the test is deemed a success. Figure 8.15 shows the success rates of all four data association schemes when the initial pose is subject to the normally distributed offset $\vec{\delta} \sim \mathcal{N}(\vec{0}, diag(\sigma_x^2, \sigma_y^2, \sigma_\theta^2)$ with the noise parameters equal to the ones modeled in the PriorFactor. Here, JCBB is the best performer, while the non-probabilistic SNN scheme is the worst.

In addition to the more realistic test of applying a zero-mean distributed offset, additional simulations were performed to establish the upper bound of offsets each of the data association methods were still able to succeed with. In this case, the applied offset is known and only applied to the initial yaw angle (not translated). This is mainly because large deviations in the starting position are unrealistic within the competition rules. Tables 8.3 and 8.4 display the results, which are in line with the results from figure 8.15 as JCBB again is the better performer, and SNN is the weakest. To illustrate the differences between the two algorithms, figure 8.16 shows an instance where JCBB makes the correct associations, where SNN would not have been able to. Adjusting the IC and JC probability thresholds (see equations (5.7) and (5.12)) from the standard 0.90

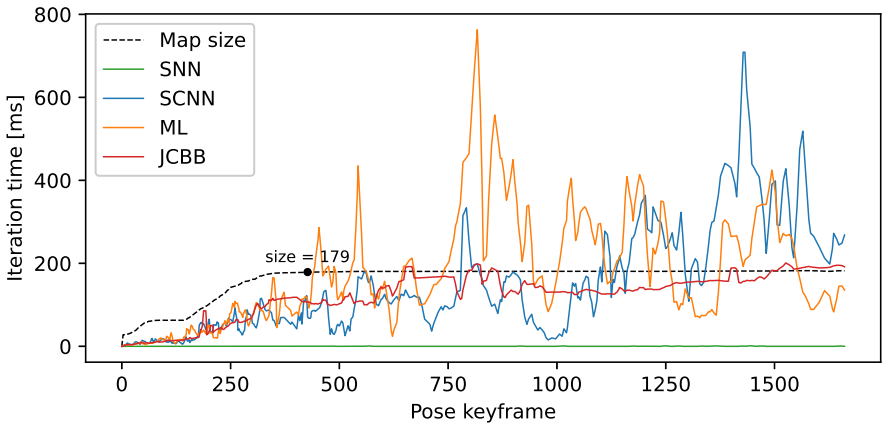


Figure 8.13: Iteration time tracked for all data association schemes on the FSG dataset

to 0.95 and 0.75 gave minimal changes in results, which suggests that these parameters can benefit from further tuning.

$chi2inv(\cdot)$ (excl. SNN)	Angle range [°]			
	SNN	SCNN	ML	JCBB
0.9 (default)	[-12.0, 13.8]	[-18.3, 20.6]	[-24.6, 25.4]	[**-29.8**, **36.1**]
0.75	-	[-18.3, 16.0]	[-24.6, 25.2]	[**-28.6**, **29.2**]
0.95	-	[-18.3, 20.6]	[-24.6, 25.4]	[**-29.8**, **36.1**]

Table 8.3: The ranges of initial unknown yaw offset that the DA methods managed with success. **Bold type** indicates best performer. These results are found for the Moholt dataset.

8.5 Lidar noise experiment

The point of this experiment was not to estimate the intrinsic noise parameters on the Hesai Pandar40 lidar, but rather the noise model of the feature extraction performed on

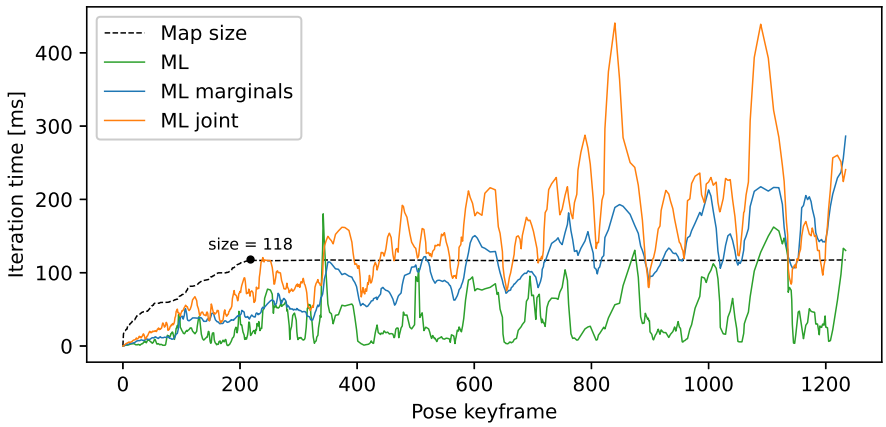


Figure 8.14: Comparison of iteration time of ML-data association using different means of covariance retrieval

the lidar pointclouds. This was done to give a general idea of how large the measurement noise is on a stationary vehicle, and if increased ranges correspond to more uncertainty when stationary. This could then possibly get integrated into the implementation of SLAM, specifically as the noise values supplied with the `BearingRangeFactors()` in the backend.

The test was performed with a stationary trolley with the lidar and a processing unit mounted. Cones were set up in multiple maps, but only two tests are included here. The first test is the straight test where cones are somewhat evenly spaced on two straight lines ahead. The results from this test are displayed in figures 8.17 and 8.18. The first figure is included to show the setup of the lidar and cones, and also to give an exaggerated indication of the distributions of detections. The covariance ellipses are plotted with 10 standard deviations. From figure 8.20, we can say that range has little effect on the uncertainty of measurements output from the lidar feature extraction. Also, because the numbers are very low, the performance is good. It is important to note that these tests were performed in ideal conditions: Stationary, nice weather and with little potential clutter in the track. Because of this, setting the measurement noise

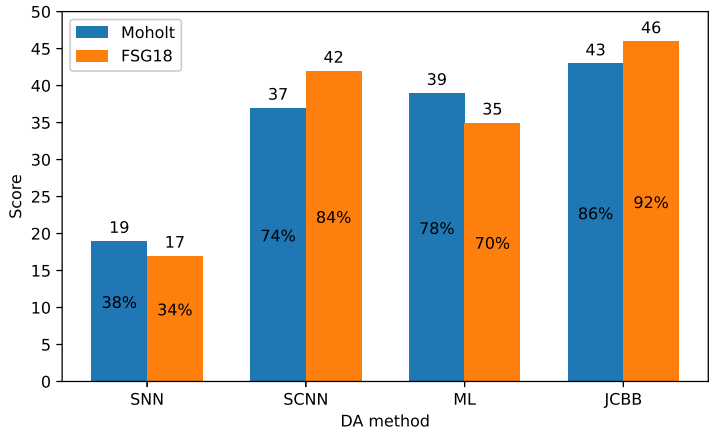


Figure 8.15: The number of successful runs of each system after being exposed to an initial unknown offset in pose. The test was performed 50 times for each data association method on both the Moholt and FSG18 datasets, and the success rates are also included.

to these estimated values is not considered to be a feasible solution unless the vehicle is stationary, if even then.

Much of the same can be said for the hairpin turn situation in figures 8.19 and 8.20. The correlation between uncertainty and range is not there, and the lidar detection is again very accurate.

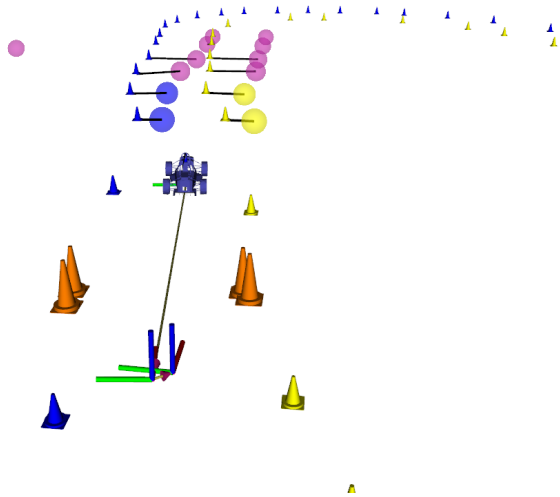


Figure 8.16: An instance where JCBB correctly associates the measurements (yellow, blue and purplue markers) with the cones after preloading the map. Visually, we can verify that SNN would possibly only have associated the closest (and maybe second) pair of cones correctly.

$chi2inv(\cdot)$ (excl. SNN)	Angle range [°]			
	SNN	SCNN	ML	JCBB
0.9 (default)	[-8.6, 5.7]	[-38.4, 22.9]	[-38.4, 24.1]	[**-40.1**, **26.9**]
0.75	-	[**-38.4**, 22.9]	[-37.8, 24.6]	[-37.2, **25.2**]
0.95	-	[-39.5, 24.6]	[-38.4, 25.8]	[**-40.7**, **26.9**]

Table 8.4: The ranges of initial unknown yaw offset that the DA methods managed with success. **Bold type** indicates best performer. These results are found for the FSG18 dataset.

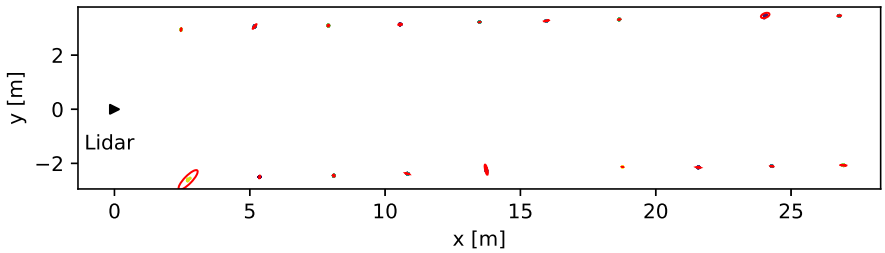


Figure 8.17: Covariance ellipses around detections on straight with $\sigma = 10$ for visual effect.

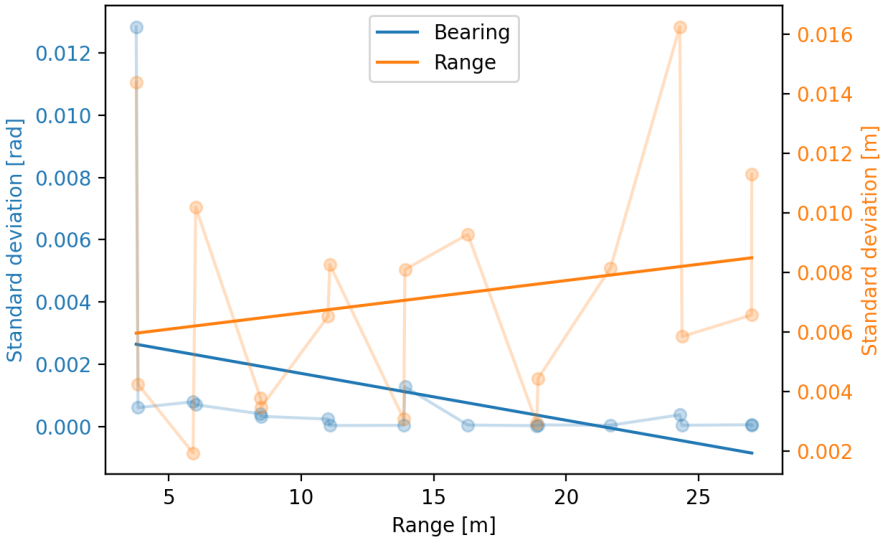


Figure 8.18: Standard deviation of bearing and range at different distances. Based on 128 sets of measurements where 13/18 cones were detected in every set.

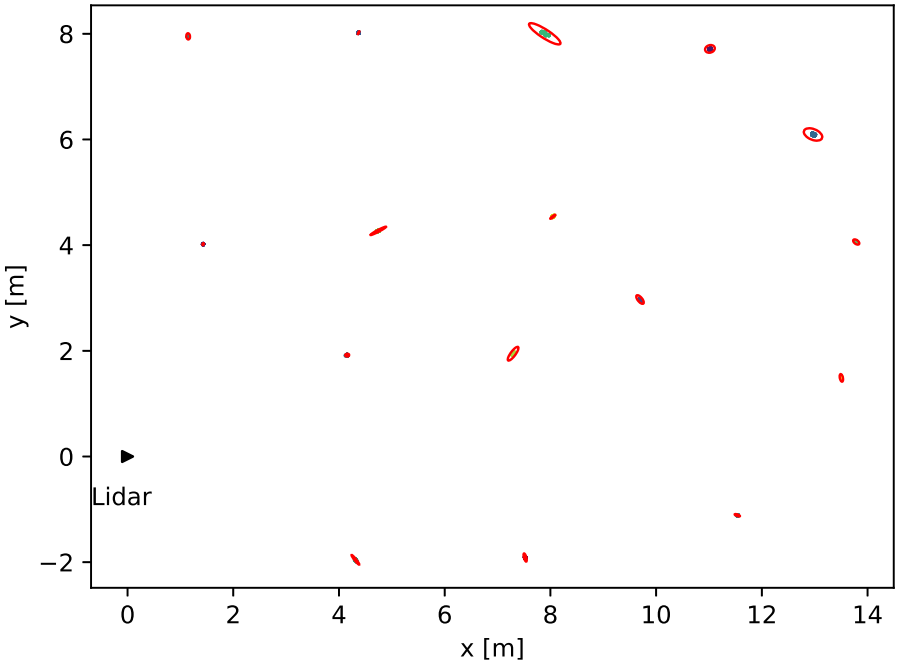


Figure 8.19: Covariance ellipses around detections in hairpin turn with $\sigma = 10$ for visual effect.

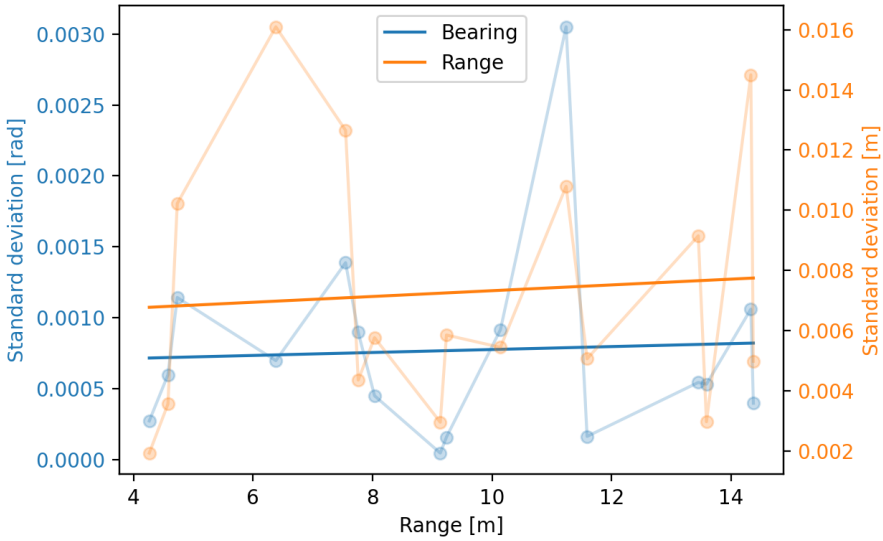


Figure 8.20: Standard deviation of bearing and range at different distances. Based on 160 sets of measurements where 14/16 cones were detected in every set.

Chapter 9

Conclusion and future work

In this thesis, the iSAM2 SLAM backend was paired with four methods for data association. The methods were compared on a dataset obtained in a realistic setting, and in simulations. As real-time performance is paramount in autonomous racing, evaluating the execution times of the methods was central. Results from both the real and artificial setting showed that an increase in execution time beyond a certain threshold is critical to performance because important data is not being processed. The slower methods have potential, however. In map preloading, the slowest method: JCBB outperforms the others, in what is an important discipline in FS competitions. Although a loop closure "stress test" was not performed, prior research [42] [29] would indicate that JCBB is the more robust method, once drift is significant enough. Probably due to the small area covered by the Moholt test track, the long term drift is not allowed to accumulate, as cones close to the starting position were within the detecting range of the lidar at multiple locations in the track. This resulted in many smaller loop closures, as opposed to a larger, more difficult one. Ideally the testing data should have originated from a larger track, so that loop closure could have been tested more.

In testing, it became clear that if only given the choice of using one of the methods, the standard SNN approach is the only possible option. This is because it is the only approach that completes the full ten laps of a trackdrive run. That being said, due to

the way the frontend is set up, there is no hindrance in changing the data association method mid-run. This way, more comprehensive schemes such as JCBB or ML-DA can be used in the initial phase of a run to allow more robust operation when preloading maps. Subject to the user's settings, the lightweight SNN scheme can take over in time before the computational load of the other methods become too large. A similar setup could be applied to the loop closing scenario. So that if the vehicle can predict a imminent loop closure (using heading, position and planned path for instance), more advanced techniques can take over in this critical phase.

Future work

- Because the computations of SCNN, ML and JCBB become slower in parallel with the number of pose keyframes in the variable set of iSAM2, exploring the possiblity of implementing the SLAM backend as an incremental *fixed-lag smoother*[1]
- As the implemented SLAM system is already multithreaded, adding a third thread responsible for covariance retrieval only should not be a very difficult addition. This will free up CPU time on the frontend tread, which would allow it to process data faster, and close the gap between the probabilistic approaches and SNN.
- A completely new approach could be taken as well. Instead of performing feature-based landmark SLAM, as is the case now, rethinking the interface between the detection modules and SLAM could allow for better solutions. The ultimate goal of the car is to find a driveable route, and so focusing too heavily on distinguishing cones might not be the optimal solution for tackling the problem. The author suggests future Revolve members to explore the usage of lidar-based occupancy grid SLAM.
- As the number of cones (landmarks) is bounded in FSD competitions, going for the standard EKF approach could also be considered. The shortcomings of

[1] iSAM2 fixed lag smoother `https://gtsam.org/doxygen/a04411.html`.

EKF-SLAM are most apparent in large-scale applications, where the number of landmarks can grow indefinitely. This is not the case for FSD. As the move towards lower powered computational units is bound to happen due to the EV and DV merge in 2022, applying a simple EKF-SLAM system could be sufficient.

References

[1] Agarwal, P. and Olson, E. [2012]. Variable reordering strategies for slam, *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp. 3844–3850.

[2] Andresen, L., Brandemuehl, A., Hönger, A., Kuan, B., Vödisch, N., Blum, H., Reijgwart, V., Bernreiter, L., Schaupp, L., Chung, J. J. et al. [2020]. Fast and accurate mapping for autonomous racing, *arXiv preprint arXiv:2003.05266* .

[3] Armstrong, M. A. [1997]. *Groups and Symmetry*, Springer Science & Business Media.

[4] Bailey, T. and Durrant-Whyte, H. [2006]. Simultaneous localization and mapping (slam): Part ii, *IEEE robotics & automation magazine* **13**(3): 108–117.

[5] Bar-Shalom, Y., Fortmann, T. E. and Cable, P. G. [1990]. Tracking and data association.

[6] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. and Leonard, J. J. [2016]. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age, *IEEE Transactions on robotics* **32**(6): 1309–1332.

[7] Castañón, D. A. [1992]. New assignment algorithms for data association, *Signal and Data Processing of Small Targets 1992*, Vol. 1698, International Society for Optics and Photonics, pp. 313–323.

[8] Cummins, M. and Newman, P. [2011]. Appearance-only SLAM at large scale with FAB-MAP 2.0, *The International Journal of Robotics Research* **30**(9): 1100–1123.

[9] Daellert, F. et al. [2020]. *GTSAM C++ Library*, Georgia Tech Borg lab. `www.gtsam.org`.

[10] Davis, T. A., Gilbert, J. R., Larimore, S. I. and Ng, E. G. [2004]. A column approximate minimum degree ordering algorithm, *ACM Transactions on Mathematical Software (TOMS)* **30**(3): 353–376.

[11] Dellaert, F. and Kaess, M. [2006]. Square root SAM: Simultaneous localization and mapping via square root information smoothing, *The International Journal of Robotics Research* **25**(12): 1181–1203.

[12] Dellaert, F., Kaess, M. et al. [2017]. Factor graphs for robot perception, *Foundations and Trends® in Robotics* **6**(1-2): 1–139.

[13] Dissanayake, M. G., Newman, P., Clark, S., Durrant-Whyte, H. F. and Csorba, M. [2001]. A solution to the simultaneous localization and map building (SLAM) problem, *IEEE Transactions on robotics and automation* **17**(3): 229–241.

[14] Dissanayake, M. G., Newman, P., Durrant-Whyte, H. F., Clark, S. and Csorba, M. [2000]. An experimental and theoretical investigation into simultaneous localisation and map building, *Experimental robotics VI*, Springer, pp. 265–274.

[15] Durrant-Whyte, H. and Bailey, T. [2006]. Simultaneous localization and mapping: part i, *IEEE robotics & automation magazine* **13**(2): 99–110.

[16] Engebretsen, M. [2018]. iSAM2 SLAM in driverless project at Revolve NTNU. Bachelor thesis at NTNU.

[17] Engel, J., Schöps, T. and Cremers, D. [2014]. LSD-SLAM: Large-scale direct monocular SLAM, *European conference on computer vision*, Springer, pp. 834–849.

[18] Forbes, C., Evans, M., Hastings, N. and Peacock, B. [2011]. *Statistical distributions*, John Wiley & Sons.

[19] *Formula Student Rules 2020* [2020]. `https://www.formulastudent.de/fsg/rules/`. [Online; accessed 20-May-2020].

[20] Frese, U. [2006]. Treemap: An O (log n) algorithm for indoor simultaneous localization and mapping, *Autonomous Robots* **21**(2): 103–122.

[21] Geiger, D., Verma, T. and Pearl, J. [1990]. Identifying independence in bayesian networks, *Networks : an international journal* **20**(5): 507–534.

[22] George, A., Gilbert, J. R. and Liu, J. W. [2012]. *Graph theory and sparse matrix computation*, Vol. 56, Springer Science & Business Media.

[23] Golub, G. H. and Van Loan, C. F. [2012]. *Matrix computations*, Vol. 3, JHU press.

[24] Grisetti, G., Kummerle, R., Stachniss, C. and Burgard, W. [2010]. A tutorial on graph-based SLAM, *IEEE Intelligent Transportation Systems Magazine* **2**(4): 31–43.

[25] Guennebaud, G., Jacob, B. et al. [2018]. Eigen linear algebra C++ library. version 3.3.7. `www.eigen.tuxfamily.org`.

[26] Gustavsen, L. [2018]. *Development of a Mapping System for an Autonomous Formula Student Race Car*, Master's thesis, Norwegian University of Science and Technology (NTNU).

[27] *Hesai Pandar P40* [2020]. `https://www.hesaitech.com/en/Pandar40`. [Online; accessed 21-May-2020].

[28] Kabzan, J., Valls, M. d. l. I., Reijgwart, V., Hendrikx, H. F. C., Ehmke, C., Prajapat, M., Bühler, A., Gosala, N., Gupta, M., Sivanesan, R. et al. [2019]. AMZ driverless: The full autonomous racing system, *arXiv preprint arXiv:1905.05150* .

[29] Kaess, M. and Dellaert, F. [2009]. Covariance recovery from a square root information matrix for data association, *Robotics and autonomous systems* **57**(12): 1198–1210.

[30] Kaess, M., Ila, V., Roberts, R. and Dellaert, F. [2010]. The bayes tree: An algorithmic foundation for probabilistic robot mapping, *Algorithmic Foundations of Robotics IX*, Springer, pp. 157–173.

[31] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. and Dellaert, F. [2011]. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering, *2011 IEEE International Conference on Robotics and Automation*, IEEE, pp. 3281–3288.

[32] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J. and Dellaert, F. [2012]. iSAM2: Incremental smoothing and mapping using the bayes tree, *The International Journal of Robotics Research* **31**(2): 216–235.

[33] Kaess, M., Ranganathan, A. and Dellaert, F. [2008]. isam: Incremental smoothing and mapping, *IEEE Transactions on Robotics* **24**(6): 1365–1378.

[34] Kepner, J. and Gilbert, J. [2011]. *Graph algorithms in the language of linear algebra*, SIAM.

[35] Kuhn, H. W. [1955]. The hungarian method for the assignment problem, *Naval research logistics quarterly* **2**(1-2): 83–97.

[36] Labbe, M. and Michaud, F. [2014]. Online global loop closure detection for large-scale multi-session graph-based slam, *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp. 2661–2666.

[37] Li, Y. and Olson, E. B. [2012]. Ipjc: The incremental posterior joint compatibility test for fast feature cloud matching, *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, pp. 3467–3474.

[38] Millar, R. B. [2011]. *Maximum likelihood estimation and inference: with examples in R, SAS and ADMB*, Vol. 111, John Wiley & Sons.

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

[40] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B. et al. [2003]. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges, *IJCAI*, pp. 1151–1156.

[41] Mur-Artal, R., Montiel, J. M. M. and Tardos, J. D. [2015]. ORB-SLAM: a versatile and accurate monocular SLAM system, *IEEE transactions on robotics* **31**(5): 1147–1163.

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

[43] Omvik, P. K. [2019]. Data association in simultaneouslocalization and mapping for anautonomous race car. Specialization project at NTNU.

[44] Open Robotics [2016]. ROS kinetic kame. `www.ros.org`.

[45] *Ouster OS-1* [2020]. `https://ouster.com/products/os1-lidar-sensor/`. [Online; accessed 21-May-2020].

[46] Palerud, B. [2020]. *Lidar-based object detection for an autonomous racecar*, Master's thesis, Norwegian University of Science and Technology (NTNU).

[47] Ranganathan, A., Kaess, M. and Dellaert, F. [2007]. Loopy SAM, *Proceedings of the 20th international joint conference on Artifical intelligence*, pp. 2191–2196.

[48] Rodriguez-Losada, D. and Minguez, J. [2007]. Improved data association for icp-based scan matching in noisy and dynamic environments, *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, pp. 3161–3166.

[49] Rusu, R. B. and Cousins, S. [2011]. 3D is here: Point Cloud Library (PCL), *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China.

[50] Shen, X., Frazzoli, E., Rus, D. and Ang, M. H. [2016]. Fast joint compatibility branch and bound for feature cloud matching, *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 1757–1764.

[51] Skibelid, A. [2019]. Odometry, mapping and localisation of an autonomous race car for Revolve NTNU. Specialization project at NTNU.

[52] Smith, R., Self, M. and Cheeseman, P. [1990]. Estimating uncertain spatial relationships in robotics, *Autonomous robot vehicles*, Springer, pp. 167–193.

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

[54] Szeliski, R. [2010]. *Computer Vision: Algorithms and Applications*, Texts in Computer Science, Springer London.
URL: *https://books.google.no/books?id=bXzAlkODwa8C*

[55] Thrun, S. [2002]. Probabilistic robotics, *Communications of the ACM* **45**(3): 52–57.

[56] Tong, Y. L. [2012]. *The multivariate normal distribution*, Springer Science & Business Media.

[57] *Vectornav VN-300* [2020]. `https://www.vectornav.com/products/vn-300`. [Online; accessed 21-May-2020].

[58] *Victoria Park Dataset* [2006]. `http://www-personal.acfr.usyd.edu.au/nebot/victoria_park.htm`.

[59] Wang, J. and Englot, B. [2018]. Robust exploration with multiple hypothesis data association, *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 3537–3544.

Per Kvinnesland Omvik

Data Association and Simultaneous Localization and Mapping for an Autonomous Racecar

NTNU
Norwegian University of
Science and Technology

revolve