Master's thesis

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

Nicholas Dalhaug

Lidar-based Localization for Autonomous Ferry

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke and Gustaf Hendeby
June 2019

NTNU
Norwegian University of
Science and Technology

Nicholas Dalhaug

Lidar-based Localization for Autonomous Ferry

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke and Gustaf Hendeby
June 2019

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

NTNU
Norwegian University of
Science and Technology

Preface

This document serves as the master's thesis for the master's project *TTK4900* in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). By contributing the last 30SP, it finalizes the Master of Technology degree. The master's project was preceded by a specialization project [13] and is a continuation within the same field of mapping and localization for autonomous docking. The specialization project was a literature study and analysis of the necessity of Simultaneous Localization And Mapping (SLAM) in mapping and localization for autonomous docking.

This master's project is instantiated under the Autoferry Project at the Centre for Autonomous Marine Operations and Systems (AMOS) at NTNU. It has the goal to analyze methods for localizing the Unmanned Surface Vehicle (USV) Milliampere using a lidar. The equipment at disposal includes data gathered in earlier experiments.

I would like to thank my supervisor Edmund Førland Brekke and co-supervisor Gustaf Hendeby for giving me the possibility to work on this project and for taking the time to counsel and supervise my activities throughout it. It has been an interesting journey and I hope it is interesting for you as well. I would also like to thank my colleagues for discussions on the relevant topics.

Abstract

This thesis wants to establish properties for localization of an Unmanned Surface Vehicle (USV), explain advantages and disadvantages of some methods for localization and suggest where to focus next. The use of a lidar to do this is motivated by the fragility of Global Navigation Satellite Systems (GNSSs). The analyzes in this thesis are important in order to lay a basis for further development, development that is based on an understanding of the underlying problem.

The particle filter using a physically based measurement model and a simple kinematics motion model can be used to localize the USV in an occupancy grid. Using only the motion model as a prior for movement gives the fastest method, using Unscented Kalman Filter (UKF) gives no benefit and using Iterative Closest Point (ICP) reduces the state estimation error. Using the physically based measurement model it is suggested to tune parameters such as measurement uncertainty to get better estimation consistency.

An analysis of the workings of the physically based measurement model is at least as important as the particle filter results. While the physical measurement model does not handle discontinuities in the map well, the ICP method is better at taking the structure of the environment into account. The physically based measurement model will, depending on the noise and tuning, have difficulties with different kinds of noise. The ICP method has different problems, but the use of it as a measurement model might solve several different issues.

The basis established by this thesis will supplement the development of robust localization methods that enable an autonomous ferry to conduct duties such as transit and safe and reliable docking . These duties should be carried out only using onboard sensors.

Sammendrag

This section gives a summary in norwegian.

Denne masteroppgaven ser på egenskaper ved lokalisering av en Unmanned Surface Vehicle (USV), forklarer fordeler og ulemper ved noen metoder for lokalisering og foreslår hvor fokus bør ligge videre. Bruken av en lidar til å gjøre dette er motivert av skjørheten til Global Navigation Satellite System (GNSS). Analysene i denne oppgaven er viktige for å lage en basis for videre utvikling, utvikling som er basert på en forståelse av det underliggende problemet.

Patrikkelfilteret med en fysikk-basert målemodell og en enkel kinematisk bevegelsesmodell kan brukes for å lokalisere USVen i et occupancy grid. Bruk av en bevegelsesmodell som forslag for bevegelse gir den raskeste metoden, å bruke Unscented Kalman Filter (UKF) gir ingen fordeler og å bruke Iterative Closest Point (ICP) reduserer feilen i estimert tilstand. Ved bruk av en fysikk-basert målemodell anbefales det å tune parametre slik som måleusikkerheten for å få bedre konsistensegenskaper.

En analyse av hvordan en fysikk-basert sensor modell virker er minst like viktig som resultatene fra partikkelfilteret. Mens den fysikk-baserte målemodellen ikke håndterer diskontinuiteter i kartet godt er ICP-metoden bedre til å ta strukturen i miljøet med i betraktningen. Den fysikk-baserte målemodellen vil, avhengig av støy og tuning, ha problemer med forksjellige typer støy. ICP-metoden har andre problemer, men bruken av den som en målemodell kan løse flere forksjellige problemer.

Grunnlaget etablert av denne oppgaven supplementerer utviklingen av robuste lokaliseringsmetoder som muliggjør at en autonom ferge kan utføre plikter slik som gjennomreise og trygg og pålitelig tillegging til kai. Disse pliktene skal utføres kun ved bruk av sensorer om bord.

Contents

Acronyms

AIS Automatic Identification System. 3

AMOS the Centre for Autonomous Marine Operations and Systems. i

ANEES Average Normalized Estimation Error Squared. 59, 68, 69, 84, 85, 88, 97, 98, 114, 116, 118–121, 123

ASV Autonomous Surface Vehicle. 1, 3, 7, 10, 16

DME Distance Measuring Equipment. 9, 12, 20, 31, 32

DOF Degrees Of Freedom. 28, 61, 69, 71, 80, 90, 97, 114, 118, 121

DP Dynamic Positioning. 4, 14

EKF Extended Kalman Filter. 39, 40, 89

ENU East North Up. 24, 28

GNC Guidance, Navigation and Control. 7

GNSS Global Navigation Satellite System. iii, v, 3, 5, 7, 14, 16, 18

GPS Global Positioning System. 7, 9, 11, 139

GUI Graphical User Interface. 20

HMM Hidden Markov Model. 44

ICP Iterative Closest Point. iii, v, 5, 11, 12, 35, 39, 61–63, 100–103, 105, 106, 108, 110–117, 119–121, 123, 124

IMU Inertial Measurement Unit. 3, 4, 7, 10, 16, 18, 19, 31, 133

INS Inertial Navigation System. 3, 5, 14, 18, 31

IS Importance Sampling. 43

LG Linear Gaussian. 59

MMSE Minimum Mean Square Error Estimate. 54, 57, 84

NED North East Down. 24, 137, 141

NEES Normalized Estimation Error Squared. 58, 59, 114

NIS Normalized Importance Sampling. 43

NTNU the Norwegian University of Science and Technology. i, 2, 13

OpenCV Open Source Computer Vision Library. 19, 20, 141

PCL Point Cloud Library. 11, 19–21, 61, 139

PF Particle Filter. 5, 8, 39, 45, 89, 99, 112, 115

RMSE Root Mean Square Error. 59, 68, 69, 84, 85, 88, 97, 98, 102, 103, 105–111, 113, 114, 116, 119, 120, 123

ROS Robot Operating System. 8, 11, 19–21, 24, 32, 131, 133, 135, 139, 141, 145

RTK-GPS Real-Time Kinematic Global Positioning System. 3, 16, 18

SIR Sampling Importance Re-sampling. 50, 52, 53, 60, 68, 69, 90–96

SIS Sequential Importance Sampling. 45, 46, 48, 52, 66, 67

SLAM Simultaneous Localization And Mapping. i, 3, 7, 8, 11, 12, 24, 122, 131, 139

SNR Signal to Noise Ratio. 69

SVD Singular Value Decomposition. 61, 62

TOF Time Of Flight. 31

UKF Unscented Kalman Filter. iii, v, 5, 39–42, 60, 89–91, 94–97, 99, 118–120, 123

UPF Unscented Particle Filter. 39, 60, 89, 90, 92–100, 114, 117–120

USV Unmanned Surface Vehicle. i, iii, v, 1, 3, 4, 7–9, 16

UT Unscented Transform. 40, 41

Chapter 1

Introduction

This chapter introduces the field of Autonomous Surface Vehicles (ASV), explains the specific problem of localizing the ferry and specifies the structure of this thesis.

1.1 Background

The development of Autonomous Surface Vehicles (ASV) has come far. Unmanned Surface Vehicles (USVs) are vehicles that operate on the surface of water without an operator on-board, with varying degrees of autonomy [44, p. x]. ASVs are USVs where the vessel is autonomous. The historical review [38] mentions that an ASV was developed in 1993 which was used to collect simple bathymetry data. Years later, prototypes have been demonstrating capabilities of autonomous operation. Jokioinen explains that autonomous ships can now be at least as safe as manned ships [28]. This article also states that the main technological challenges regard cost and reliability. Those are therefore important research topics within the field of ASVs together with development of new features.

There are several reasons for making an autonomous ship. One such reason is that they might be more cost effective and more environmentally friendly than manned ships. For military use it could manage tasks that are dangerous for humans or it could increase the situational awareness on missions. In urban environments, autonomous ferries could replace manned ferries and bridges [3]. The fact that they are autonomous can also give a better integration with higher level systems. For example when it comes to minimizing the ferry usage or waiting time, the ferry can be integrated in an intelligent higher level public transportation system. Lastly, the reduced cost of ferries might enable ferries to work in environments where they have been too expensive to be implemented. This might be the case for several places along the coast or fjords where a manned ferry would not be economically efficient.

The Fløtmann in Trondheim, a man rowing people across a specific water passage, was a time-saver for people working or travelling in the Brattøra region [33]. The history goes back to the 1880s with the constructions around the railroad and when people needed to get across the water passage between Ravnkloa and Brattøra. The Fløtmann rowed people across the passage until 1965. The tradition was revived in 1997, but only for certain days during the summer months. This passage is therefore not easy to traverse outside of those days, and especially not if one wants to transport a bicycle over the passage.

1.2 Motivation

The Autoferry project is a project at the Norwegian University of Science and Technology (NTNU) [3]. Its goal is to develop methods and increase the relevant level of knowledge that will enable the production and use of autonomous passenger ferries. These ferries should be fully electrical and able to transport people in urban water passages on-demand, making them arrive when a button is pressed. A key part of the project is the ferry prototype Milliampere shown in Figure 1.1. The ferry is designed to roam the passage between Ravnkloa and Brattøra in Trondheim. (See Chapter 3 for more technical information about this specific platform.) As well as collision avoidance, this ferry will need a navigation system in order to fulfill its task of autonomous people transport.

Figure 1.1: The ferry prototype Milliampere as it, not so autonomously, leaves the quay. From the left: Brage Sæther and Emil Hjelseth Thyri. Photo taken by Nicholas Dalhaug.

An intricate part of a ferry's capabilities is its ability to dock. Therein lies the necessity of precise navigation and localization as the vessel approaches the quay. Using its on-board sensors, the ferry should be able to find its own location and

orientation, find where the quay is and therefore know its position and orientation relative to the quay. This can be done in many different ways, but an important part is knowing where the quay is, whether this is given before the docking as a map or figured out through data analysis and an understanding of what a quay is. The transport of people necessitates that the docking is safe, reliable and effective. These factors put requirements on the sensors and sensor fusion software used on the vessel, including those for localization.

Navigation in general needs good sensors, and this is especially true for autonomous vehicles. For a system to navigate autonomously, it needs to utilize several sensors and combine them to estimate its position and orientation. An influential step towards good measurement of the heading of vehicles like ships or planes was the invention of the gyrocompass, patented by Elmer Sperry in 1911 [7]. Since then, there has been an increasing effort in making sensors better, smaller and more affordable. Some typical sensor types in autonomous ship navigation today are Global Navigation Satellite System (GNSS), Automatic Identification System (AIS), Inertial Measurement Unit (IMU), Inertial Navigation System (INS), sonar, lidar and radar.

Even though Real-Time Kinematic Global Positioning Systems (RTK-GPSs) are accurate enough, GNSS might not always be available or trustworthy. That might happen in certain environments or if the signals are being jammed, intentionally or unintentionally, or attacked [49]. The vessel therefore needs to be equipped with other sensors that should be able to locate it. An ASV can then use the information gathered about the environment in methods for localization. It can fuse the data to a representation that can further supplement systems such as path planning and control, to finally make the vessel move towards the dock.

Many articles exist on localization using lidars. Some articles use features extracted as landmarks [73] [10] while others use scan-matching [71] [72] [20]. Some articles use a lidar placed on ground vehicles [73] [71] [10] while others place the lidar on water surface vehicles [46] [68] [22]. Some experiment in indoor environments [4] and some in outdoor environments [73] [71] [10]. But the author of this master's thesis has not found an article that surveys localization using a lidar on USVs, nor an article doing localization with a physically based sensor model of the lidar in an occupancy grid.

1.3 Problem description

This master's project is partly based on an earlier project about different Simultaneous Localization And Mapping (SLAM) implementations for autonomous docking using lidar [46]. That thesis is discussed in more detail in Section 2.3. It describes the inadequacy of certain pure implementations of known SLAM methods for the

autonomous ferry case. This leaves the question of how to go forward with the goal of autonomous localization of the vessel.

The goal of this master's project is to analyze methods for localizing the USV Milliampere using a lidar. This is in an attempt to find out how to go forward in the development of a localization method for the vessel. There are many different ways of doing this, as the literature overview in Chapter 2 states, but certain properties of the sensor must first be established. The problem has therefore been delimited to a narrower problem described next.

1.4 Delimitations

The problem of lidar localization has in this project been delimited to particle filters. This is because of its ability to represent multi-modal distributions and its weak assumptions on both the process model and the measurement model, this is explained in more detail in Section 6.1.

The problem is further delimited to the use of a physically based sensor model. This delimitation is made in an attempt to analyze the behavior of the localization method in an intuitive and understandable way where the artifacts from the relation between the map and the state are easy to understand.

The problem is also restricted to the use of an occupancy grid. This is because it is easy to make and edit by hand. It is also flexible with what structures it can represent, for example not only lines.

Lastly, since the problem is in the stage of analysis, it has been restricted to the case of simulations. A lot can be learned from simulations, for example about the behavior of a particle filter used in an occupancy grid with a physically based sensor model.

The platform Milliampere, see Chapter 3, has many different sensors, but the most relevant for this project are the lidar and the IMU.

1.5 Contribution

The developments in this master's project are supposed to supplement the already existing capabilities of Milliampere, as well as projects that are presently being developed. The vessel already has methods for path following, thrust allocation [62] and Dynamic Positioning (DP) implemented. Other projects currently in work are

for example navigation based on GNSS and INS [55]. The developments in this master's project therefore become one localization component in a bigger system of many components.

This thesis describes the execution of these tasks:

- Do a short literature review, Chapter 2. This gives an overview of some different methods and important topics in localization and shows different uses of localization as well as how the choice of method depends on the application
- Study the platform and data from earlier experiments, Chapter 3 and Appendix A.
- Create a map, in the form of an occupancy grid, in which localization can be done, Appendix B.
- Simulate localization with a particle filter and a physically based sensor model. The different approaches chosen are:
  - Generic Particle Filter (PF), Chapter 6. The theory about the dynamics of the system and the sensor models in Chapter 4 combined with the Bayesian state estimation theory described in Chapter 5 is used.
  - PF with Unscented Kalman Filter (UKF) used to make an importance density, Chapter 7.
  - PF with Iterative Closest Point (ICP) used to make an importance density, Chapter 8.

Through these tasks, an analysis has been made of UKF in the occupancy grid and ICP in an occupancy grid, both using a physically based sensor model. Based on the analyzes, a discussion about the problem at hand is made and concluded with suggestions of future work, Chapter 9 and Chapter 10. Some aspects on how to implement the methods have been discussed in Appendix C.

Chapter 2

Overview of literature

Some papers and other projects related to this master's project are described below in order to get an overview of different ways to do localization.

2.1 USV challenges

There have been much development within the field of USVs and with it also some challenges. This has been discussed in articles such as [35], which summarizes many elements of USV development with many references and is a good introduction to USVs from a Guidance, Navigation and Control (GNC) point of view. On the topic of navigation, sensors normally only give position and orientation measurements, and it is necessary to do state estimation in order to get velocity. This might for example be done by combining Global Positioning System (GPS) and IMU measurements in a state estimator such as a Kalman filter. For localization without GNSS a solution might be using a lidar for perceiving the environment and do SLAM, which is explained later in this chapter.

There are still many challenges related to single ASVs. Path planning and path re-planning have several important challenges. The same goes for control systems. On the topic of navigation, sensing technologies like radar, sonar and vision all have their problems. For radar this can be the detection precision, for sonar it can be sensitivity to noise and for vision it can be phenomena like fog or changing lighting conditions. One solution to these single challenges is to combine the sensors. The use of GPS and IMU together in a state estimator is common. But a challenge there is autonomous exploration in environments with degraded GPS signals, like near trees or under bridges. A solution to this is the use of active sensors for state estimation. Autonomous docking is a challenge in relation to control, and the article [35] states that the research at that time on USV navigation using active and passive ranging

sensors had been minimal. It references several papers on vision-based docking, and it references one article on a docking procedure for USVs [9]. Many of these challenges are still relevant today, but there have been some development in certain fields.

2.2 Localization and mapping

Localization and mapping are in general different tasks. They have both been studied in detail, see for example the books [60, 57]. Mapping is the act of creating a map from observations of the environment or objects to be mapped. It requires a known pose, position and orientation, for the sensor whose relative measurements then can be combined into a map. Localization is the act of finding or estimating the pose of a robot based on measurements and an understanding of the environment, often in the form of a map. This can be regarded as a sort of state estimation, where the state is the pose of a robot. These tasks generally depend on each other, which makes it difficult when both the environment and the pose of the robot is unknown. When the word "robot" is used here, it means the object whose state is to be estimated. This can for example be a boat.

Simultaneous Localization And Mapping (SLAM) is the act of simultaneously localizing a robot and mapping the environment. Several tutorials have been made on this topic [17, 5, 60, 57] and a survey of how far it has come is given in [11]. Stated simply, SLAM uses an initial pose estimate together with sensor measurements along a path and the registration of the same objects or landmarks several times, so-called loop closures, to calculate the most probable map and path estimates. An example of a SLAM method is FastSLAM [42, 41]. It uses a PF, explained in Chapter 5, for path estimation and Kalman filters for landmark location estimation. SLAM is very useful for applications where both the environment and the robot pose is unknown, but might be unnecessarily complex for other applications.

2.3 Earlier project on lidar SLAM comparison

An earlier master's project on the ferry Milliampere was [46]. The thesis compares the lidar SLAM methods Hector-SLAM [32], LOAM [74] and BLAM [45] that are all available as open-source in implementation Robot Operating System (ROS). It also goes into detail about different sensors and the equipment used on Milliampere. The thesis concludes that none of the pure implementations are adequately suited as a stand-alone solution for the autonomous ferry. It shows that Hector-SLAM is quite promising and suggests several possible ways of improving the implementation, but did not initially give a good enough map for autonomous docking.

2.4 Bayesian navigation using DME

Lidar and radar are examples of Distance Measuring Equipment (DME). An example of navigation using DME is where Karlsson and Gustafsson propose a radar based navigation system for a surface vessel [31]. The radar range measurements are compared with information from a digital sea chart and combined with a particle filter to give pose estimates. This becomes more robust than GPS, partly because of the higher energy of signals and much shorter distances. This method of navigating seems to work well from the article. It might also be easily converted to using a lidar or an other DME instead of the radar as no part of the method required anything radar specific.

The motion model [31] used is three dimensional, with Cartesian position and crab angle, the angle between the velocity vector of the ship and the stem of the ship. They also measure the longitudinal and lateral speed of the ship so that less states need to be estimated. The measurement model is range measurements with noise. So the particle filter does for every iteration a prediction using the motion model. After the prediction, there is a measurement step that compares the real measurements with simulated ones, from a database of expected land areas, for each particle. The simulation might be some sort of ray-casting, but this is not specified. Then the re-sampling of the particle filter is done and the method continues. The particle filter is explained in more detail in Chapter 5.

2.5 Ray tracing and likelihood fields

A lidar can not only be used to localize USVs, but robots in general, including autonomous cars. A master's thesis from Chalmers University [26] describes how to localize an autonomous car using a lidar and a particle filter. It tries out two different sensor models, the ray tracing model and the likelihood model, both of which are described in Chapter 4. The likelihood model is taking the measured rays and for each ray find the shortest distance from the end point to an obstacle in the map. This is faster than the ray tracing model by not having to simulate rays for each particle, but just project the end points of the measurements onto the closest object. In the beginning of the authors's project, a simulation environment was constructed in which different localization methods could be tried out. And the authors explain through this how the lateral and longitudinal error behaves. The thesis concludes that both methods work very well based on logged data from actual driving and ground truth GPS data.

2.6 3D likelihood fields

Another project using a particle filter and a lidar for localization is described in the paper by Merriaux et al. [40], even though the paper focuses a lot on the map representation with regards to memory and time efficiency. It was made for the ARGOS Challenge contests in which a robot should navigate a complex known industrial environment. The method uses what they call a 3D likelihood field that is a map with likelihoods of lidar impact and introduces hybrid octrees. Together with the hybrid octree map, a particle filter is used. The measurement function uses ray tracing to find the impact in the map for each particle in the particle filter. The given probability likelihood is then the distance between this point and the impact of the corresponding ray in the lidar measurement. An IMU was used for the motion update. Their hybrid octree map representation is deemed faster than the regular octree, and relatively smaller and smaller as the map size increases. The localization is said to work fairly well also with dynamic changes made by the jury in the contest. The paper concludes with stating that the method is fast and robust and that the map representation is suited for complex environments and embedded systems.

2.7 Maritime RobotX Challenge

The Maritime RobotX Challenge is a competition for students [52]. It was created to support the advancements within autonomous vehicle technology, focusing on ASV platforms and sensors. The latest and third competition was held in 2018 in Hawaii. Some of the tasks in this challenge are to autonomously pass through gates, avoid obstacles, find totems, read patterns and symbols, and dock. The docking task used a dock similar to the one in Figure 2.1, where the ASV should find the correct dock through symbol matching and then dock there.

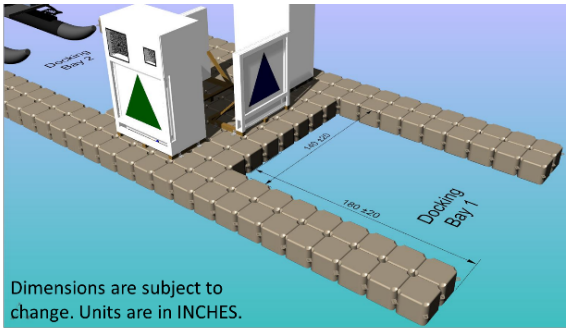


Figure 2.1: An illustration of the dock in the Maritime RobotX Challenge 2018. Courtesy of RoboNation [52].

One team that competed in the competition in 2016 was from Flinders University

[68]. They primarily used a Real Time Kinematic (RTK) GPS for navigation. Obstacles were located using a Simrad 4G maritime navigation radar and a Velodyne LiDAR HDL-32 lidar. To find obstacles and localize relative to them, the FastSLAM algorithm was used and was implemented using the Point Cloud Library (PCL) and ROS. They removed unwanted points on the shore by restricting points to be within a polygon and removed water reflection by setting a minimum intensity threshold. Thereafter clustering was done and a list of landmark measurements was sent to the SLAM node. The classification of 3D shapes was done using an ICP method, see Chapter 8 for a more detailed explanation, to compare point cloud clusters to template models, giving a best-fit score and a relative transformation. The map consisted of sparse objects with identifiers and properties relating to the relevant tasks, like type, color and shape. Different sensors were fused through a particle filter, more precisely a bootstrap version since the objects are stationary.

As several other teams, also the one from the University of Florida used an occupancy grid as a map [22]. They used the lidar data to find objects through clustering, and treated them as obstacles. The docking procedure consisted of finding the correct symbol that marked the dock, and approaching it while the obstacle avoidance system tried not to collide with the dock.

2.8 Changing environments

Many of the methods discussed, and localization and SLAM methods in general, assume a static environment, but the real world can rarely be considered static. One obvious example would be that the lighting conditions change over time, from night to day. Another example is the weather, changing from calm to a storm. These example might be less of a problem for indoor implementations. Examples there might be the rearranging of furniture, doors that open and close or people walking in the environment. Most environments change over time, and mapping such an area would mean knowing exactly how it changes. Knowing this, the map could further be used in localization. The map representation in a SLAM method would have to take the changes in the environment into account.

A paper that tries to solve the problem of tackling dynamic environments in localization is [34]. There they assume that the map consists of a set of features where the number of features is not certain. The observation of each feature is also uncertain. This is contrary to [61] where there is not a set of features, but the dynamic elements are embedded into an occupancy grid and probabilities of observing occupied cells.

Tipaldi et al. [61] test their method successfully in a parking lot where cars move, using a combination of a Rao-Blackwellized particle filter and a hidden Markov model for their dynamic occupancy grid. The sensor used is a laser range finder.

This results in the ability of using the original map and changing it as new measurements come in. A weak prior on the map and a weak prior on the initial pose is used, making this a method in between SLAM, where the map is assumed unknown, and global Monte Carlo localization, where the map is assumed known. The Rao-Blackwellized particle filter is used for estimating the state trajectory and the map. The dynamic occupancy grid, like most occupancy grids, assumes the independence between observations given the map and among cells. Each cell also has a hidden Markov model with a specified transition probability, observation model and initial state distribution. The transition probabilities in the Markov model are learned through expectation maximization. What differentiates this map and state estimation from SLAM is the separation of robot trajectory and map estimation. Where SLAM estimates the distribution $p(x_{1:t}, m_t | z_{1:t}, u_{1:t-1}, m_0, x_0)$, this method separates

$$p(x_{1:t}, m_t | z_{1:t}, u_{1:t-1}, m_0, x_0) = p(m_t | x_{1:t}, z_{1:t}, m_0, x_0) p(x_{1:t} | z_{1:t}, u_{1:t-1}, m_0, x_0),$$

since $p(m_t | x_{1:t}, z_{1:t}, m_0, x_0)$ can be computed analytically, making the estimation more efficient. Here x is the state, m is the map, z are the observations and u are the control inputs. The paper finally demonstrates outperforming the popular Monte Carlo localization by updating the map and not lose track when the initial map was not correct [61]. It concludes with stating that the described method is suited for long term operations in changing environments. Monte Carlo localization is explained later in the form of the particle filter, Chapter 5.

2.9 The broad topic of localization

Even though the methods discussed above were restricted DME such as lidars, every method is different. The topic of localization is broad, with some methods being pure localization while others used SLAM. State estimation can be done with for example particle filters, Kalman filters or both. The map representations can for example be lines or occupancy grids, in a traditional sense or with more complex estimation for each grid cell. The robots that are localized range from small almost toy-like platforms to vehicles intended for human transport. In some cases, it might be enough to do clustering to find objects, and find relative pose for that object using a ICP method. In other cases, finding the relative pose of clusters of data might not be as simple as doing ICP since the data comes from a larger map in which the the data can fit to several poses. ICP is explained in detail in Chapter 8. The best way to do localization depends on the application, which in this project is the localization of an autonomous ferry.

Chapter 3

Platform and map

This chapter describes the equipment used and the environment in which the ferry should work.

3.1 Milliampere introduction

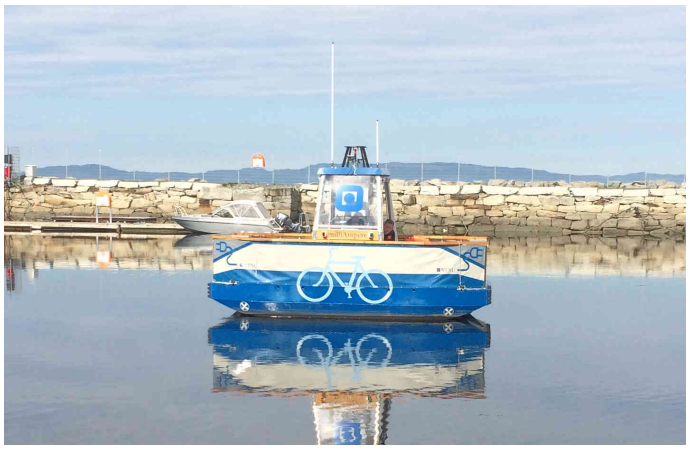


Figure 3.1: Image of Milliampere being tested. On-board is Brage Sæther and Emil Hjelseth Thyri. Photo taken by Nicholas Dalhaug.

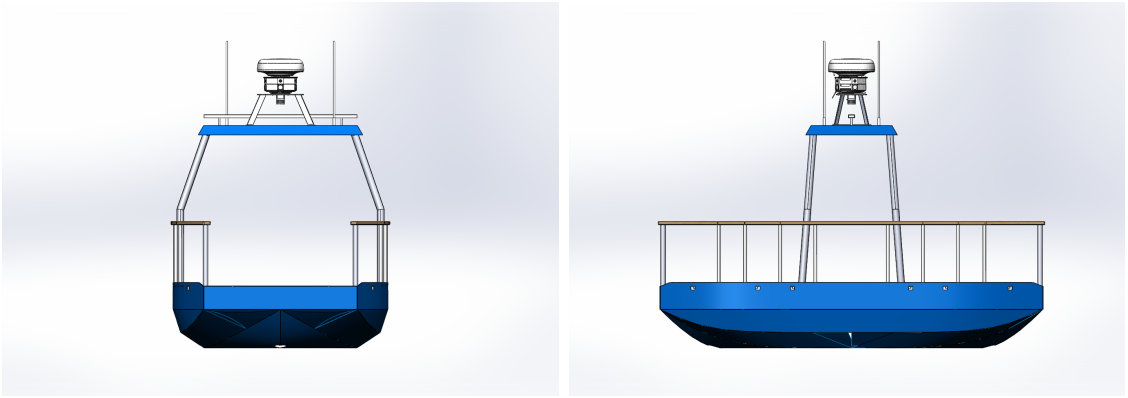
Milliampere, see Figure 3.1, is a part of the ongoing Autoferry Project [3] at NTNU. It is a half-sized prototype of an autonomous ferry that is being designed. The ferry is intended to become a fully autonomous ferry that will roam the passage between Ravnkloa and Brattøra in Trondheim. It will be electric and the ferry should be able to transport people across the approximately 110 m wide passage back and forth throughout the day. On each side of the passage there is a dock. The northern dock

is fixed to concrete and the southern dock is floating and points out in the water, see Section 3.3 for a more detailed description of the quays in question.

Several project have been conducted within the Autoferry project. The vessel has methods for path following, thrust allocation [62] and DP implemented. One ongoing project right now is for navigation based on GNSS and INS [55].

3.2 Vessel

The ferry Milliampere is itself approximately 3 m tall, see an illustration of the ferry in Figure 3.2. Most of the sensors are placed on top of the vessel. There are openings both in the front and in the back for pedestrians and cyclists to board.



(a) Front (b) Side

Figure 3.2: Illustration of the ferry Milliampere. Source: Egil Eide.

3.3 Quay

In this master's project, there are different relevant quays. The most obvious one is the one where the ferry is intended to roam autonomously, the Ravnkloa and Brattøra quays. Another one is the Brattøra harbour, which has been used for testing and implementation of methods. Both are described in more detail below.

3.3.1 Ravnkloa and Brattøra quays

The quays at Ravnkloa and Brattøra are the ones at which the ferry is intended to roam autonomously. They are shown in Figure 3.3. The passage is approximately

110 m wide. There is a certain amount of traffic there, including the boat that takes tourists to the island Munkholmen. The Munkholm boat leaves from the southern quay and goes through the tunnel in the top of the figure on its way to Munkholmen. It is therefore some times during the summer docked at the southern quay, and the autonomous ferry should be able to handle other vessels being docked and moving in the passage as it transports people.

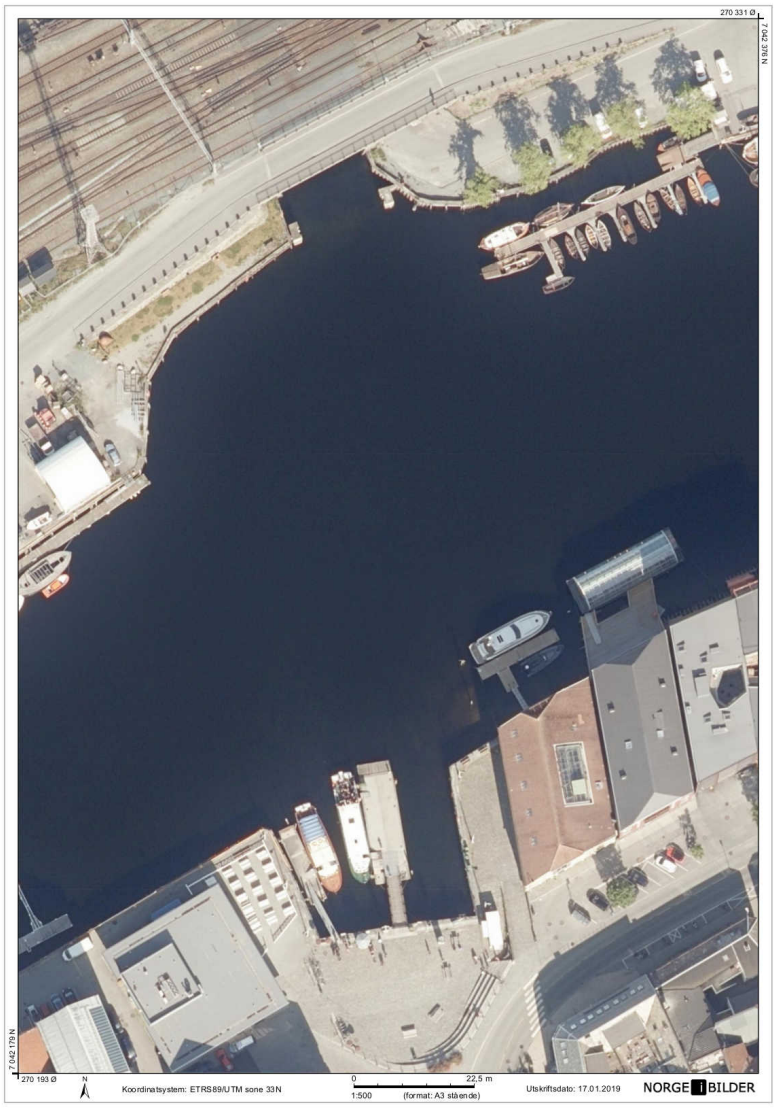


Figure 3.3: The figure shows an aerial photo of the passage between Ravnkloa and Brattøra in Trondheim, including the docks on each side. The northern dock is to the left of the tunnel, while the southern dock is the floating structure in the middle lower part of the image. © Statens kartverk, Geovekst og kommunene, Trondheim 2017.

The northern most quay is fixed to concrete and is made of wood. It is to the left of the tunnel in the top part of Figure 3.3. This quay is not the most important

part of the project. It might be changed or moved, and it is assumed that it will be possible to dock here if it is possible to dock at the other quay.

The biggest floating quay on the southern side of the passage is for now the primary quay to which the vessel should be able to dock autonomously. See the bottom part of Figure 3.3. It is approximately 0.5 m tall, over the water line. The relevant place for the USV is specifically on the tip of the biggest floating quay. The surrounding area is made of concrete with several places for boats to dock. It is a place with many people and shops nearby.

3.3.2 Brattøra harbour

For some testing and developing of methods, the Brattøra harbour was used. See Figure 3.4 for a map of the area. It includes several places to dock, as can be seen from the boats in the image. There are often boats there, also during the experiments related to this master's project. But the area includes many different features useful for tracking. It is also quite large, giving much space for testing of an autonomous vessel.

3.4 Sensors

The primary navigation system on the ASV is GNSS. But the ferry is equipped with the following sensors related to this project:

- GNSS in the form of a RTK-GPS
- IMU
- Lidar

The workings of some sensors are described in Chapter 4, and more details about the specific sensors used is described below.

3.4.1 Lidar

The lidar used in this project is the Velodyne LiDAR Puck, also known as VLP-16, see Figure 3.5. For an explanation of the workings of a lidar, see Section 4.4. This is a 3D lidar, capable of measuring with a 360° horizontal field of view and 30° vertical field of view [65]. It has a range of 100 m, a range accuracy of ±3 cm (typical) and it fits approximately inside the palm of a hand. As the name suggests, it has 16

Figure 3.4: Aerial photo of Brattøra harbour. Inside the harbour there are mainly two floating quays, excluding the top one where there are no boats. Those two floating quays have several boats docked at the time the image was taken. © Statens kartverk, Geovekst og kommunene, Trondheim 2017.

vertically spaced IR emitters which spin horizontally clockwise, each of which has a paired IR detector.

The vertical angular resolution is 2.0°, from −15.0° to 15.0°. This includes rays at ±15° and ±1° and excludes a ray at 0°.

The horizontal angular resolution is 0.1° to 0.4°, and the rotation rate is 5 Hz to 20 Hz with a deviation of $\pm 3\,\mathrm{min}^{-1}$ [66]. This deviation is due to a feedback controller being responsible for keeping the right rotation speed. More precisely, the lidar can rotate between $300\,\mathrm{min}^{-1}$ and $1200\,\mathrm{min}^{-1}$ including the increments of $60\,\mathrm{min}^{-1}$ [66, p. 50]. The firing timing is fixed to 55.296 µs per firing cycle. This gives an azimuth resolution satisfying

$$\Delta\alpha = r\frac{\mathrm{rot}}{\mathrm{min}} \cdot \frac{1\,\mathrm{min}}{60\,\mathrm{s}} \cdot \frac{360^\circ}{\mathrm{rot}} \cdot \frac{55.296 \cdot 10^{-6}\,\mathrm{s}}{\text{firing cycle}}, \tag{3.1}$$

where r is the number of rotations per minute the sensor does.

This results in a data throughput of $\sim 300\,000\ \frac{\text{points}}{\mathrm{s}}$. The lidar also has different re-

turn modes, strongest, last and dual. They say something about what measurements to gather, as lasers can hit multiple objects.

Figure 3.5: A picture of the lidar used in this project. Courtesy of Velodyne LiDAR.

This project considers the number of horizontal lidar rays to be 1800 based on the relevant data from earlier experiments, see Appendix A for information about the experiment data.

3.4.2 GNSS

The Global Navigation Satellite System (GNSS) equipped on the vessel is a RTK-GPS. This sensor has been important for finding the actual state in the mapping process in Appendix B, but is at the same time the sensor which this project tries to localize without.

3.4.3 IMU

An Inertial Measurement Unit (IMU) is a sensor that measures motion. It is often equipped with three or more accelerometers for estimating linear acceleration and three or more angular rate sensor to estimate the rotational velocity. Sometimes it can also include three or more magnometers, used to measure the magnetic field vector which enables an absolute orientation estimation. IMUs can through these measurements be very useful for state estimation resulting in an Inertial Navigation System (INS), usually by some sort of integration to find position and orientation which would require some noise compensation. IMUs are also generally cheap sensors, depending on the noise magnitude that is acceptable.

IMUs can be used for velocity estimation, which is relevant in this project for finding the noise on the constant velocity model.

3.5 Software

Some of the software used in this project is:

- ROS
- Rviz
- OctoMap
- Open Source Computer Vision Library (OpenCV)
- PCL

They are described in more detail below.

3.5.1 ROS

Robot Operating System (ROS) is an open source communication layer above the host operating system [50]. It works by running a "roscore" that handles the overhead of the communication, making the development of independent "nodes" possible. These nodes are the different tasks running on the system, and they communicate over "topics" by subscribing to "topics", publishing to "topics" and using standard message types. In ROS the standard for representing transformations between different reference frames, such as different parts of a robot or different robots, is to publish the transformation on a topic called "\tf" using a standard message type "tf/tfMessage". ROS is made to simplify communication between different parts of robotic systems as a standardization of protocols. For a tutorial on ROS, the author suggests reading [47].

One of the big benefits of using ROS is the recording and playback of "rosbags". A "rosbag" is a file containing all messages sent over topics in the period of recording, including when each message was sent. This enables publishing the same data on the same "topics" in the exact same sequence and after the same elapsed time as when the recording happened. When sensor data is captured, "rosbags" makes possible the development of "nodes" as if they were tested in the exact same setting as when the data was captured. It is useful for the testing of methods in robotic systems.

3.5.2 Rviz

Rviz is a ROS package for visualizing different data through the ROS framework [70]. It handles several of the standard message types in for example the "sensor_msgs" and "nav_msgs" packages. It can for example display point cloud data through the "sensor_msgs/PointCloud2" message or display the robotic platform through the use of a description file and the different transforms sent over ROS. Rviz is used as a node that subscribes to the relevant topics on which the data is published and is operated through a Graphical User Interface (GUI).

3.5.3 OctoMap

OctoMap is a framework for generating 3D models of the environment [27]. It is based on octrees and uses probabilistic occupancy estimation. Octrees divide each node into eight octants recursively until a minimum size. This can be utilized to reduce memory demand of maps by representing eight octants or more as one node. Occupied space is estimated using the endpoint of a measurement from DME, and free space is the space between the sensor and that endpoint. The software can only do mapping, and the pose of the sensor needs to be supplied in order to get a correct map. OctoMap has been integrated into ROS and can be used by sending data over topics. The software is also able to reduce the map to a 2D occupancy grid.

3.5.4 OpenCV

Open Source Computer Vision Library (OpenCV) was designed to be a common infrastructure for applications within computer vision and machine learning [48]. The library contains many methods and algorithms, from displaying of images to facial recognition software. It has been downloaded over 14 million times and is used by companies, researchers and hobbyists alike. It has interfaces for C++, Python, Java and Matlab. The library also has an interface for ROS, including the translating of different message types to images. This has been useful for creating images from occupancy grids in ROS.

3.5.5 PCL

Point Cloud Library (PCL) is an open source library made for efficient handling of point clouds [54]. With faster and more precise DME sensors, efficient handling of the corresponding sensor data becomes more important. The library provides

methods for the most important building blocks of point cloud handling, for example for filtering and point cloud registration. It is a templated C++ library with definitions of different point cloud data structures. PCL also has ROS integration with translation from ROS data types to PCL data types.

3.6 Spatial resolution

From the properties of the different parts of the platform, it is possible to calculate the resulting spatial resolution of the lidar at certain distances. It is not necessary to check the horizontal spatial resolution of the lidar because of the grater horizontal angular resolution and that the quay is wider than it is tall. It might be desirable to see the quay in the lidar data at approximately a distance of $l = 10\,\text{m}$, see Figure 3.6. The lidar is at a height $h = 3\,\text{m}$ above the water. The vertical angular resolution of the lidar is $\Delta\theta = 2°$. θ is the angle for a lidar ray, and h_θ is the resulting height above the horizontal ray for different rays.

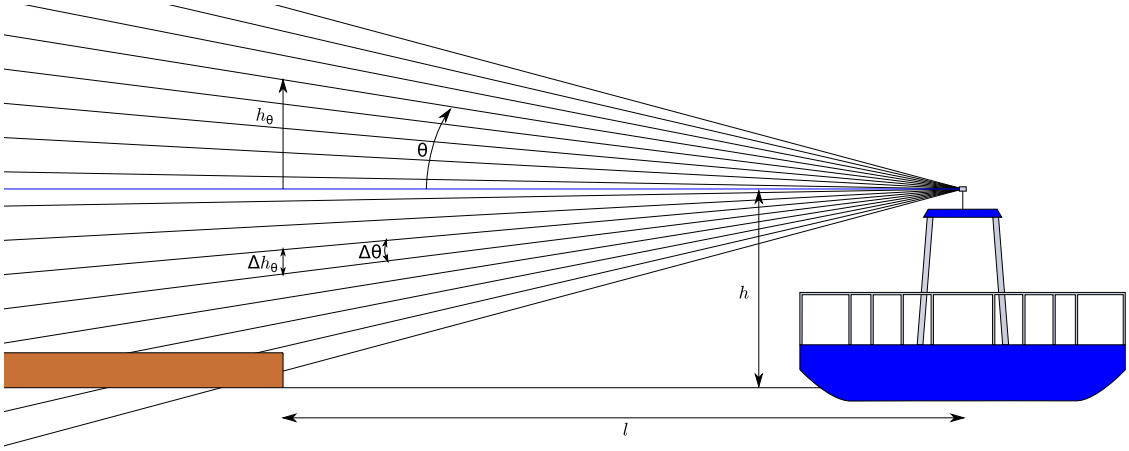


Figure 3.6: The figure illustrates the ferry as it is a distance l from the quay. The blue line illustrates the horizontal line from the lidar, there is no lidar ray there at 0°.

3.6.1 Ray height

The height of a ray, over the horizontal ray of the lidar, at a distance l horizontally from the lidar can be calculated as

$$h_\theta = l \cdot \tan(\theta),$$

see Figure 3.6 and the calculation in Table 3.1. It shows the calculations of h_θ for each ray and the resulting absolute difference between rays, Δh_θ, taken between the ray of the column it is at and the next column.

θ [deg]	-15	-13	-11	-9	-7	-5	-3	-1	1
h_θ [m]	-2.679	-2.309	-1.944	-1.584	-1.228	-0.875	-0.524	-0.175	0.175
Δh_θ [m]	0.371	0.365	0.360	0.356	0.353	0.351	0.350	0.349	

Table 3.1: Spatial resolution calculations. The values above 0° are the same, but with the opposite sign.

The calculations show that the best resolution at the distance $l = 10\,\text{m}$ is $\Delta h_\theta = 0.349\,\text{m}$. If the quay is 0.5 m tall, the quay will be visible, but only with at most two rays. It also shows that the lowest ray, at $\theta = -15°$ is very close to the water line, $|h_\theta| = 2.679\,\text{m}$ is almost equal to $h = 3\,\text{m}$. So the quay will not be visible from the vessel if it is much closer than this. This also does not take self occlusion into account, the fact that parts of the vessel might be in the way.

This sensor configuration height will result in the tip of the quay not being seen in the lidar data when the vessel is close to the quay. The resolution of the sensor can also make the tip of the quay not showing in the data at longer distances, see Appendix C for an illustration of rays going above the quay. This increases the demand for some sort of outlier detection for the data and the ability to localize relative to other surrounding structures.

3.6.2 Other configurations

To find where the ray hits the water line, this relationship is used:

$$l = \frac{h}{\tan(\theta)},$$

where the variables are as given in Figure 3.7. θ is the angle of the ray, 15° in this case. h is the lidar's height above the water line, and l is the distance from the vessel to where the ray intersects the water line. For $h = 3.00\,\text{m}$ this gives $l = 11.2\,\text{m}$. To find where the lowest ray hits the top of the quay, the calculation becomes

$$l = \frac{h - 0.500\,\text{m}}{\tan(\theta)} = \frac{3.00\,\text{m} - 0.500\,\text{m}}{\tan(15°)} = 9.33\,\text{m}.$$

So if the vessel is any closer to the quay than $l = 9.33\,\text{m}$, the front of the quay is not visible and can not be used to localize the quay. The top of the quay is still visible in the data, but it does not give information about the horizontal position of the lidar nor the vessel.

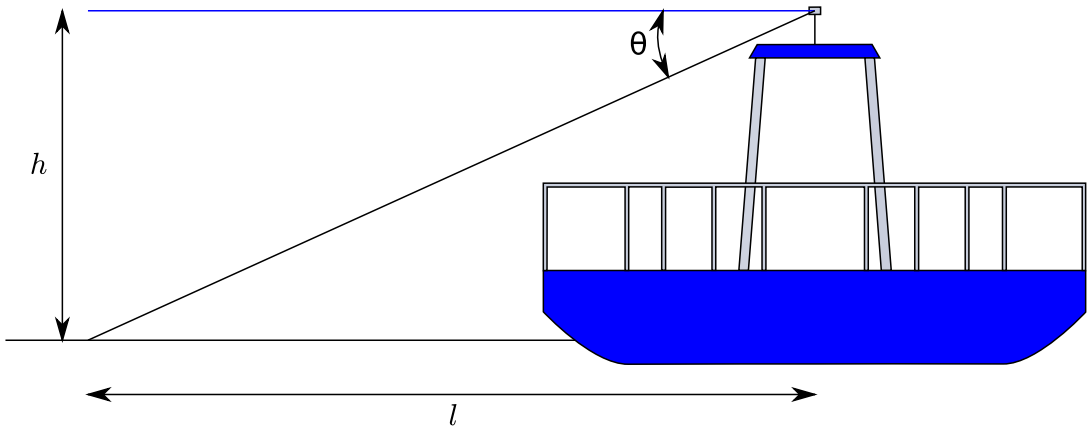


Figure 3.7: Illustration of the vessel, one ray and where it hits the water line.

If for example it had been desirable to see the front of the quay at $l = 5.00\,\text{m}$, this would result in the lidar being placed at $h = 5.00\,\text{m} \cdot \tan(15.0°) + 0.500\,\text{m} = 1.84\,\text{m}$. That would give a resolution of about $\Delta h_\theta = 0.2\,\text{m}$. An other possibility would be to rotate the lidar, so that the 0°-line is not horizontal. But then the data would not represent both sides of the vessel equally well. The point cloud would also be angled.

3.7 Map representations

There are many ways of representing a map. A 2D map can for example consist of lines. But like many map representations, this would restrict what the map can model, in this case only the edge of 2D structures. In this project, the map used is a 2D occupancy grid, see [19] for a thorough description of how occupancy grids work. Occupancy grids are maps that rasterize the real world into what we can call grid cells in 2D and voxels in 3D, similar to the pixels of images. Each cell holds the probability of the cell being occupied or not, a probabilistic occupancy grid, or each cell can simply be free, occupied or unknown. This map representation is flexible when it comes to the structures it can represent, but it is limited to a certain resolution. It is also easy to edit manually since the grid can be converted back and forth to an image.

The occupancy grid is a location-based map representation, where the indexing is based on the position [60, p. 152]. This is contrary to feature-based maps, where each feature has an index. Since occupancy grids are location-based, they are also volumetric [60], they label all locations in the map. This property enables the

occupancy grid to not only say where objects are, but also where there are no objects, the free space.

3.8 Reference frames

Motion and measurements are expressed relative to reference frames. The world reference frame is commonly placed on the surface of the earth with the approximation that it is inertial. This is a good approximation for most systems. The orientation of the world frame can for example be in the North East Down (NED) configuration. This means the x-axis points north, the y-axis points east and the z-axis points down into the ground [21, p. 16]. The reference frame used in this project is in the East North Up (ENU) configuration, with x-axis east, y-axis north and z-axis up. This is because the standard of ROS [39] specifies that as well as the map frame being fixed, the z-axis of the map frame should point upwards and that the map frame should be fixed.

3.9 Mapping

To do localization it might be necessary with some sort of map. The earlier master's project on SLAM [46] did not give a map in the sense of a single file, but rather implementations of different SLAM methods that give maps over a topic. And also since that project was not finished in the beginning of this project, the author needed to make a map. Because of the simplicity in representation, the flexibility and the fact that it can easily be visualized as an image, the author chose to make an occupancy grid. Details of how this is done is explained in Appendix B. The result is shown in Figure 3.8.

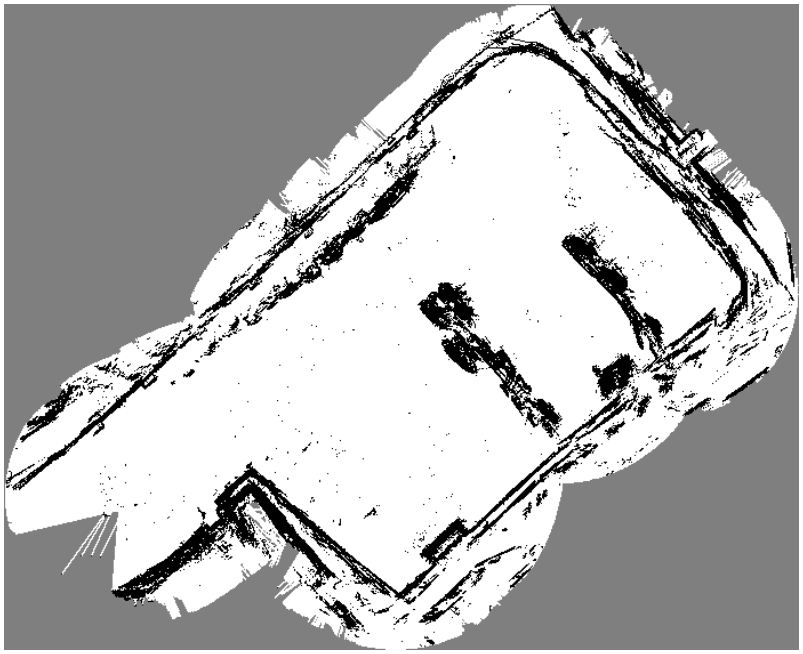


Figure 3.8: The occupancy grid from the mapping as an image, when the mapping range was 50 m and the resolution was 0.5 m. Up is north, right is east. The white cells are free space, the black cells are occupied space and the gray cells are unknown.

Chapter 4

Dynamics and sensor models

The models explained in this chapter are intended to be used for localization. More precisely, they will be used in a probabilistic framework for state estimation. The process model will be used for both estimation and for simulations, while the measurement model will use one model for the estimation and an other for simulating measurements.

4.1 State estimation

The state and measurements for a time independent discrete time nonlinear autonomous dynamic system is in the state estimation literature often written on the state space form

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{v}_{k-1}) \\ \boldsymbol{z}_k &= \boldsymbol{h}(\boldsymbol{x}_k, \boldsymbol{n}_k), \end{aligned} \tag{4.1}$$

where $\boldsymbol{f}(\cdot)$ and $\boldsymbol{h}(\cdot)$ are nonlinear functions corresponding to a process model and a measurement model respectively, $\boldsymbol{x}_k$ is the unobserved state at time step k, $\boldsymbol{z}_k$ are the measurements, $\boldsymbol{v}_{k-1}$ is the process noise and $\boldsymbol{n}_k$ is the measurement noise. The nonlinear function are considered known, but not necessarily analytically linearizable.

In this case autonomous means that the system only depends on the state at the last time step. This is without any other input to the process. If the process model did also depend on an input $\boldsymbol{u}_k$, the function could for example become $\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{v}_{k-1})$. This could for example be the case if the velocity of a vessel was not constant, but was rather given to the system as an input.

4.2 Ship dynamics

Ships have complicated dynamics, and the modeling of them requires simplification. One possible simplification is reducing the motion of the ship to three Degrees Of Freedom (DOF). Using the state $\boldsymbol{\eta} = \begin{bmatrix} N & E & \psi \end{bmatrix}^\mathsf{T}$ with velocity in body coordinates $\boldsymbol{\nu} = \begin{bmatrix} u & v & r \end{bmatrix}^\mathsf{T}$ the simplified dynamics become [21, p. 134]

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}_\mathrm{Z}(\psi)\boldsymbol{\nu} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{\nu}, \tag{4.2}$$

$$(\boldsymbol{M}_\mathrm{A} + \boldsymbol{M}_\mathrm{RB})\,\dot{\boldsymbol{\nu}}_r + \boldsymbol{N}(\boldsymbol{\nu}_r)\,\boldsymbol{\nu}_r = \boldsymbol{\tau} + \boldsymbol{\tau}_\mathrm{wind} + \boldsymbol{\tau}_\mathrm{wave}. \tag{4.3}$$

N is the north position, E is the east position and ψ is the heading relative to a fixed world frame. $\boldsymbol{M}_\mathrm{A}$ is the added mass matrix, $\boldsymbol{M}_\mathrm{RB}$ is the rigid body mass matrix and $\boldsymbol{N}(\boldsymbol{\nu}_r)$ consists of the nonlinear Coriolis and damping terms and is dependant on the relative velocity $\boldsymbol{\nu}_r = \boldsymbol{\nu} - \boldsymbol{\nu}_c$. $\boldsymbol{\tau}$ is the force vector made by the actuators of the ship, $\boldsymbol{\tau}_\mathrm{wind}$ and $\boldsymbol{\tau}_\mathrm{wave}$ are forces due to the environment and the hydrodynamic forces $\boldsymbol{\tau}_\mathrm{hyd}$ are included in the mass matrix and the nonlinear term. This formulation assumes irrotational constant ocean currents $\boldsymbol{\nu}_c$, movement only in the horizontal plane, a parametrization of the Coriolis matrix independent of linear velocity and no hydro-static forces $\boldsymbol{\tau}_\mathrm{hs}$.

After simplifying the dynamics of a ship a lot, the model is in a sense too inaccurate for many application and still too complex to be used as is. The kinematics in (4.2) are accurate assuming motion in only three DOF, but the kinetics in (4.3) have many tune-able parameters that might be sensitive to uncertainties. The different matrices will have to be estimated in order to use this model. It can also be simplified further by for example decoupling surge motion from sway and yaw. See [21] for more details about ship modelling and navigation.

4.3 Process model used

The dynamics are simplified by assuming only three DOF, see Section 4.2 for theory about ship dynamics. Using an ENU reference frame, see Section 3.8, the state vector becomes $\boldsymbol{x} = \boldsymbol{\eta} = \begin{bmatrix} E & N & \psi \end{bmatrix}^\mathsf{T}$, which is respectively east position along the x-axis, north position along the y-axis and heading around the z-axis with zero at the x-axis, also called yaw angle. The corresponding velocity vector becomes $\boldsymbol{\nu} = \begin{bmatrix} u & v & r \end{bmatrix}^\mathsf{T}$. That is the speed along the body x-axis, y-axis and yaw rate respectively, see Figure 4.1 for an illustration of the chosen body axes.

The dynamics are reduced to the kinematics in (4.2) with $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$ as described

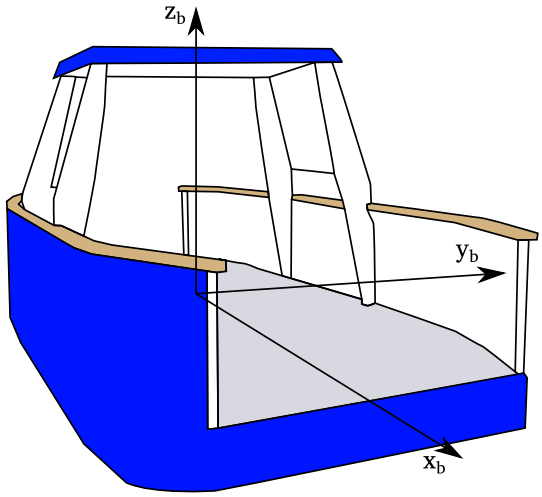


Figure 4.1: Vessel with body axes shown. x_b is pointing through the front of the vessel, and z_b points up from the vessel.

above:

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}_{\mathrm{Z}}(\psi)\boldsymbol{\nu}, \tag{4.2 revisited}$$

see Section 4.2 for the background theory. To approximate this on a computer, the Euler method is used

$$\boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_k + \boldsymbol{R}_{\mathrm{Z}}(\psi)\boldsymbol{\nu}_k\Delta_k,$$

where Δ_k is the time step between iteration k and $k+1$. This assumes that the vessel speed is either constant and given or is given to the simulation in each iteration. It can be simulated by for example assuming $\boldsymbol{\nu}$ is constant and has noise on it, with noise as described in Appendix A.4. The process model then becomes

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{R}_{\mathrm{Z}}(\psi)\left(\boldsymbol{\nu}_{k,\mu} + \boldsymbol{\nu}_{k,\sigma}\right)\Delta_k, \tag{4.4}$$

where $\boldsymbol{\nu}_{k,\mu}$ is the mean velocity and $\boldsymbol{\nu}_{k,\sigma} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Sigma}_\nu\right)$ is the noise. The elements of the diagonal

$$\boldsymbol{\Sigma}_\nu = \begin{bmatrix} \sigma^2_{\text{surge}} & 0 & 0 \\ 0 & \sigma^2_{\text{sway}} & 0 \\ 0 & 0 & \sigma^2_{\text{yaw}} \end{bmatrix}$$

can be deduced from Table A.1. This constant velocity model was chosen so that no new states needed to be added and the particle filter still only needs to handle three states. A realization of the path for the vessel with process noise is shown in Figure 4.2.

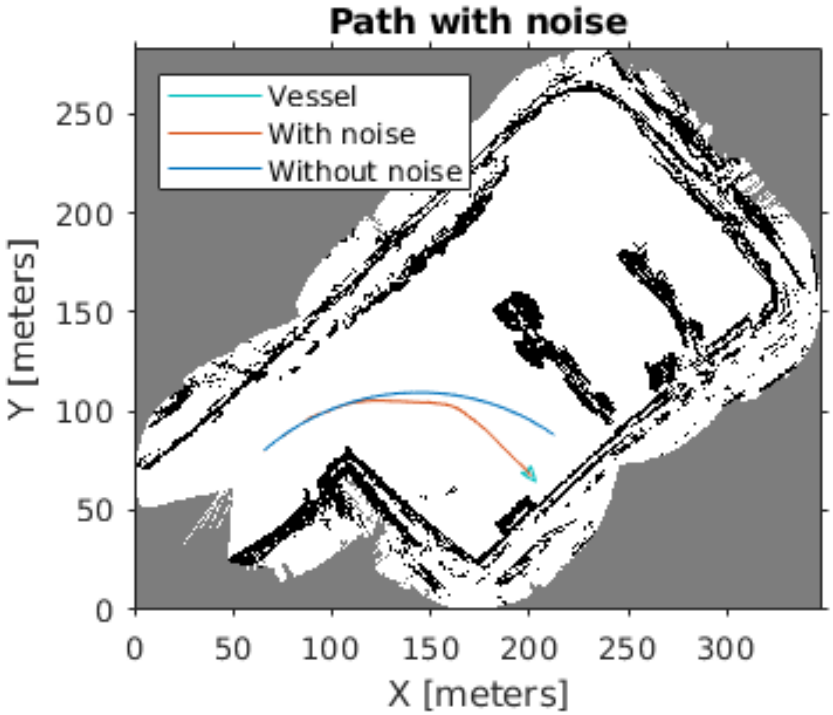


Figure 4.2: Realizations of the path taken by the vessel with and without the process noise, starting from the same position. The map is the same as Figure 3.8, but with noise removed and made to represent the area better.

4.3.1 Considerations

The kinetics have been discarded for simplicity and lack of data and time to model the vessel. If the vessel is modelled and the thrust $\boldsymbol{\tau}$ from the actuators is known, the vessel dynamics could be simulated more accurately. This encourages cooperation with other parts of the ferry project, like the control systems that probably have a model of the vessel. On the other hand, the model would not fit very well because of the complexity of ship dynamics and might not be desired for use in state estimation.

Instead of using a constant velocity, we can assume $\boldsymbol{\nu}_{k,\mu}$ is given. This can for example be the case if the velocity is measured directly, or if INS is used. An INS can for example integrate the IMU acceleration and rotation speed measurements to get a dead reckoning velocity measurement. This velocity error can then be analyzed to find the variance in the Gaussian model used above and the process model will otherwise be the same.

4.4 Lidar

The lidar is a DME that uses the Time Of Flight (TOF) methodology, see Figure 4.3. Similarly to the radar, it uses electromagnetic waves together with precise time measurements to measure the distance to objects. The lidar uses IR light instead of radio. The laser pulse traverses through the air until it hits an object. Then some of the energy is reflected, traverses back and is measured by a detector. Some lidars can then measure both the position of collision and intensity of light reflected. With the speed of light known and the time from excitation through reflection and to measurement known, the distance can be calculated as

$$R = \frac{c \cdot t}{2},$$

where R is the distance to the object, c is the speed of light and t is the time measured.

One ray of a 3D lidar can be represented by the azimuth angle α and the vertical angle ω as shown in Figure 4.3. The 3D coordinates of a point measured with distance R can then be calculated as shown in the figure, transforming from spherical coordinates to Cartesian coordinates. The high frequency of a lidar make it a sensor with big data throughput and often necessitates a lot of computational resources.

One full horizontal rotation of the lidar is often considered as one measurement of the environment. These measurements have mainly two representations, the point cloud and the laser scan. Since every ray gives three Cartesian coordinates from a 3D lidar, one full rotation gives many 3D points that are together called a point

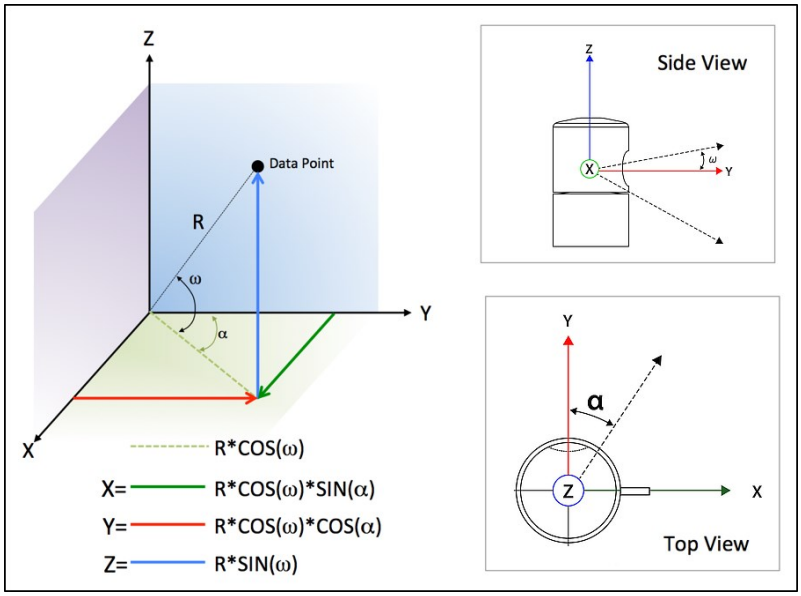


Figure 4.3: Illustration of how a lidar works. Courtesy of Velodyne LiDAR [66].

cloud. Each point can also have data about the intensity of light reflected at that position, see Section 3.4.1 for an example of a 3D lidar and Appendix A.3 for an analysis of point cloud data. The laser scan represents 2D DME measurements. They can still represent full rotations, but contain data in the form of azimuth angles α and distances R instead of three coordinated. Both types of lidar data are available in ROS, see Section 3.5.1 for an explanation of ROS. And these full rotation measurements can then be used for example in localization with a good sensor model.

4.5 Sensor models

A physically based sensor model for the lidar has been analyzed in for example [53, 56]. A sensor model for the lidar, and principally many different sensors, accounts for the uncertainty in the measurements and the relationship between the measurements and the underlying state variables [60, pp. 149]. Such a model can be formulated as $p(\boldsymbol{z}_k \mid \boldsymbol{x}_k, m)$, where $\boldsymbol{z}_k$ are the measured distances to objects at time k, $\boldsymbol{x}_k$ is the state at time k and m is the map. The map is used together with the state to find what measurements the lidar should give $\boldsymbol{z}_{k|k-1}$, and the measurements are then assumed to be similar but with noise.

4.5.1 Ray tracing

In location-based maps like occupancy grids, $\boldsymbol{z}_{k|k-1}$ is found using ray tracing. Ray tracing is the act of projecting a single beam from the sensor and following or tracing that beam. It is done by moving through the map from $\boldsymbol{x}_k$ in the direction determined by the lidar. It can therefore be used to calculate $\boldsymbol{z}_{k|k-1}$ by tracing the beam until a collision is found in the map, finding an occupied cell in the case of an occupancy grid. Ray tracing is illustrated in Figure 4.4, where the measurements are found using $N_m = 20$ rays.

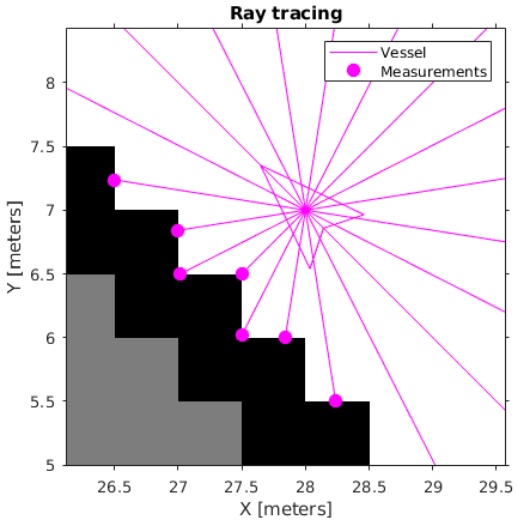


Figure 4.4: Illustration of ray tracing in an occupancy grid with resolution of 0.5 $\frac{\text{m}}{\text{px}}$.

4.5.2 Combining measurements

For a lidar with N_m measurements in a scan, and assuming each measurement is independent, the probability of a scan is the product of the likelihoods

$$p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k, m\right) = \prod_{n=1}^{N_m} p\left(z_k^n \mid \boldsymbol{x}_k, m\right).$$

Given a certain state, a map and the azimuth angle of a ray with index n, the measured range z_k^n is distributed according to $p\left(z_k^n \mid \boldsymbol{x}_k, m\right)$. There are many different models for range finders like lidars, some of which are mentioned below.

4.5.3 Beam model

Four possible measurement errors of the lidar are [60, pp. 153]:

- errors in measurement precision,
- errors due to unknown objects,
- failure to detect object,
- and unexplained noise or phantom readings.

The likelihood $p\left(z_k^n \mid \boldsymbol{x}_k, m\right)$ is a combination of different probability density functions corresponding to different types of errors. Two of these are explained below.

The range to an object is not measured perfectly and often has a local uncertainty around the expected range. This can be modelled using a Gaussian distribution [60, p. 155]

$$p_{\text{hit}}\left(z_k^n \mid \boldsymbol{x}_k, m\right) = \begin{cases} \alpha \mathcal{N}\left(z_k^n; z_{k|k-1}^n, \sigma_{\text{hit}}^2\right) & \text{if } 0 \leq z_k^n \leq z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

$$\mathcal{N}\left(z_k^n; z_{k|k-1}^n, \sigma_{\text{hit}}^2\right) = \frac{1}{\sqrt{2\pi\sigma_{\text{hit}}^2}} \mathrm{e}^{-\frac{\left(z_k^n - z_{k|k-1}^n\right)^2}{2\sigma_{\text{hit}}^2}}, \tag{4.6}$$

where α is a normalization constant and $z_{k|k-1}^n$ is the expected or true range from a assumed perfect map. α is often very close to 1, especially when $z_{k|k-1}$ is not close to the endpoints 0 and $z_{\max}$. This is because the integral of a probability density function should always be one, which the Gaussian distribution is. But if we cut of the Gaussian distribution and make it zero outside of an interval, the integral will not be one and a normalization constant is necessary.

Unexplained noise, phantom readings, unaccountable measurements or clutter can also occur and can be modelled as a uniform distribution $\mathcal{U}(0, z_{\max})$ [60, p. 157]

$$p_{\text{rand}}\left(z_k^n \mid \boldsymbol{x}_k, m\right) = \begin{cases} \frac{1}{z_{\max}} & \text{if } 0 \leq z_k^n \leq z_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

The different probability density functions can be combined to form the likelihood as

$$p\left(z_k^n \mid \boldsymbol{x}_k, m\right) = \begin{bmatrix} \alpha_{\text{hit}} & \alpha_{\text{rand}} \end{bmatrix} \begin{bmatrix} p_{\text{hit}}\left(z_k^n \mid \boldsymbol{x}_k, m\right) \\ p_{\text{rand}}\left(z_k^n \mid \boldsymbol{x}_k, m\right) \end{bmatrix},$$

where α_{hit} and α_{rand} are positive weights that sum to one. They can be found in different ways, including visual inspection and error minimization. The likelihood now contains the different errors used and can further be utilized in for example Bayesian filtering, see Section 5.4.

There are some negative aspects of the beam model. The main difficulty of the beam model is the ray tracing necessary to find the expected measurements [60, pp. 167]. This can be very computationally demanding, especially in already demanding state

estimation systems like the particle filter. Another difficulty is the lack of smoothness of $p\left(z_k^n \mid \boldsymbol{x}_k, m\right)$, making close states have very different expected measurements in cluttered environments. Some other models that do not have the same difficulties as the beam model are mentioned in the next section.

4.5.4 Other models

The likelihood field model overcomes the limitations of the beam model [60, ch. 6.4]. Different types of distributions are combined in the same way as earlier, but the key difference is that $p_{\text{hit}}\left(z_k^n \mid \boldsymbol{x}_k, m\right)$ is not found through ray tracing. It is now modelled with a Gaussian $d \sim \mathcal{N}\left(0, \sigma_{\text{hit}}\right)$, where d is the distance to the closest object from a measurement. That is, each measurement point is transformed into the map frame and the distance to the closest object is found. Since this model does not take the whole ray path into account, a problem is that it operates as if it can see through walls. The posteriors are smoother and the computation is much faster than the beam model. It works well, even though it is not generated based on a model of the physics of the sensor. The likelihood field can often be precomputed for all positions in a map which greatly reduces computation time. And the effect on the probability when using a grid for the precomputation is typically small.

Some other sensor models are correlation-based and feature-based models, but these are not explained in more detail here. See [60, pp. 174] for more information about them. The correlation between a measurement and the map can for example be found using an ICP method [8], if both the map and the measurements are represented by point clouds. The ICP method is used in this project, explained in Section 5.9 and used in Chapter 8, but as a method of state estimation rather than a measurement model.

4.6 Measurement model used

The measurements used in this project are made into 2D measurements from a 3D lidar as explained in the implementational aspects in Appendix C. Even though some data from the real vessel is gathered, there have been no experiments to calculate the optimal sensor model. One such experiment might be to place the ferry a set distance from a quay and register how many rays go into the water, over the quay and how many hit the actual quay. Then the model could have parameters defined from these experiments and fitted to the beam model described in Section 4.5.3. Since this is not the case, the model is created by using the map that was made from real data, see Appendix B. It is important to differentiate between two maps:

- Simulation map: The map created from real data in Appendix B. This map can

be used to create an approximate sensor model because it contains somewhat realistic noise. It can also be used to create partly realistic measurements. The map is illustrated in Figure 3.8.

- Estimation map: A simpler map where noise has been removed, mostly points that are known to be water. This is a map that should represent the quay and be the map the vessel tries to localize in and calculate expected measurements from. It is handmade to represent the area instead of the measurements. This map is shown in Figure 4.2.

Each map is read into Matlab and made into a "robotics.OccupancyGrid". This data type has already defined functions for ray tracing and transforming from indices to map coordinates. The measurements are therefore simulated by doing ray tracing in the map with noise, whereas the expected measurements used by the particle filter to evaluate particles are made from ray tracing in the simpler map. Figure 6.15 illustrates the difference, where there are measurements in the middle of the water.

There are two different kinds of noise used: Gaussian noise for measurement precision and uniform noise for clutter. There are also two different methods used for measurements, one for creating them and one for weighting them. The Gaussian and uniform errors were chosen because the measurements did not seem to be much affected by the other errors mentioned in Section 4.5.3. They are combined with weighting as mentioned in Section 4.5.3.

For evaluating measurements the model is

$$\boldsymbol{z}_k = \begin{cases} \boldsymbol{h}_r(\boldsymbol{x}_k) + \boldsymbol{n}_k & \text{with probability} \quad \alpha_{\text{hit}} \\ \boldsymbol{u}_k & \text{with probability} \quad \alpha_{\text{rand}} \end{cases}, \tag{4.8}$$

where α_{hit} and α_{rand} are parameters that will be estimated, $\boldsymbol{z}_k \in \mathbb{R}^{N_m}$ is a concatenation of N_m range measurements at time step k, $\boldsymbol{h}_r(\boldsymbol{x}_k)$ finds expected measurements through ray tracing from the state $\boldsymbol{x}_k$, each different ray i has $u_{k,i} \sim \mathcal{U}(0, z_{\max})$ uniformly distributed noise and $\boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_z)$,

$$\boldsymbol{\Sigma}_z = \begin{bmatrix} \sigma_{hit}^2 & 0 & \cdots & 0 \\ 0 & \sigma_{hit}^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{hit}^2 \end{bmatrix}.$$

Since the measurements are made synthetically, the Gaussian variance can be set to anything.

When creating measurements, the uniform noise is introduced by using the map with noise as mentioned earlier. The measurement model becomes

$$\boldsymbol{z}_k = \boldsymbol{h}_r(\boldsymbol{x}_k) + \boldsymbol{u}_k + \boldsymbol{n}_k, \tag{4.9}$$

where $\boldsymbol{h}_r(\boldsymbol{x}_k) + \boldsymbol{u}_k$ is approximated in one step through ray tracing of the map with realistic noise. This is for simplicity and to make the noise comes from the water and not actually be uniform.

4.6.1 Finding the parameters

The beam model is chosen because it is based on the physics of the sensor, see Section 4.5.3 for a description a the sensor model. For the ray tracing to work on the 2D map created earlier, the sensor is modelled as a 2D sensor, as described in Appendix C. So the angles used are azimuth angles and the measurements are distances to the collisions. The functions for $p_{\text{hit}}\left(z_k^n \mid \boldsymbol{x}_k, m\right)$ and $p_{\text{rand}}\left(z_k^n \mid \boldsymbol{x}_k, m\right)$ are implemented as shown in (4.5) and (4.7), respectively. The result is illustrated in Figure 4.5 with $\alpha_{\text{hit}} = 0.7$, $\alpha_{\text{rand}} = 0.3$ and $\sigma_{\text{hit}}^2 = 0.7$. One thing to notice is that the noise is actually not uniform, but there is mostly noise between the object and the vessel, where there is water. On the other hand, the fact that points can be behind the objects is not included in the simulations but is a problem in real life, as shown in Figure C.2. These are things that can be tuned using real data gathered.

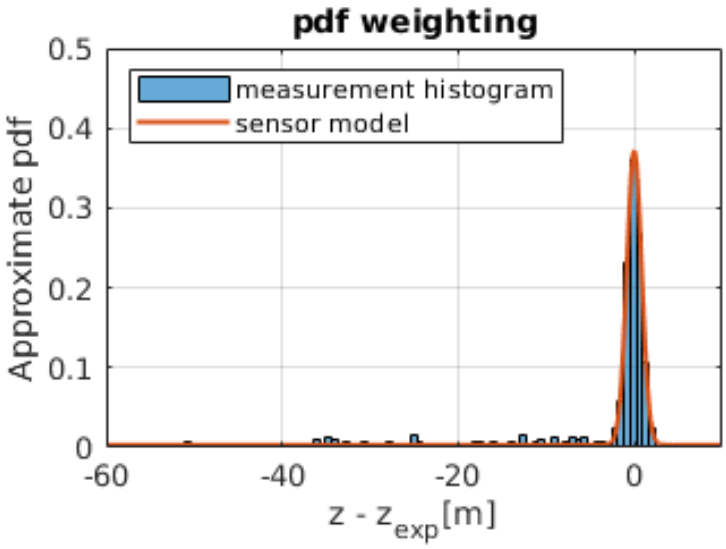


Figure 4.5: The combined probability density functions to fit measurements.

In the sensor model, the Gaussian uncertainty is due to range uncertainty for the sensor and the Gaussian noise from the actual sensor is very small, described in Section 3.4.1. To find a more realistic likelihood, the simulations add noise in x- and y-coordinates to simulate discretization of the map, which is a phenomenon not

otherwise included in the simulations:

$$\begin{bmatrix} z_{k,i,x} \\ z_{k,i,y} \end{bmatrix} = \boldsymbol{h}_i\left(\boldsymbol{x}_k\right) + \boldsymbol{n}_{k,i}$$
$$\boldsymbol{n}_{k,i} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\right)$$
$$z_{k,i} = \sqrt{z_{k,i,x}^2 + z_{k,i,y}^2}$$
$$\boldsymbol{z}_k = \begin{bmatrix} z_{k,1} & \cdots & z_{k,N_m} \end{bmatrix}^\mathsf{T},$$

where $\boldsymbol{h}_i\left(\boldsymbol{x}_k\right)$ gives the x- and y-coordinates of the collision point of ray i from the lidar in relative coordinates. N_m is the number of measurements from one sweep of the lidar and is simulated to be $N_m = 100$ due to computational limitations, which is a lot less than the lidar gives. But the number of measurements used will depend on the method chosen and is stated clearly for the relevant parts of the thesis. The rays are spread uniformly in the horizontal plane. It is assumed that it can work for more measurements in a case where it works for this number of measurements. The map has a resolution of 0.5 m and the variance is set to 0.5 in both directions, a standard deviation of approximately 0.7. The measurements are then made into azimuth angles and distances to collision to coincide with the sensor.

Chapter 5

State estimation methods

This chapter explains several methods of state estimation, including UKF, particle filter, Unscented Particle Filter (UPF) and ICP. It explains the particle filter as a Bayesian state estimation method that uses Monte Carlo simulations. The theory is gathered from a range of sources, but a good tutorial on particle filters is [1].

5.1 Relation to localization

As mentioned before, localization can be regarded as a state estimation problem, where the state is the pose of the robot. The robot is the object whose state will be estimated. The pose of a robot is its position and orientation with respect to a global reference frame, the map frame. One type of state estimation is Bayesian state estimation where an approximate probability distribution is propagated through Bayes rule in order to represent a probability distribution for the actual state. The PF localization, also known as Monte Carlo Localization, is a popular type of Bayesian state estimation where the state is the pose of the robot. It uses particles to represent the state probability distribution. How this works is described below.

5.2 The unscented Kalman filter

The Kalman filter is a well-known solution to the least squares method, see [69] for an introduction to the Kalman filter and the Extended Kalman Filter (EKF). The difference between the Kalman filter and the EKF is that where the Kalman filter can be used for state estimation of linear processes with linear measurements,

governed by linear differential equations, the EKF uses linearizations of nonlinear processes and measurement functions.

The Unscented Kalman Filter (UKF) was introduced by Julier and Uhlmann in [29]. It uses the typical Kalman filter structure of a prediction step and a correction step, without the linearization needed in the EKF. More precisely, it assumes a Gaussian distribution and uses sampled points to parameterize mean and covariance, a process called the Unscented Transform (UT). This yields performance similar to the Kalman filter for linear systems but can also be generalized to nonlinear systems [29].

The UKF extends the UT to the prediction and correction framework of the Kalman filter [67]

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}\left(\boldsymbol{z}_k - \hat{\boldsymbol{z}}_k^-\right),$$

where $\hat{\boldsymbol{x}}_k$ is the estimated state at time step k, $\hat{\boldsymbol{x}}_k^-$ is the predicted state based on time step $k-1$, $\boldsymbol{K}$ is the Kalman gain, $\boldsymbol{z}_k$ is the measurement at time step k and $\hat{\boldsymbol{z}}_k^-$ is the predicted measurement based on data at time step $k-1$.

5.2.1 The unscented transform

The Unscented Transform (UT) is a "method for calculating the statistics of a random variable which undergoes a nonlinear transformation" [29]. It finds the mean and covariance under the assumption of a Gaussian distribution, see an illustration of the UT in Figure 5.1. The random variable $\boldsymbol{x} \in \mathbb{R}^{n_x}$ is represented using a set of $2n_x+1$ weighted sigma points that have a sample mean $\bar{\boldsymbol{x}}$ and covariance $\boldsymbol{P}_{xx}$. The goal is to find the mean $\bar{\boldsymbol{z}}$ and covariance $\boldsymbol{P}_{zz}$ from the nonlinear relation

$$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}),$$

where $\boldsymbol{f}(\boldsymbol{x})$ is a nonlinear function.

The sigma points are chosen as [67]

$$\begin{aligned}
\boldsymbol{\mathcal{X}}_0 &= \bar{\boldsymbol{x}} \\
\boldsymbol{\mathcal{X}}_i &= \bar{\boldsymbol{x}} + \left(\sqrt{(n_x+\lambda)\boldsymbol{P}_{xx}}\right)_i, \quad i = 1, \ldots, n_x \\
\boldsymbol{\mathcal{X}}_{i+n_x} &= \bar{\boldsymbol{x}} - \left(\sqrt{(n_x+\lambda)\boldsymbol{P}_{xx}}\right)_i,
\end{aligned}$$

where $\left(\sqrt{(n_x+\lambda)\boldsymbol{P}_{xx}}\right)_i$ is the ith row of the matrix square root and

$$\lambda = \alpha^2(n_x+\kappa) - n_x \tag{5.1}$$

is a scaling parameter. α determines the spread of the sigma points, κ is a secondary scaling parameter which is often set to zero and β incorporates prior knowledge about

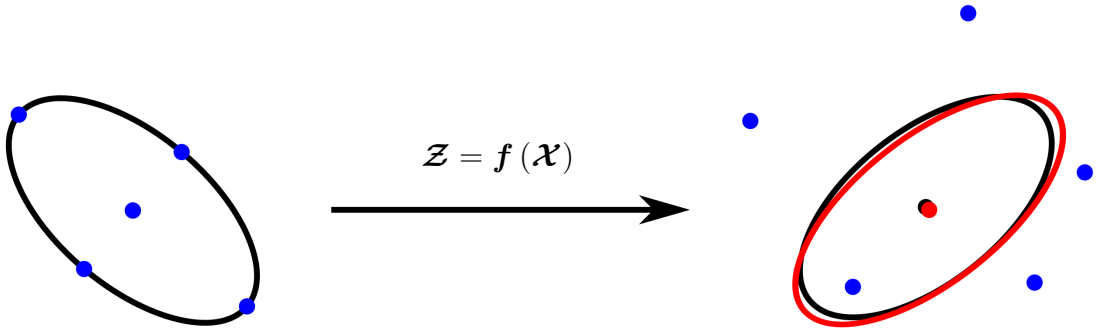


Figure 5.1: The principle of the UT is transforming the blue sigma points to get the propagated statistics shown in red. The red point is the UT mean, the red ellipse is the UT covariance. The figure is similar to Figure 2 in [29] and Figure 1 in [67].

the distribution of $\boldsymbol{x}$, optimally $\beta = 2$ for Gaussian distributions. The corresponding weights are

$$\begin{aligned} W_0^{(m)} &= \frac{\lambda}{n_x + \lambda} \\ W_0^{(c)} &= \frac{\lambda}{n_x + \lambda} + (1 - \alpha^2 + \beta) \\ W_i^{(m)} &= W_i^{(c)} = \frac{1}{2(n_x + \lambda)}, \quad i = 1, \ldots, 2n_x, \end{aligned}$$

where $W_i^{(m)}$ is used for finding the mean and $W_i^{(c)}$ is used to calculate the covariance. The sigma points are then propagated through the nonlinear function

$$\boldsymbol{\mathcal{Z}}_i = \boldsymbol{f}(\boldsymbol{\mathcal{X}}_i), \quad i = 0, \ldots, 2n_x$$

and the resulting mean and covariance are calculated by

$$\begin{aligned} \bar{\boldsymbol{z}} &= \sum_{i=0}^{2n_x} W_i^{(m)} \boldsymbol{\mathcal{Z}}_i \\ \boldsymbol{P}_{zz} &= \sum_{i=0}^{2n_x} W_i^{(c)} (\boldsymbol{\mathcal{Z}}_i - \bar{\boldsymbol{z}}) (\boldsymbol{\mathcal{Z}}_i - \bar{\boldsymbol{z}})^\mathsf{T}. \end{aligned}$$

5.2.2 The filter

The UKF augments the state to be $\boldsymbol{x}_k^a = \left[\boldsymbol{x}_k^\mathsf{T}, \boldsymbol{v}_k^\mathsf{T}, \boldsymbol{n}_k^\mathsf{T}\right]^\mathsf{T}$ where $\boldsymbol{v}_k$ and $\boldsymbol{n}_k$ is noise according to (4.1). This results in $n_x^a = n_x + n_v + n_n$ dimensions of the state, where $\boldsymbol{x}_k \in \mathbb{R}^{n_x}$, $\boldsymbol{v}_k \in \mathbb{R}^{n_v}$ and $\boldsymbol{n}_k \in \mathbb{R}^{n_n}$. The resulting number of sigma points is then

$$2n_x^a + 1 = 2(n_x + n_v + n_n) + 1. \tag{5.2}$$

The sigma points are written as $\boldsymbol{\mathcal{X}}^a = \left[(\boldsymbol{\mathcal{X}}^x)^\mathsf{T}, (\boldsymbol{\mathcal{X}}^v)^\mathsf{T}, (\boldsymbol{\mathcal{X}}^n)^\mathsf{T}\right]^\mathsf{T}$. The resulting augmented covariance is

$$\boldsymbol{P}_k^a = \begin{bmatrix} \boldsymbol{P}_k & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{P}_v & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{P}_n \end{bmatrix},$$

where the covariances of the process noise and the measurement noise, $\boldsymbol{P}_v$ and $\boldsymbol{P}_n$ respectively, are known.

The UKF is initialized with state estimate $\hat{\boldsymbol{x}}_0$ and covariance estimate $\boldsymbol{P}_0$. Then the UKF is used iteratively. One iteration of the filter is shown in Algorithm 1. It is worth noting that the equations used in that algorithm does not require linearization of the models used, an explanation of the model with process function $\boldsymbol{f}$ and measurement function $\boldsymbol{h}$ is given in Section 4.1.

Algorithm 1 An iteration of the UKF. Based on [67] and [30]. The algorithm uses the notation in Section 5.2.2.

function UKF($\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{P}_{k-1}, \boldsymbol{z}_k$)
  Augment system:
    $\hat{\boldsymbol{x}}_{k-1}^a = \left[\hat{\boldsymbol{x}}_{k-1}^\mathsf{T}, \mathbf{0}^\mathsf{T}, \mathbf{0}^\mathsf{T}\right]^\mathsf{T}$
    $\boldsymbol{P}_{k-1}^a = \begin{bmatrix} \boldsymbol{P}_{k-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{P}_v & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{P}_n \end{bmatrix}$

  Calculate sigma points $\boldsymbol{\mathcal{X}}_{k-1}^a$ from $\hat{\boldsymbol{x}}_{k-1}^a$ and $\boldsymbol{P}_{k-1}^a$ as in Section 5.2.1.

  Time update:
    $\boldsymbol{\mathcal{X}}_k^{x-} = \boldsymbol{f}\left(\boldsymbol{\mathcal{X}}_{k-1}^x, \boldsymbol{\mathcal{X}}_{k-1}^v\right)$
    $\hat{\boldsymbol{x}}_k^- = \sum_{i=0}^{2n_x^a} W_i^{(m)} \boldsymbol{\mathcal{X}}_{i,k}^{x-}$
    $\boldsymbol{P}_k^- = \sum_{i=0}^{2n_x^a} W_i^{(c)} \left(\boldsymbol{\mathcal{X}}_{i,k}^{x-} - \hat{\boldsymbol{x}}_k^-\right)\left(\boldsymbol{\mathcal{X}}_{i,k}^{x-} - \hat{\boldsymbol{x}}_k^-\right)^\mathsf{T}$

  Measurement update:
    $\boldsymbol{\mathcal{Z}}_k^- = \boldsymbol{h}\left(\boldsymbol{\mathcal{X}}_{i,k}^{x-}, \boldsymbol{\mathcal{X}}_{k-1}^n\right)$
    $\hat{\boldsymbol{z}}_k^- = \sum_{i=0}^{2n_x^a} W_i^{(m)} \boldsymbol{\mathcal{Z}}_{i,k}^-$
    $\boldsymbol{P}_{\hat{\boldsymbol{z}}_k,\hat{\boldsymbol{z}}_k} = \sum_{i=0}^{2n_x^a} W_i^{(c)} \left(\boldsymbol{\mathcal{Z}}_{i,k}^- - \hat{\boldsymbol{z}}_k^-\right)\left(\boldsymbol{\mathcal{Z}}_{i,k}^- - \hat{\boldsymbol{z}}_k^-\right)^\mathsf{T}$
    $\boldsymbol{P}_{\hat{\boldsymbol{x}}_k,\hat{\boldsymbol{z}}_k} = \sum_{i=0}^{2n_x^a} W_i^{(c)} \left(\boldsymbol{\mathcal{X}}_{i,k}^{x-} - \hat{\boldsymbol{x}}_k^-\right)\left(\boldsymbol{\mathcal{Z}}_{i,k}^- - \hat{\boldsymbol{z}}_k^-\right)^\mathsf{T}$
    $\boldsymbol{K} = \boldsymbol{P}_{\hat{\boldsymbol{x}}_k,\hat{\boldsymbol{z}}_k} \boldsymbol{P}_{\hat{\boldsymbol{z}}_k,\hat{\boldsymbol{z}}_k}^{-1}$
    $\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}\left(\boldsymbol{z}_k - \hat{\boldsymbol{z}}_k^-\right)$
    $\boldsymbol{P}_k = \boldsymbol{P}_k^- - \boldsymbol{K}\boldsymbol{P}_{\hat{\boldsymbol{z}}_k,\hat{\boldsymbol{z}}_k}\boldsymbol{K}^\mathsf{T}$

  return $\hat{\boldsymbol{x}}_k, \boldsymbol{P}_k$
end function

5.3 Monte Carlo simulation

Monte Carlo simulation is the act of simulating a process many times with random sampling to estimate some statistic. This is a very general method and is useful for many applications. The particle filter is an example of how Monte Carlo simulations can be used for state estimation.

Monte Carlo simulation can be used for estimating the probabilities of different outcomes of a system from the use of realizations of random input to the system. This can for example be useful in the estimation of a probability density function $p(x)$ that is difficult to sample from. In this case assume that it is possible to sample from an other probability distribution $q(x)$. Let $x^{\{i\}}$, $i = 1, \ldots, N$ be samples from $q(x)$ and let

$$w^{\{i\}} = \frac{p(x^{\{i\}})}{q(x^{\{i\}})}.$$

Moreover, let $\bar{w}^{\{i\}}$ denote the normalized weight

$$\bar{w}^{\{i\}} = \frac{w^{\{i\}}}{\sum_{i=1}^{i=N} w^{\{i\}}}.$$

and let x^* be a sample from the discrete distribution over $\{x^{\{1\}}, \ldots, x^{\{N\}}\}$ with mass $\bar{w}^{\{i\}}$ on $x^{\{i\}}$. Then x^* is approximately distributed according to $q(x)$, with better approximation as N increases [58]. This process of sampling is called Importance Sampling (IS) or Normalized Importance Sampling (NIS) [16]. Monte Carlo simulation can in this way be used to approximate sampling from $p(x)$ by the sampling described above.

5.4 Bayes Filter

Bayesian state estimation is concerned with calculating the probability density function $p\left(\boldsymbol{x}_{0:k} \mid \boldsymbol{z}_{1:k}\right)$, where $\boldsymbol{x}_{0:k} \triangleq \{\boldsymbol{x}_i \mid i \in 0, \ldots, k\}$ is the hidden sequence of discrete time states of a robot and $\boldsymbol{z}_{1:k} \triangleq \{\boldsymbol{z}_i \mid i \in 1, \ldots, k\}$ are the observations at the different time steps. Using Bayes rule the distribution can be written as

$$p\left(\boldsymbol{x}_{0:k} \mid \boldsymbol{z}_{1:k}\right) = \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}, \boldsymbol{x}_{0:k}\right) p\left(\boldsymbol{x}_{0:k} \mid \boldsymbol{z}_{1:k-1}\right)}{p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}\right)}.$$

This can furthermore be written on a recursive form by factorizing $p\left(\boldsymbol{x}_{0:k} \mid \boldsymbol{z}_{1:k-1}\right)$:

$$p\left(\boldsymbol{x}_{0:k} \mid \boldsymbol{z}_{1:k}\right) = \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}, \boldsymbol{x}_{0:k}\right) p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}, \boldsymbol{x}_{0:k-1}\right)}{p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}\right)} p\left(\boldsymbol{x}_{0:k-1} \mid \boldsymbol{z}_{1:k-1}\right). \quad (5.3)$$

These equations are very general and can be made easier to use by the Markov assumption discussed next.

A Markov model is a model that satisfies the Markov property or equivalently assumes the Markov assumption: that the probability of being in a state $\boldsymbol{x}_k$ only depends on the latest state $\boldsymbol{x}_{k-1}$ and not earlier states [51]. Furthermore, if the states are not directly observable, the model is called a Hidden Markov Model (HMM) [51]. It assumes the observation $\boldsymbol{z}_k$ is a probabilistic functions of the state $\boldsymbol{x}_k$. The resulting model is shown in Figure 5.2, illustrating the dependency between states and measurements via arrows. Using a HMM, the probability distribution $p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}, \boldsymbol{x}_{0:k}\right)$ simplifies to $p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k\right)$ and $p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}, \boldsymbol{x}_{0:k-1}\right)$ simplifies to $p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}\right)$. These two probability density function are just different notation for the same state estimation models mentioned earlier, the state space form in (4.1).

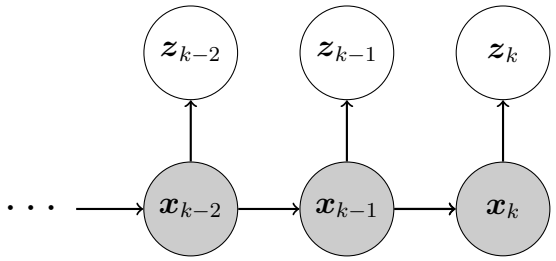


Figure 5.2: A HMM for the state estimation problem.

The Bayes filter uses a HMM for estimating the state probability distribution. Using (5.3) in combination with a HMM the state distribution takes the form

$$p\left(\boldsymbol{x}_{0:k} \mid \boldsymbol{z}_{1:k}\right) = \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k\right) p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}\right)}{p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}\right)} p\left(\boldsymbol{x}_{0:k-1} \mid \boldsymbol{z}_{1:k-1}\right). \tag{5.4}$$

In practise, it is often desirable only to estimate the current state $\boldsymbol{x}_k$ and not the full history of states. This is can be done by marginalizing over all possible earlier states $\boldsymbol{x}_{k-1}$:

$$\begin{aligned} p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}\right) &= \int p\left(\boldsymbol{x}_{k-1:k} \mid \boldsymbol{z}_{1:k}\right) \mathrm{d}\boldsymbol{x}_{k-1} \\ &= \int \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k\right) p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}\right)}{p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}\right)} p\left(\boldsymbol{x}_{k-1} \mid \boldsymbol{z}_{1:k-1}\right) \mathrm{d}\boldsymbol{x}_{k-1}. \end{aligned}$$

This can be factorized into the prediction step, called the Chapman-Kolmogorov equation [1]

$$p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}\right) = \int p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}\right) p\left(\boldsymbol{x}_{k-1} \mid \boldsymbol{z}_{1:k-1}\right) \mathrm{d}\boldsymbol{x}_{k-1} \tag{5.5}$$

and the filtering step

$$p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}\right) = \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k\right)}{p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}\right)} p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}\right) = \eta p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k\right) p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}\right), \tag{5.6}$$

where $\eta = \left(p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}\right)\right)^{-1}$ is a normalization constant.

The Bayesian filter can be used in many applications, not only robotics, but the filter seen in Algorithm 2 is the most general recursive estimation algorithm for estimating the probability distribution over the state of the robot [60, pp. 26]. It uses (5.5) and (5.6) with a measurement model for $p(\boldsymbol{z}_k \mid \boldsymbol{x}_k)$ and a motion model for $p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$ in order to use the new measurement $\boldsymbol{z}_k$ to update the state estimate. The normalization constant η is the same for all states $\boldsymbol{x}_k$ and does not need to be calculated explicitly since it only makes the distribution a proper probability distribution with integral equal to one.

Algorithm 2 The Bayes filter as an algorithm.

function BAYESFILTER($p(\boldsymbol{x}_{k-1} \mid \boldsymbol{z}_{1:k-1}), \boldsymbol{z}_k$)
  for all x_k **do**
    $p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}) = \int p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})\, p(\boldsymbol{x}_{k-1} \mid \boldsymbol{z}_{1:k-1})\, \mathrm{d}\boldsymbol{x}_{k-1}$
    $p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}) = \eta p(\boldsymbol{z}_k \mid \boldsymbol{x}_k)\, p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1})$
  end for
  return $p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k})$
end function

5.5 Particle filter

The Particle Filter (PF) is an implementation of the Bayes filter. There are many different versions of the particle filter. Some making it more robust, faster and giving better convergence in certain applications. It is a filtering method, and similarly to the Kalman filter, it can be used for state estimation. It works by taking measurements, doing prediction and doing Bayesian probability propagation to estimate a probability density function for the state. It represents the probability density function $p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k})$ using so-called particles that are samples drawn from the distribution. This makes it able to represent many different probability distributions, including multi-modal distributions contrary to the Kalman filter. Some relevant literature on the subject is [60, pp. 96] and [1, 14]. It can be simple to implement a basic particle filter, yet difficult to make it work well. The particle filter has several difficulties regarding sample size and might be computationally expensive [15].

5.5.1 Sequential Importance Sampling

The Sequential Importance Sampling (SIS) method is in a sense a simplification of the generic particle filter. It is ambiguous from the literature if it uses re-sampling or not, but here it is shown without re-sampling following [1]. The SIS method implements the recursive Bayes filter with Monte Carlo simulations [1]. The posterior probability distribution is represented by a set of random samples $\boldsymbol{x}^{\{i\}}$ and

associated weights $w^{\{i\}}$ for the particles with index $i = 1, \ldots, N$, as described in Section 5.3 about Monte Carlo simulation. The Bayes filter, described in Section 5.4, is then used to update the approximate probability distribution using models of the sensor and the process dynamics. The probability distribution of the state given the measurements is

$$p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}\right) \approx \sum_{i=1}^{N} \bar{w}_k^{\{i\}} \delta\left(\boldsymbol{x}_k - \boldsymbol{x}_k^{\{i\}}\right), \tag{5.7}$$

where $\bar{w}_k^{\{i\}}$ is the normalized weight of particle i at time step k and $\delta(\cdot)$ is the Dirac delta.

The resulting SIS filter is shown in Algorithm 3. To be able to use Monte Carlo simulation, the importance density $q\left(\boldsymbol{x}\right)$ needs to be specified and the resulting re-weighting strategy calculated. That is done in Section 5.5.2.

Algorithm 3 The SIS algorithm.

function SIS_FILTER($\{\boldsymbol{x}_k^{\{i_p\}}, w_k^{\{i_p\}}\}_{i_p=1}^N, \boldsymbol{z}_k$)
  for $i_p = 1, \ldots, N$ **do** ▷ Sample particles
    $\boldsymbol{x}_k^{\{i_p\}} \sim q\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i_p\}}, \boldsymbol{z}_k\right)$
  end for
  for $i_p = 1, \ldots, N$ **do** ▷ Re-weight
    Assign weight $w_k^{\{i_p\}}$ according to (5.8)
  end for
  return $\{\boldsymbol{x}_k^{\{i_p\}}, w_k^{\{i_p\}}\}_{i_p=1}^N$
end function

5.5.2 Weighting and importance density

The particle filter uses Monte Carlo simulations to approximate sampling from the actual state distribution $p\left(\boldsymbol{x}\right)$, which can be difficult to sample from. The state particles $\left\{\boldsymbol{x}^{\{i\}}\right\}_{i=1}^N$ are assumed easily generated from the importance density $q\left(\boldsymbol{x}\right)$, see Section 5.3 for an explanation of Monte Carlo simulations. The weighting used depends on likelihoods of different measurements and states, which might for example require a sensor model and process model. Some examples of sensor models are described in Section 4.5 and an example of process model is described in Section 4.2. The weights follow the relation

$$\bar{w}^{\{i\}} \propto \frac{p(\boldsymbol{x}^{\{i\}})}{q(\boldsymbol{x}^{\{i\}})},$$

where $\bar{w}^{\{i\}}$ is the normalized weight of particle i. The approximation of the state probability distribution then becomes

$$p\left(\boldsymbol{x}\right) \approx \sum_{i=1}^{N} \bar{w}^{\{i\}} \delta\left(\boldsymbol{x} - \boldsymbol{x}^{\{i\}}\right),$$

where $\delta(\cdot)$ is the Dirac delta.

From the Bayes filter calculation (5.4) using that $p\left(\boldsymbol{z}_k \mid \boldsymbol{z}_{1:k-1}\right)$ is a normalizing constant for a certain time step, the probability distribution of the states satisfies the relation

$$p\left(\boldsymbol{x}_{0:k} \mid \boldsymbol{z}_{1:k}\right) \propto p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k\right) p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}\right) p\left(\boldsymbol{x}_{0:k-1} \mid \boldsymbol{z}_{1:k-1}\right).$$

Assuming the importance density can be factorized into

$$q\left(\boldsymbol{x}_{0:k} \mid \boldsymbol{z}_{1:k}\right) = q\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{0:k-1}, \boldsymbol{z}_{1:k}\right) q\left(\boldsymbol{x}_{0:k-1} \mid \boldsymbol{z}_{1:k-1}\right),$$

the weights become

$$\begin{aligned}
\bar{w}_k^{\{i\}} &\propto \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k^{\{i\}}\right) p\left(\boldsymbol{x}_k^{\{i\}} \mid \boldsymbol{x}_{k-1}^{\{i\}}\right) p\left(\boldsymbol{x}_{0:k-1}^{\{i\}} \mid \boldsymbol{z}_{1:k-1}\right)}{q\left(\boldsymbol{x}_k^{\{i\}} \mid \boldsymbol{x}_{0:k-1}^{\{i\}}, \boldsymbol{z}_{1:k}\right) q\left(\boldsymbol{x}_{0:k-1}^{\{i\}} \mid \boldsymbol{z}_{1:k-1}\right)}, \\
&\propto \bar{w}_{k-1}^{\{i\}} \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k^{\{i\}}\right) p\left(\boldsymbol{x}_k^{\{i\}} \mid \boldsymbol{x}_{k-1}^{\{i\}}\right)}{q\left(\boldsymbol{x}_k^{\{i\}} \mid \boldsymbol{x}_{0:k-1}^{\{i\}}, \boldsymbol{z}_{1:k}\right)}
\end{aligned}$$

since $\bar{w}_{k-1}^{\{i\}} \propto \frac{p\left(\boldsymbol{x}_{0:k-1}^{\{i\}} \mid \boldsymbol{z}_{1:k-1}\right)}{q\left(\boldsymbol{x}_{0:k-1}^{\{i\}} \mid \boldsymbol{z}_{1:k-1}\right)}$. Lets assume that the density is independent on earlier measurements and earlier states except the last one, that is the Markov property, $q\left(\boldsymbol{x}_k^{\{i\}} \mid \boldsymbol{x}_{0:k-1}^{\{i\}}, \boldsymbol{z}_{1:k}\right) = q\left(\boldsymbol{x}_k^{\{i\}} \mid \boldsymbol{x}_{k-1}^{\{i\}}, \boldsymbol{z}_k\right)$. This gives the weight update equation

$$\bar{w}_k^{\{i\}} \propto \bar{w}_{k-1}^{\{i\}} \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k^{\{i\}}\right) p\left(\boldsymbol{x}_k^{\{i\}} \mid \boldsymbol{x}_{k-1}^{\{i\}}\right)}{q\left(\boldsymbol{x}_k^{\{i\}} \mid \boldsymbol{x}_{k-1}^{\{i\}}, \boldsymbol{z}_k\right)}, \tag{5.8}$$

for the posterior approximation shown in (5.7).

Optimal importance density

There are many different choices of importance densities. The optimal importance density that minimizes the variance of the true weights conditioned on the earlier

state $\boldsymbol{x}_{k-1}^{\{i\}}$ and the new measurement $\boldsymbol{z}_k$ is [16]

$$q\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i\}}, \boldsymbol{z}_k\right)_{\text{opt}} = p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i\}}, \boldsymbol{z}_k\right) = \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k\right) p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i\}}\right)}{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_{k-1}^{\{i\}}\right)}$$

Substituting this into (5.8) gives the weight update equation

$$\bar{w}_k^{\{i\}} \propto \bar{w}_{k-1}^{\{i\}} p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_{k-1}^{\{i\}}\right).$$

It might be difficult to find or use the optimal importance density. There are two cases where it is possible [1]

- $\boldsymbol{x}_k$ is a member of a finite set.
- $p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i\}}, \boldsymbol{z}_k\right)$ is Gaussian. This can for example happen if the measurements are linearly dependant on the state.

Prior as importance density

A common approximation to the optimal importance density is to use the prior [1]

$$q(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i\}}, \boldsymbol{z}_k) = p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i\}})$$

which gives the weights

$$\bar{w}_k^{\{i\}} \propto \bar{w}_{k-1}^{\{i\}} p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k^{\{i\}}\right). \tag{5.9}$$

This gives simple expressions that are quick to evaluate.

5.5.3 The problem of degeneracy

A typical problem of the SIS filter is the problem of degeneracy. This is where, after some iterations, all particles but one have negligible weight. It means that a lot of computational effort is wasted on less likely particles and also that the particles might not be covering the most optimal area in order to get the best estimates. The brute force solution is to simply increase the amount of particles N, but this comes with severe increase of computational effort. Two other solutions to the degeneracy problem are

- Better choice of importance density.

- Re-sampling: Sample the particles from the distribution in order to only have particles in the most probable states. This is discussed more in Section 5.5.4.

A measure of the degeneracy of the particles is the effective sample size [1]

$$N_{\text{eff}} = \frac{N}{1 + \text{Var}\left(w_k^{\{i_p\}*}\right)},$$

where $w_k^{\{i_p\}*} = \frac{p\left(\boldsymbol{x}_k^{\{i_p\}}|\boldsymbol{z}_{1:k}\right)}{q\left(\boldsymbol{x}_k^{\{i_p\}}|\boldsymbol{z}_{1:k},\boldsymbol{x}_{k-1}^{\{i_p\}}\right)}$ is called the true weight. This can be approximated by

$$\hat{N_{\text{eff}}} = \frac{1}{\sum_{i_p=1}^{N}\left(\bar{w}_k^{\{i_p\}}\right)^2}, \tag{5.10}$$

where $\bar{w}_k^{\{i_p\}}$ is the normalized weight from (5.8).

5.5.4 Re-sampling

To avoid the degeneracy problem, re-sampling can be done. By sampling particles from the existing particles, particles with negligible weights will be lost and only the most probable particles will remain. Some methods of re-sampling are described below.

Sampling a discrete distribution

The weight of each particle in a particle filter represents the likelihood of that particle, so that the particles together represent a discrete probability distribution as formulated in (5.7). It is possible to sample such a discrete probability distribution, as described in [14], by first sampling u from the standard uniform distribution $U \sim \mathcal{U}(0,1)$. Then the particle p is found by picking the particle that satisfies

$$\sum_{i=1}^{i_p-1} w^{\{i\}} < u \quad \text{and} \quad \sum_{i=1}^{i_p} w^{\{i\}} \geq u, \tag{5.11}$$

where i_p is the index of the particle and $w^{(i)}$ is the normalized weight of a particle. Assuming all particles have a probability above zero, this will be a single particle. Since the weights sum up to one, this will make particles with big weights more likely to be picked. If this is done for every particle in the particle filter, the result will be a subset of particles representing the discrete probability distribution. The more likely states have more particles representing it, and the less likely states have fewer or even none. Weighting these new particles equally should make an integral over the actual distribution of possible states approximately equal to the equivalent sum over the states of the particle filter.

Low variance sampling

The variance of a sampler is the variability due to random sampling [60, p. 108]. Drawing every sample based on independent random number might give higher sampling variance. A sampling method to mitigate this is the low variance sampling or systematic sampling, given in Algorithm 4. Assume the weights are normalized and there are N evenly spaced probabilities u. The method finds for each probability the particle that makes the cumulative weight W go above it and thereby solving (5.11). More precisely, it draws a sample from the uniform distribution $U \sim \mathcal{U}(0, N^{-1})$. Then it takes N probabilities spaced evenly with a step of N^{-1} and finds for which particle the cumulative distribution passes that probability. The first probability is at the random number chosen. That is, it takes the sample with index i_p that satisfies (5.11), where the u's are described as above instead of each being independent. This gives a lower variance sampling in $\mathcal{O}(N)$ time complexity.

Algorithm 4 Sampling algorithm with low variance from [60, pp. 110] and [1]. Assumes normalized weights $\bar{w}^{\{i_p\}}$.

function RE-SAMPLE($\{\boldsymbol{x}_k^{\{i_p\}}, \bar{w}_k^{\{i_p\}}\}_{i_p=1}^N$)
  $r = \text{uniform}(0, N^{-1})$
  $W = \bar{w}^{\{1\}}$
  $i_p = 1$
  for $n = 1, \ldots, N$ **do**
    $u = r + (n-1)N^{-1}$
    while $W < u$ **do** ▷ Solve (5.11).
      $i_p = i_p + 1$
      $W = W + \bar{w}^{\{i_p\}}$
    end while
    $\boldsymbol{x}_k^{\{n\}*} = \boldsymbol{x}_k^{\{i_p\}}$
    $\bar{w}_k^{\{n\}*} = N^{-1}$
  end for
  return $\{\boldsymbol{x}_k^{\{n\}*}, \bar{w}_k^{\{n\}*}\}_{n=1}^N$
end function

Regularized particle filter

Particle impoverishment is when the particles lose diversity [1]. This can happen since the default re-sampling gives a subset of the particles, and there can be many repeated particles. It might eventually lead to particle collapse, where all particles represent equal states. It can especially be a problem in the case of low process noise, with a narrow true probability distribution. One attempt of solving this is instead of re-sampling at every time step as in the Sampling Importance Re-sampling (SIR) filter, only re-sample when the empirical particle variance is under a certain threshold.

Another possible attempt of solving the problem of particle impoverishment is using the regularized particle filter [1, 43]. It is very similar to the generic particle filter, but has a step of re-sampling that diversifies the particles width the use of the empirical covariance. The samples are then from the approximation

$$p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}\right) \approx \sum_{i_p=1}^{N} \hat{w}_k^{\{i_p\}} K_h\left(\boldsymbol{x}_k - \boldsymbol{x}_k^{\{i_p\}}\right),$$
$$K_h\left(\boldsymbol{x}\right) = \frac{1}{h^{n_x}} K\left(\frac{\boldsymbol{x}}{h}\right),$$

where $K(\cdot)$ is the kernel density, h is the scalar bandwidth and n_x is the dimension of the state vector $\boldsymbol{x}$. The kernel density is in general a symmetric probability density function used to spread the particles and must satisfy certain properties [43]. The kernel $K(\cdot)$ and bandwidth h are chosen to minimize the mean integrated square error between the true posterior and the regularized approximation [1]:

$$\mathrm{MISE}\left(\hat{p}\right) = \mathbb{E}\left[\int \left(\hat{p}\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}\right) - p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}\right)\right)^2 \mathrm{d}\boldsymbol{x}_k\right],$$

where $\hat{p}\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}\right)$ is the approximation of $p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}\right)$. In the special case of equal weights $\hat{w}^{\{i_p\}} = \frac{1}{N}$, the optimal kernel that minimizes $\mathrm{MISE}\left(\hat{p}\right)$ becomes the Epanechnikov kernel [43]

$$K_{\mathrm{opt}}\left(\boldsymbol{x}\right) = \begin{cases} \frac{n_x+2}{2c_{n_x}}\left(1 - \left\|\boldsymbol{x}\right\|^2\right) & \text{if } \left\|\boldsymbol{x}\right\| < 1 \\ 0 & \text{otherwise,} \end{cases}$$

where c_{n_x} is the volume of the unit sphere in $\mathbb{R}^{n_x}$. Assuming the density is Gaussian with unit covariance matrix, the optimal bandwidth becomes

$$h_{\mathrm{opt}} = AN^{-\frac{1}{n_x+4}},$$
$$A = \left(\frac{8}{c_{n_x}}\left(n_x+4\right)\left(2\sqrt{\pi}\right)^{n_x}\right)^{\frac{1}{n_x+4}}.$$

To make sampling the density simpler, the Epanechnikov kernel can be replaced by the Gaussian kernel [43]. This assumes the same as earlier, but additionally assumes covariance equal to the empirical covariance and applies a linear transformation of $\boldsymbol{x}^{\{i_p\}}$ into $A^{-1}\boldsymbol{x}^{\{i_p\}}$. The optimal bandwidth now becomes

$$h_{\mathrm{opt}} = AN^{-\frac{1}{n_x+4}},$$
$$A = \left(\frac{4}{n_x+2}\right)^{\frac{1}{n_x+4}}.$$

The assumptions for the Epanechnikov and Gaussian kernels to be optimal are not always valid. Nevertheless, the regularized particle filter performs well in practical cases with big problems of sample impoverishment [1].

A regularized particle filter is implemented similarly as the generic particle filter, but with the regularized re-sampling as shown in Algorithm 5. It uses the re-sampling from Algorithm 4, but also scatters the particles depending on the bandwidth and the empirical covariance.

Algorithm 5 The re-sampling step of a regularized particle filter.

function REGULARIZED_RE-SAMPLING($\{\boldsymbol{x}_k^{\{i_p\}}, w_k^{\{i_p\}}\}_{i_p=1}^N$)
  Calculate empirical covariance matrix $\boldsymbol{S}_k$ from $\{\boldsymbol{x}_k^{\{i_p\}}, w_k^{\{i_p\}}\}_{i_p=1}^N$ ▷ using (5.13)
  Calculate $\boldsymbol{D}_k$ such that $\boldsymbol{D}_k \boldsymbol{D}_k^\mathsf{T} = \boldsymbol{S}_k$
  $\{\boldsymbol{x}_k^{\{i_p\}}, w_k^{\{i_p\}}\}_{i_p=1}^N = \text{RE-SAMPLE}(\{\boldsymbol{x}_k^{\{i_p\}}, w_k^{\{i_p\}}\}_{i_p=1}^N)$ ▷ Using Algorithm 4
  for $i_p = 1, \ldots, N$ **do**
    $\boldsymbol{\epsilon}^{\{i_p\}} \sim K$ ▷ Sample the kernel
    $\boldsymbol{x}_k^{\{i_p\}} = \boldsymbol{x}_k^{\{i_p\}} + h_{\text{opt}} \boldsymbol{D}_k \boldsymbol{\epsilon}^{\{i_p\}}$
  end for
  return $\{\boldsymbol{x}_k^{\{i_p\}}, w_k^{\{i_p\}}\}_{i_p=1}^N$
end function

5.5.5 The generic particle filter

Building on the SIS filter, a generic particle filter using re-sampling when the effective sample size is small is shown in Algorithm 6. Here the re-sampling in Algorithm 4 is used. In general, any re-sampling method and importance density function might be used.

5.5.6 Sampling Importance Re-sampling

The acronym SIR is ambiguous in the literature, where it can be short for Sequential Importance Re-sampling, but the notation used here follows [1].

The Sampling Importance Re-sampling (SIR) filter is an implementation of the generic particle filter shown in Algorithm 6. It assumes known state dynamics and measurement function and that it is possible to sample from the prior $p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i_p\}}\right)$. An example of the dynamics of a system can be seen in Section 4.2. The likelihood $p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k\right)$ must be possible to evaluate up to proportionality. Then choosing the importance density as the prior, as in (5.9) and applying the re-sampling at every iteration gives the SIR filter shown in Algorithm 7.

Algorithm 6 The generic particle filter algorithm. N_{T} is a threshold for the effective sample size. Here the re-sampling is done using Algorithm 4, but it can be any re-sampling method.

function PARTICLE_FILTER($\{\boldsymbol{x}_{k-1}^{\{i_p\}}, w_{k-1}^{\{i_p\}}\}_{i_p=1}^{N}, \boldsymbol{z}_k$)
  for $i_p = 1, \ldots, N$ **do** ▷ Sample particles
    $\boldsymbol{x}_k^{\{i_p\}} \sim q\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i_p\}}, \boldsymbol{z}_k\right)$
  end for
  for $i_p = 1, \ldots, N$ **do** ▷ Re-weight
    $w_k^{\{i_p\}} = \bar{w}_{k-1}^{\{i_p\}} \frac{p\left(\boldsymbol{z}_k | \boldsymbol{x}_k^{\{i_p\}}\right) p\left(\boldsymbol{x}_k^{\{i_p\}} | \boldsymbol{x}_{k-1}^{\{i_p\}}\right)}{q\left(\boldsymbol{x}_k^{\{i_p\}} | \boldsymbol{x}_{k-1}^{\{i_p\}}, \boldsymbol{z}_k\right)}$
  end for
  for $i_p = 1, \ldots, N$ **do** ▷ Normalize weights
    $\bar{w}_k^{\{i_p\}} = w_k^{\{i_p\}} \left(\sum_{i=1}^{N} w_k^{\{i\}}\right)^{-1}$
  end for
  Calculate $\hat{N}_{\mathrm{eff}}$ from (5.10)
  if $\hat{N}_{\mathrm{eff}} < N_{\mathrm{T}}$ **then** ▷ Small effective sample size
    $\{\boldsymbol{x}_k^{\{i_p\}}, \bar{w}_k^{\{i_p\}}\}_{i_p=1}^{N} = \text{RE-SAMPLE}(\{\boldsymbol{x}_k^{\{i_p\}}, \bar{w}_k^{\{i_p\}}\}_{i_p=1}^{N})$
  end if
  return $\{\boldsymbol{x}_k^{\{i_p\}}, \bar{w}_k^{\{i_p\}}\}_{i_p=1}^{N}$
end function

Algorithm 7 The SIR filter.

function SIR_FILTER($\{\boldsymbol{x}_{k-1}^{\{i_p\}}, w_{k-1}^{\{i_p\}}\}_{i_p=1}^{N}, \boldsymbol{z}_k$)
  for $i_p = 1, \ldots, N$ **do** ▷ Sample particles
    $\boldsymbol{x}_k^{\{i_p\}} \sim p\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i_p\}}\right)$
  end for
  for $i_p = 1, \ldots, N$ **do** ▷ Re-weight
    $w_k^{\{i_p\}} = p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k^{\{i_p\}}\right)$
  end for
  for $i_p = 1, \ldots, N$ **do** ▷ Normalize weights
    $\bar{w}_k^{\{i_p\}} = w_k^{\{i_p\}} \left(\sum_{i=1}^{N} w_k^{\{i\}}\right)^{-1}$
  end for
  $\{\boldsymbol{x}_k^{\{i_p\}}, \bar{w}_k^{\{i_p\}}\}_{i_p=1}^{N} = \text{RE-SAMPLE}(\{\boldsymbol{x}_k^{\{i_p\}}, \bar{w}_k^{\{i_p\}}\}_{i_p=1}^{N})$
  return $\{\boldsymbol{x}_k^{\{i_p\}}, \bar{w}_k^{\{i_p\}}\}_{i_p=1}^{N}$
end function

5.5.7 Point estimation

In many cases, the desired output of the particle filter might not be a set of particles but rather the best possible estimate of the true state of a system based on the particles. There are many different ways of choosing such an estimate. One possible way is simply picking the particle with the highest weight. But this might not give the desired result depending on the application.

An other way is finding the Minimum Mean Square Error Estimate (MMSE) by calculating the weighted mean and covariance of all particles. The weighted mean can be calculated as

$$\boldsymbol{\mu} = \sum_{i_p=1}^{N} \bar{w}^{\{i_p\}} \boldsymbol{x}^{\{i_p\}}, \tag{5.12}$$

where $\bar{w}^{\{i_p\}}$ is the normalized weight of particle i_p. This is for the logarithmic case described in Section 5.6.5.

The weighted empirical covariance of the particles can be calculated as

$$\boldsymbol{S} = \sum_{i_p=1}^{N} \left(\boldsymbol{x}^{\{i_p\}} - \boldsymbol{\mu}\right) \bar{w}^{\{i_p\}} \left(\boldsymbol{x}^{\{i_p\}} - \boldsymbol{\mu}\right)^{\mathsf{T}}. \tag{5.13}$$

5.6 Particle filter in logarithmic space

When likelihoods are calculated for the particle filter and small probabilities are multiplied together, the resulting likelihood can become too small to work with. To avoid numerical errors, it is possible to work in the log-space instead of working on regular probabilities. Multiplying two probabilities are then done by summing the logarithm of them. But then the problem of sampling occurs, where it is not trivial to sample the distribution using only the logarithm of each particle weight. This can be solved by converting the weights back to the linear space, normalizing them and then do the regular sampling of a discrete distribution. To utilize the log-space even further, the distribution can be normalized before being converted to the linear space by doing as described in Section 5.6.1. But in some cases, the likelihoods before sampling can be too small and give inaccuracies even after normalization. This can be solved by not leaving the log-space for the re-sampling.

A paper that goes through the whole process of using a particle filter in log-space is [23], which describes what the author calls Log-PF. This includes the weight updating, normalization, re-sampling and the estimation in log-space. Some of these methods are described below. Using logarithmic weights $\hat{w}_k^{\{i\}} = \ln\left(w_k^{\{i\}}\right)$ the

equation for the particle approximation of the probability density function becomes

$$p\left(\boldsymbol{x}_k \mid \boldsymbol{z}_{0:k}\right) \approx \sum_{i=1}^{N} \mathrm{e}^{\hat{w}_k^{\{i\}}} \delta\left(\boldsymbol{x}_k - \boldsymbol{x}_k^{\{i\}}\right).$$

5.6.1 Normalizing in log-space

As described in [23], the normalized weight

$$\bar{w}_k^{\{i\}} = \frac{w_k^{\{i\}}}{\sum_{j=1}^{N} w_k^{\{j\}}}$$

can be calculated in log-space as

$$\ln\left(\bar{w}_k^{\{i\}}\right) = \ln\left(\frac{w_k^{\{i\}}}{\sum_{j=1}^{N} w_k^{\{j\}}}\right) = \ln\left(w_k^{\{i\}}\right) - \ln\left(\sum_{j=1}^{N} w_k^{\{j\}}\right) = \hat{w}_k^{\{i\}} - \hat{W}_k,$$

where $\hat{W}_k$ is the logarithm of the sum of the weights. For this to be done purely in log-space it becomes

$$\hat{W}_k = \ln\left(\sum_{j=1}^{N} \mathrm{e}^{\hat{w}_k^{\{j\}}}\right).$$

To calculate such a sum, the Jacobian logarithm can be utilized. That is

$$\ln\left(\mathrm{e}^{\hat{w}^{\{1\}}} + \mathrm{e}^{\hat{w}^{\{2\}}}\right) = \max\left(\hat{w}^{\{1\}}, \hat{w}^{\{2\}}\right) + \ln\left(1 + \mathrm{e}^{-\left|\hat{w}^{\{1\}} - \hat{w}^{\{2\}}\right|}\right)$$

can be done iteratively by

$$\hat{W}^{(i)} = \ln\left(\sum_{j=1}^{i} \mathrm{e}^{\hat{w}^{\{j\}}}\right) = \max\left(\hat{W}^{(i-1)}, \hat{w}^{\{i\}}\right) + \ln\left(1 + \mathrm{e}^{-\left|\hat{W}^{(i-1)} - \hat{w}^{\{i\}}\right|}\right).$$

The normalization of weights in log-space is now reduced to subtracting the logarithm of the sum of the weights from the logarithm of the weights. The resulting algorithm is given in Algorithm 8.

5.6.2 Effective sample size from logarithms of weights

To calculate the effective sample size $\hat{N}_{\text{eff}}$ using the logarithms of weights we use (5.10) [23]

$$\ln\left(\hat{N}_{\text{eff}}\right) = -\ln\left(\sum_{i_p=1}^{N} \mathrm{e}^{2\hat{w}^{\{i_p\}}}\right),$$

where $\hat{w}^{\{i_p\}}$ is the logarithm of the normalized weight of particle with index i_p. The logarithm of the sum can be calculated using the Jacobian logarithm in Algorithm 8.

Algorithm 8 Iterative Jacobian logarithm

function JACOBILOG($\{\hat{w}^{\{i\}}\}_{i=1}^{N}$)
  $\hat{W} = \hat{w}^{\{1\}}$
  for $i = 2, \ldots, N$ **do**
    $\hat{W} = \max\left(\hat{W}, \hat{w}^{\{i\}}\right) + \ln\left(1 + \mathrm{e}^{-|\hat{W} - \hat{w}^{\{i\}}|}\right)$
  end for
  return $\hat{W}$
end function

5.6.3 Logarithmic low variance sampling

The method described in Algorithm 4 can be translated to use logarithmic weights. This is because $\ln(x)$ is strictly increasing for $x > 0$ and the cumulative distribution can be calculated using the iterative Jacobian logarithm Algorithm 8. The method is shown in Algorithm 9.

Algorithm 9 Sampling using weights in the logarithmic space. The logarithmic version of Algorithm 4, described in [23]. $\{\hat{w}^{\{i_p\}}\}_{i_p=1}^{N}$ are the logarithm of the normalized weights.

function LOG_RE-SAMPLE($\{\boldsymbol{x}_k^{\{i_p\}}, \hat{w}_k^{\{i_p\}}\}_{i_p=1}^{N}$)
  $r = \text{uniform}(0, N^{-1})$
  $\hat{W} = \hat{w}^{\{1\}}$ ▷ Initialize cumulative distribution
  $i_p = 1$
  for $n = 1, \ldots, N$ **do**
    $u = \ln(r + (n-1)N^{-1})$
    while $\hat{W} < u$ **do** ▷ Solve (5.11).
      $i_p = i_p + 1$
      $\hat{W} = \max\left(\hat{W}, \hat{w}^{\{i_p\}}\right) + \ln\left(1 + \mathrm{e}^{-|\hat{W} - \hat{w}^{\{i_p\}}|}\right)$
    end while
    $\boldsymbol{x}_k^{\{n\}*} = \boldsymbol{x}_k^{\{i_p\}}$
    $\hat{w}_k^{\{n\}*} = -\ln(N)$
  end for
  return $\{\boldsymbol{x}_k^{\{n\}*}, \hat{w}_k^{\{n\}*}\}_{n=1}^{N}$
end function

5.6.4 Sampling using the Gumbel-max trick

To re-sample the distribution without leaving log-space, the Gumbel distribution might be used. The Gumbel-max trick is described in [36]. To sample the standard Gumbel distribution, it is possible to simply take the negative logarithm twice of a

realization u of the standard uniform distribution

$$g^{\{i\}} = -\ln\left(-\ln\left(u^{\{i\}}\right)\right), \quad u^{\{i\}} \sim \mathcal{U}(0,1),$$

where i is the index of the sample. The sample of the discrete distribution can then be found through

$$i_p = \underset{i\in[1,N]}{\operatorname{argmax}}(\ln\left(w^{\{i\}}\right) + g^{\{i\}}),$$

where i_p is the index of the particle, N is the number of particles, $g^{\{i\}}$ is the sample from the Gumbel distribution and $w^{\{i\}}$ is the likelihood of the particle. This is also described in Algorithm 10. It is explained in [37] that $\operatorname{argmax}_{i\in[1,N]}(\phi(i) + g^{\{i\}})$ is distributed according to

$$\underset{i\in[1,N]}{\operatorname{argmax}}(\phi(i) + g^{\{i\}}) \sim \frac{\mathrm{e}^{\phi(i)}}{\sum_{i\in[1,N]}\mathrm{e}^{\phi(i)}},$$

where $\phi(i)$ is the log-unnormalized mass for a categorical distribution over classes $i \in [1, N]$.

This becomes in the case of logarithmic weights

$$\underset{i\in[1,N]}{\operatorname{argmax}}(\ln\left(w^{\{i\}}\right) + g^{\{i\}}) \sim \frac{\mathrm{e}^{\ln\left(w^{\{i\}}\right)}}{\sum_{i\in[1,N]}\mathrm{e}^{\ln\left(w^{\{i\}}\right)}} = \frac{w^{\{i\}}}{\sum_{i\in[1,N]} w^{\{i\}}}.$$

The re-sampling of the particle filter can therefore be done through doing the Gumbel-max trick for every particle in the filter. This has a time complexity of $\mathcal{O}(N^2)$.

Algorithm 10 Sampling a discrete distribution in log-space. From N particles the approximate sample of the discrete distribution is i_p. $u^{\{i\}}$ is a sample from the uniform distribution and $g^{\{i\}}$ is a sample from the Gumbel distribution.

1: **for** $i = 1, \ldots, N$ **do**
2: $\quad u^{\{i\}} = \text{uniform}(0, 1)$
3: $\quad g^{\{i\}} = -\ln\left(-\ln\left(u^{\{i\}}\right)\right)$
4: **end for**
5: $i_p = \operatorname{argmax}_{i\in[1,N]}(\ln\left(w^{\{i\}}\right) + g^{\{i\}})$

5.6.5 Point estimation in logarithmic space

Two methods for finding a state estimate from the particles are discussed in [23]. One of these is the MMSE which is defined as

$$\hat{\boldsymbol{x}}_{\text{MMSE}} = \sum_{i_p=1}^{N} w^{\{i_p\}}\boldsymbol{x}^{\{i_p\}},$$

where $\boldsymbol{x}^{\{i_p\}}$ is the state estimate of particle i_p, $w^{\{i_p\}}$ is the normalized weight and N is the number of particles. In log-space, this can be computed using the Jacobian logarithm, but it requires the separation of positive and negative values. Let

$$[\hat{\boldsymbol{x}}_{\text{MMSE}}]_l = \sum_{i_p=1}^{N} \mathrm{e}^{\hat{w}^{\{i_p\}}} \left[\boldsymbol{x}^{\{i_p\}}\right]_l$$

be the lth element of the state estimate. Let the indices of positive and negative values of $\left[\boldsymbol{x}^{\{i_p\}}\right]_l$ be represented by

$$\mathcal{I}_{+,l} = \left\{ i_p \mid i_p \in \{1, \ldots, N\} \bigwedge \left[\boldsymbol{x}^{\{i_p\}}\right]_l > 0 \right\},$$
$$\mathcal{I}_{-,l} = \left\{ i_p \mid i_p \in \{1, \ldots, N\} \bigwedge \left[\boldsymbol{x}^{\{i_p\}}\right]_l < 0 \right\}.$$

Then

$$\begin{aligned}
[\hat{\boldsymbol{x}}_{\text{MMSE}}]_l &= \sum_{i_p \in \mathcal{I}_{+,l}} \mathrm{e}^{\hat{w}^{\{i_p\}}} \left|\left[\boldsymbol{x}^{\{i_p\}}\right]_l\right| - \sum_{i_p \in \mathcal{I}_{-,l}} \mathrm{e}^{\hat{w}^{\{i_p\}}} \left|\left[\boldsymbol{x}^{\{i_p\}}\right]_l\right| \\
&= \sum_{i_p \in \mathcal{I}_{+,l}} \mathrm{e}^{\hat{w}^{\{i_p\}} + \ln\left(\left|\left[\boldsymbol{x}^{\{i_p\}}\right]_l\right|\right)} - \sum_{i_p \in \mathcal{I}_{-,l}} \mathrm{e}^{\hat{w}^{\{i_p\}} + \ln\left(\left|\left[\boldsymbol{x}^{\{i_p\}}\right]_l\right|\right)} \\
&= \mathrm{e}^{\ln\left(\sum_{i_p \in \mathcal{I}_{+,l}} \mathrm{e}^{\hat{w}^{\{i_p\}} + \ln\left(\left|\left[\boldsymbol{x}^{\{i_p\}}\right]_l\right|\right)}\right)} - \mathrm{e}^{\ln\left(\sum_{i_p \in \mathcal{I}_{-,l}} \mathrm{e}^{\hat{w}^{\{i_p\}} + \ln\left(\left|\left[\boldsymbol{x}^{\{i_p\}}\right]_l\right|\right)}\right)},
\end{aligned}$$

where the exponents can be calculated using the Jacobian logarithm in Algorithm 8.

5.7 Simulation filter evaluation

5.7.1 Monte Carlo simulation consistency test

Some consistency properties can be tested if the ground truth of the state to be estimated is known [6, p. 234]. These are:

- The state error should be acceptable as zero mean.
- The state error should have a magnitude corresponding to the covariance of the filter.

Let

$$\boldsymbol{e}_k = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \tag{5.14}$$

be the state error at time step k, where $\hat{\boldsymbol{x}}_k$ is the estimated state. We can then define the Normalized Estimation Error Squared (NEES) as

$$\epsilon_k = \boldsymbol{e}_k^{\mathsf{T}} \boldsymbol{S}_k^{-1} \boldsymbol{e}_k,$$

where $\boldsymbol{S}_k$ is the empirical covariance of the filter. Furthermore, the Average Normalized Estimation Error Squared (ANEES) is

$$\bar{\epsilon}_k = \frac{1}{N_{\text{runs}}} \sum_{i=1}^{N_{\text{runs}}} \epsilon_{k,i},$$

where $\epsilon_{k,i}, \quad i = 1, \ldots, N_{\text{runs}}$ are the NEES for different Monte Carlo simulations i.

The Linear Gaussian (LG) assumption is that the initial state, process noise and measurement noise is Gaussian and the state evolution and measurements are linearly dependant on the state. Under these assumptions and that the filter is consistent, called hypothesis H_0, ϵ_k is chi-square distributed with n_x degrees of freedom, where n_x is the dimension of the state $\boldsymbol{x}_k$[6, p. 234]. Moreover, the ANEES times the number of runs is chi-squared distributed with $N_{\text{runs}} n_x$ degrees of freedom, $N_{\text{runs}} \bar{\epsilon}_k \sim \chi^2_{N_{\text{runs}} n_x}$. This results in a two-sided consistency test acceptance interval for the ANEES with r_1 and r_2 being the sides of the interval

$$P\left\{\bar{\epsilon}_k \in [r_1, r_2] \mid H_0\right\} = 1 - \alpha,$$

where for example $\alpha = 0.05$ for a 95% confidence interval. $\bar{\epsilon}_k$ can then be plotted for each time step k together with the lower and upper bound r_1 and r_2 to see if the ANEES satisfies the test.

The consistency properties are important even if the LG assumptions do not hold. The confidence interval can then be used to say something about how well the error is covered by the Gaussian distribution with covariance equal to the empirical covariance.

5.7.2 RMSE

Knowing the true state of a robot in simulations makes it possible to calculate the Root Mean Square Error (RMSE) at time step k as

$$\mathbf{RMSE}_k = \sqrt{\frac{1}{N_{\text{runs}}} \sum_{i=1}^{N_{\text{runs}}} \boldsymbol{e}_{k,i} \odot \boldsymbol{e}_{k,i}}, \tag{5.15}$$

where $\boldsymbol{e}_{k,i}$ is the state error in (5.14) for different simulation runs i at time step k and $\boldsymbol{a} \odot \boldsymbol{b}$ takes the Hadamard product of $\boldsymbol{a}$ and $\boldsymbol{b}$, that is element wise multiplication. This results in the vector $\mathbf{RMSE}_k$ with the RMSE values for each component of the state at time step k from N_{runs} runs.

5.8 The unscented particle filter

The Unscented Particle Filter (UPF) uses a bank of UKFs in order to obtain a proposal for the importance density of a particle filter [64]. Using the SIR filter in Section 5.5.6 can fail if the likelihood of measurements is too peaked compared to the prior or when the measurements appear in the tail of the prior. Those cases can often occur when the sensors used are very accurate. This is illustrated in Figure 5.3, which shows the case of sampling using the prior versus sampling using UKF.

(a) Using prior

(b) Using UKF. The blue graph is the likelihood estimated from one UKF iteration.

Figure 5.3: Illustrates the different in particle sampling using the prior versus the UKF for estimating the importance density in the particle filter. The black dots are the particles.

The UPF is an implementation of the generic particle filter in Algorithm 6 where the optimal importance density is approximated for each particle using the UKF [63]. The resulting importance density becomes for a particle with index i_p

$$\boldsymbol{x}_k^{\{i_p\}} \sim q\left(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}^{\{i_p\}}, \boldsymbol{z}_k\right) = \mathcal{N}\left(\boldsymbol{x}_k; \hat{\boldsymbol{x}}_k^{\{i_p\}}, \boldsymbol{P}_k\right),$$

where $\hat{\boldsymbol{x}}_k^{\{i_p\}}$ and $\boldsymbol{P}_k$ is the output from the UKF using $\boldsymbol{x}_{k-1}^{\{i_p\}}$ and $\boldsymbol{P}_{k-1}$ as input. The corresponding weighting becomes

$$\bar{w}_k^{\{i_p\}} \propto \bar{w}_{k-1}^{\{i_p\}} \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k^{\{i_p\}}\right) p\left(\boldsymbol{x}_k^{\{i_p\}} \mid \boldsymbol{x}_{k-1}^{\{i_p\}}\right)}{q\left(\boldsymbol{x}_k^{\{i_p\}} \mid \boldsymbol{x}_{k-1}^{\{i_p\}}, \boldsymbol{z}_k\right)}, \qquad \text{(5.8 revisited)}$$

which should now be possible to evaluate directly.

5.9 Iterative Closest Point method

If the state of a robot is implicitly defined by the transform from one point cloud to another, this transform can be used for localization. If the correspondences between points in two different point clouds is known, the method of point set registration is to find the transform that minimizes the error between these known correspondences.

But in many cases the correspondences is not known. One way of solving this is to iteratively find the closest point in the fixed point cloud from every point in the moving point cloud, minimized that error and then find new correspondences. This is called the Iterative Closest Point (ICP) method.

One of the first articles on ICP was [8], which states that "The iterative closest point algorithm always converges monotonically to a local minimum with respect to the mean-square distance objective function." This is a great property for localization, but local minima not being global minima can become a problem as we will note later. The article describes how to register 3D point clouds, finding the 6 DOF transformation, and also minimizing the error from a point cloud to geometric entities like lines or triangle faces. But ICP can also be used in spaces of more or fewer dimensions. An nD implementation exists in the PCL[1] for C++, and also Matlab[2] has a 3D implementation.

The authors of [8] preferred using quaternions over the Singular Value Decomposition (SVD) [2] approach to minimize the error. They seem to be using the Hamilton convention [59], with the real part first, right-handedness, operator being passive and operator being local-to-global. The quaternions have been used to make a rotation matrix: The unit rotation quaternion $\boldsymbol{q}_R = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^\mathsf{T}$, where $q_0^2+q_1^2+q_2^2+q_3^2 = 1$, gives the rotation matrix

$$\boldsymbol{R}(\boldsymbol{q}_R) = \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & q_0^2+q_2^2-q_1^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2+q_3^2-q_1^2-q_2^2 \end{bmatrix}.$$

The complete registration state vector is $\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_R^\mathsf{T} & \boldsymbol{q}_T^\mathsf{T} \end{bmatrix}^\mathsf{T}$ where $\boldsymbol{q}_T = \begin{bmatrix} q_4 & q_5 & q_6 \end{bmatrix}^\mathsf{T}$ is the translation vector. Let $\boldsymbol{P} = \{\boldsymbol{p}_i\}$ be the set of measured data points and $\boldsymbol{X} = \{\boldsymbol{x}_i\}$ be the model points, where the number of points $N_X = N_P$ and each point in $\boldsymbol{P}$ corresponds to the point i $\boldsymbol{X}$ with the same index. For point clouds of different sizes, $N_X = N_P$ is enforced through the closest point method, finding the closest point in the other point cloud for every point in the first cloud. The mean square objective function to be minimized is

$$f(\boldsymbol{q}) = \frac{1}{N_P}\sum_{i=1}^{N_P} \left\| \boldsymbol{x}_i - \boldsymbol{R}(\boldsymbol{q}_R)\boldsymbol{p}_i - \boldsymbol{q}_T \right\|^2.$$

Using the means for each point set

$$\boldsymbol{\mu}_P = \frac{1}{N_P}\sum_{i=1}^{N_P} \boldsymbol{p}_i \quad \text{and} \quad \boldsymbol{\mu}_X = \frac{1}{N_X}\sum_{i=1}^{N_X} \boldsymbol{x}_i,$$

[1] The point registration library in the PCL can be found here: `http://docs.pointclouds.org/trunk/group__registration.html`.

[2] A point cloud registration method for Matlab can be found here: `https://se.mathworks.com/help/vision/ref/pcregistericp.html`.

the cross covariance matrix becomes

$$\boldsymbol{\Sigma}_{PX} = \frac{1}{N_P}\sum_{i=1}^{N_P}(\boldsymbol{p}_i - \boldsymbol{\mu}_P)(\boldsymbol{x}_i - \boldsymbol{\mu}_X)^\mathsf{T}.$$

The vector $\boldsymbol{\Delta} = \begin{bmatrix}\boldsymbol{A}_{23} & \boldsymbol{A}_{31} & \boldsymbol{A}_{12}\end{bmatrix}^\mathsf{T}$ is made from $\boldsymbol{A} = \boldsymbol{\Sigma}_{PX} - \boldsymbol{\Sigma}_{PX}^\mathsf{T}$ and the symmetric 4×4 matrix $\boldsymbol{Q}\left(\boldsymbol{\Sigma}_{PX}\right)$ is made

$$\boldsymbol{Q}\left(\boldsymbol{\Sigma}_{PX}\right) = \begin{bmatrix}\mathbf{tr}\left(\boldsymbol{\Sigma}_{PX}\right) & \boldsymbol{\Delta}^\mathsf{T} \\ \boldsymbol{\Delta} & \boldsymbol{\Sigma}_{PX} + \boldsymbol{\Sigma}_{PX}^\mathsf{T} - \mathbf{tr}\left(\boldsymbol{\Sigma}_{PX}\right)\boldsymbol{I}_{3\times 3}\end{bmatrix}.$$

The optimal rotation is found as the unit eigenvector corresponding to the maximum eigenvalue of the matrix $\boldsymbol{Q}\left(\boldsymbol{\Sigma}_{PX}\right)$ and the optimal rotation is found as

$$\boldsymbol{q}_T = \boldsymbol{\mu}_X - \boldsymbol{R}(\boldsymbol{q}_R)\boldsymbol{\mu}_P.$$

This process is denoted as

$$(\boldsymbol{q}, \boldsymbol{d_{ms}}) = \boldsymbol{Q}(\boldsymbol{P}, \boldsymbol{X}), \tag{5.16}$$

where $\boldsymbol{d_{ms}}$ is the mean square point matching error. Other methods exist for finding possible steps $\boldsymbol{q}$, for example SVD [2] or solve a least squares problem by linearizing the measurement function on a manifold using Lie algebra [12, 18].

The resulting algorithm is shown in Algorithm 11, where $\boldsymbol{\tau}$ is a threshold for when the error is small enough. An illustration of two 2D point clouds being matched is shown in Figure 5.4. For each iteration, the closest points are calculated. Then the measured data is transformed by the method described above and new closest points are being calculated. This is done iteratively and results in the clouds in Figure 5.4a being transformed into the clouds in Figure 5.4b.

(a) Before ICP.

(b) After ICP.

Figure 5.4: Two point clouds being matched using ICP. The gray dots are point in the model $\boldsymbol{X}$, and the orange points are in the measured data points $\boldsymbol{P}$.

Algorithm 11 The ICP algorithm from [8].

function ICP($\boldsymbol{P} = \{\boldsymbol{p}_i\}_{i=1}^{N_P}, \quad \boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{N_X}$)
  $\boldsymbol{P}_0 = \boldsymbol{P}, \quad \boldsymbol{q}_0 = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^\mathsf{T}, \quad k = 0$
  repeat
    Compute the closest points: $\boldsymbol{Y}_k = \boldsymbol{C}(\boldsymbol{P}_k, \boldsymbol{X})$.
      Cost: $\mathcal{O}(N_P N_X)$
    Compute the registration: $(\boldsymbol{q}_k, \boldsymbol{d_k}) = \boldsymbol{Q}(\boldsymbol{P}_0, \boldsymbol{Y}_k)$ from (5.16).
      Cost: $\mathcal{O}(N_P)$
    Apply the registration:
      $\boldsymbol{P}_{k+1} = \left\{\boldsymbol{R}(\boldsymbol{q}_R)\boldsymbol{p}_i + \boldsymbol{q}_T \mid \boldsymbol{p}_i \in \boldsymbol{P}_0, i \in 1, \ldots, N_P, \boldsymbol{q}_k = \begin{bmatrix}\boldsymbol{q}_R^\mathsf{T} & \boldsymbol{q}_T^\mathsf{T}\end{bmatrix}^\mathsf{T}\right\}$
      Cost: $\mathcal{O}(N_P)$
  until $\boldsymbol{d_k} - \boldsymbol{d_{k+1}} < \boldsymbol{\tau}$
end function

Chapter 6

Particle filter analysis and simulations

Simulations have been done in Matlab. This was chosen because Matlab is a great environment to test methods and visualize different kinds of data in. The simulations use synthetic velocity data and measurement data as described in Section 4.3 and Section 4.6 respectively. Remember from Chapter 4 that the map used is based on gathered data, but the measurements created using that map and ground truth state trajectory are synthetically generated.

6.1 Particle filter method

Two main reasons for using the particle filter in this problem are:

- The particle filter is able to represent multi-modal distributions. For areas of symmetry in the map, the lidar scans might not be enough to differentiate different possible states, which makes it necessary to represent more than one possible state. It also enables global localization, where the initial position is unknown.
- The requirements on the sensor model and process model are weak. There are requirements on the possibility to evaluate certain likelihoods and sampling certain distributions. Which distributions that have to be sampled depends on the type of particle filter used. This enables the use of a sensor model which is so non-linear that it is difficult to formulate as a function of the state, which is the case for the model and map used here.

This section describes the components used in the particle filter. A simple map is

used in this analysis, see Figure 6.1. The simulations in the bigger map are described later in Section 6.3. We will use this map in order to test how different properties of a map changes the properties of the filter.

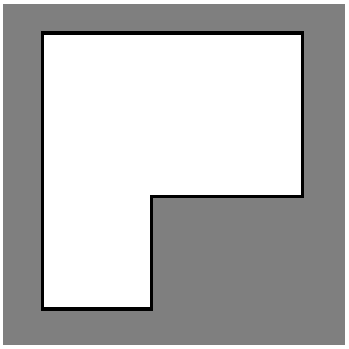


Figure 6.1: A simplified map to analyze the behavior of the particle filter in the relevant application. The resolution is $0.5\,\frac{\mathrm{m}}{\mathrm{px}}$, making the stripe of occupied cells 1 px thick.

6.1.1 Weighting and importance density

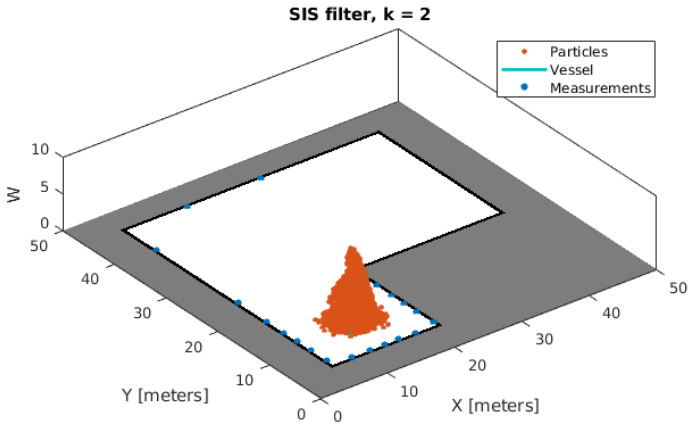
It is quickly noticed that the values multiplied together becomes too small for Matlab, so the logarithms of the weights are used, see Section 5.6. The chosen importance density is in this chapter the prior, but the later chapters will show the use of different importance densities. The weight update equation using the prior becomes

$$w_k^{\{i\}} \propto w_{k-1}^{\{i\}} p(\boldsymbol{z}_k \mid \boldsymbol{x}_k^{\{i\}}). \qquad \text{(5.9 revisited)}$$

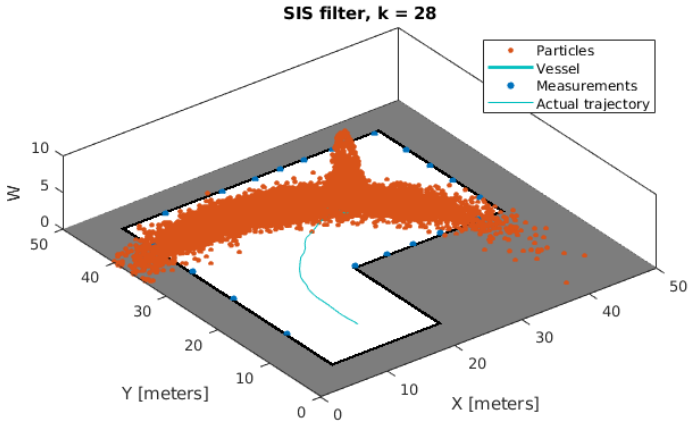
The multiplication is substituted by summing. So the re-weighting with the sensor model becomes summing the log of the likelihood of each single measurement, using the sensor model in Section 4.6 for each measurement.

The prior uses the dynamics from Section 4.3 with the Gaussian noise as described in Appendix A.4. In the sampling of the importance density, each particle is moved with the dynamics and added noise. An illustration of the behavior of the resulting SIS filter, see Algorithm 3, is shown in Figure 6.2. The weights have been scaled to have the maximum height of 10 for viewing the distribution. The variance of single measurements has been set to $\sigma_{\text{hit}}^2 = 30\,\text{m}^2$ for illustration purposes in the weighting, but the measurements used have no noise on them, as can be seen by the measurements in the figure hitting exactly the occupied spaces. The weights have been set equal after re-sampling in each iteration, otherwise almost all particles would get a weight of zero due to particle impoverishment, which we will come back to. Here the whole state $\boldsymbol{x} = \left[E, N, \psi\right]^\mathsf{T}$ is estimated. The figure illustrates how the importance density over time spreads the particles out in E and N, but they are also spread out in heading ψ. It also shows the weighting of the particles and the

Gaussian nature of this simple estimation. A non-linearity can be observed from the distribution in Figure 6.2a, but this will be discussed later when we will see that the map can make the state distribution less like a Gaussian.



(a) Initial distribution



(b) Final distribution

Figure 6.2: Illustration of the spread from a SIS filter.

6.1.2 Normalization and re-sampling

In order to not use particles with very low values and thereby reduce the spread of the particles, re-sampling is done, as mentioned in Section 5.5.4. This requires normalization which is done using the logarithmic weights as described in Section 5.6.1.

The re-sampling is first implemented using the Gumbel-max trick, as described in Section 5.6.4. After the sampling seems to work, the low variance sampling in logarithmic space as shown in Algorithm 9 is used. This is in order to have increased robustness and based on the preferences of the authors of [1]. And after the re-sampling, the particles are given equal weights as in Monte Carlo simulations.

Re-sampling at every time step makes this the SIR filter, described in Algorithm 7. The resulting distribution is shown in Figure 6.3. It illustrates how the particles are not spread out in areas of low likelihood when the distribution is re-sampled at every time step, compared to Figure 6.2b.

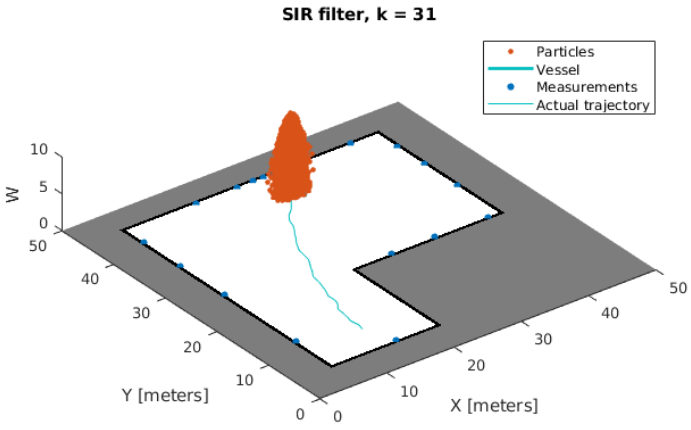


Figure 6.3: The final distribution when using the SIR filter.

6.1.3 Evaluating the filter

To evaluate the simulations, the consistency can be tested using ANEES as described in Section 5.7.1. The ANEES is then compared to the corresponding confidence interval. The empirical covariance can be found using (5.13) and the position estimated using (5.12). The error can be evaluated using RMSE as shown in (5.15).

6.1.4 Regularized re-sampling

The particle filters used in this project do utilize regularization. The regularization is implemented as described in Section 5.5.4 using a Gaussian kernel. In order to illustrate what the regularization does, the only state estimated is the heading. If the particles before regularization in Figure 6.4 are to be used further and there is low process noise, each particle will not move much relative to the actual path and the distribution will not be represented well. The actual state will in that case approximately stay between the same particles for future time steps. If regularization is used, the empirical covariance of the particles is calculated and each particle is spread in order to represent the distribution better. In the figure, the particles have been re-weighted in order to show the distribution. But all particles are given equal weight after the re-sampling step.

The author believes that it is important in this case to differentiate between problems of low process noise relative to measurement noise and problems of high process

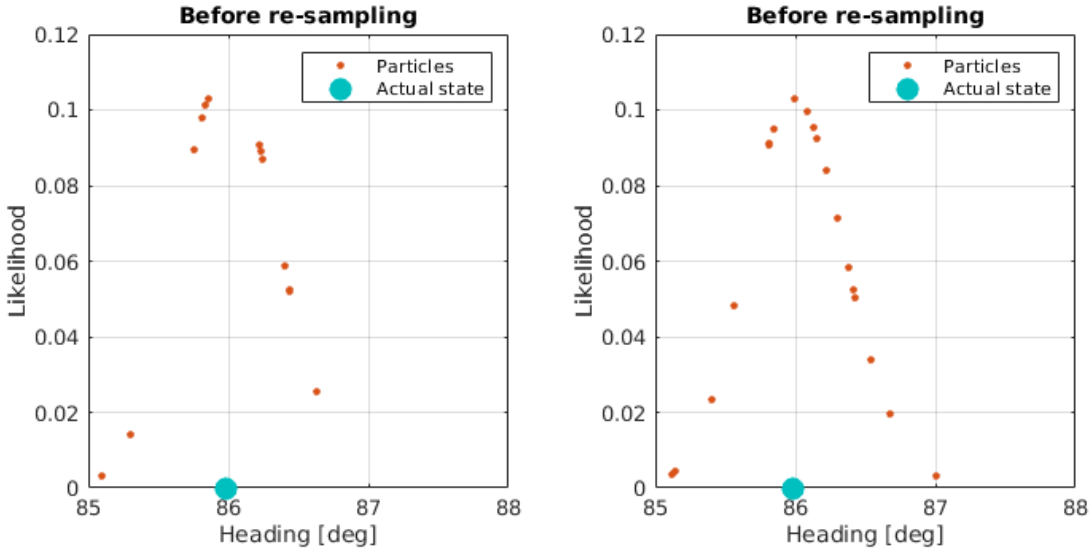
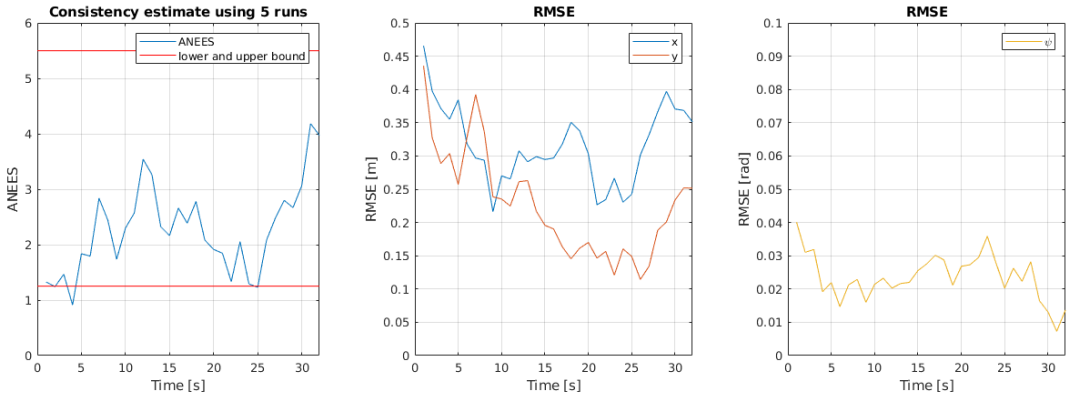


Figure 6.4: The particle distribution before and after regularization.

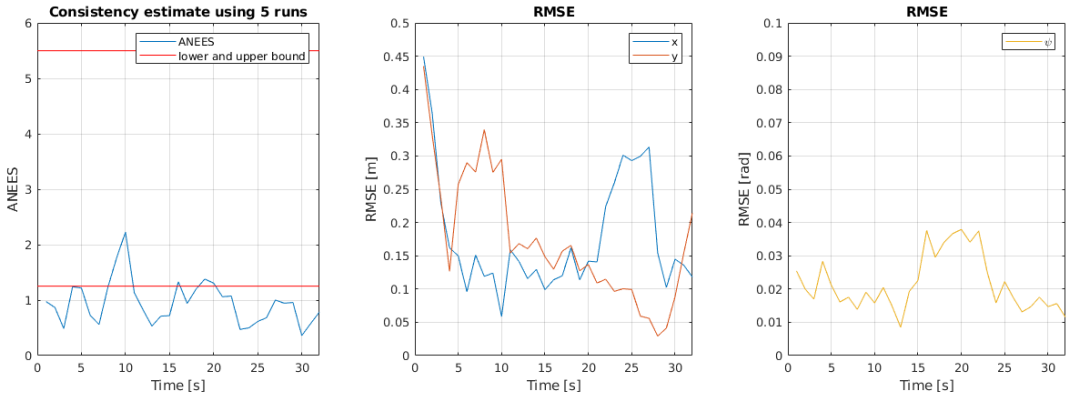
noise relative to measurements. Using the terminology of [24] these are low and high Signal to Noise Ratio (SNR) respectively. When the process noise is small, the regularized particle filter can be a good idea, as described above. When the process noise on the other hand is big, the particles will be spread a lot in the sampling and the spread of the regularization makes minimal difference when it comes to the sampled state of a particle.

The difference between low and high process noise also changes how one might view the particles. If the process noise is small it is reasonable to look at each particle as representing a proposal trajectory of the state. This makes sense based on the derivation of the Bayes filter. But in the case of high process noise and the use of the SIR filter, the particles can represent the distribution well, even though each particle might have noise realizations that make the corresponding trajectory very different from the actual trajectory.

To further illustrate the effect of the regularized particle filter, the simulations run with low process noise compared to the measurement noise. The whole 3 DOF are used. Visual inspection of the particles showed that without regularization, the particles are often in a small but likely area. But with regularization the particles are more spread out. The resulting ANEES and RMSE is shown in Figure 6.5. The ANEES is consistently smaller when regularization is used. The spread of the particles introduced by the regularization might be the reason why the ANEES is below the lower bound of the confidence interval.



(a) Without regularization



(b) With regularization

Figure 6.5: Evaluation of particle filter with and without regularization when low process noise is used. Uses $N_{\text{runs}} = 5$ and a 95% confidence interval.

6.2 Particle filter properties

This section looks at the properties in the application of this project. The same map as earlier in Figure 6.1 is used in order to test how different properties of a map changes the properties of the filter. Mostly the heading is estimated and used for visualization, but the properties in x- and y-direction seems to be similar. The properties are specific to this sensor model and type of map and will be used in order to discuss the behavior of the particle filter in the three DOF case.

6.2.1 Corners

The map used does not give a linear relation between the state an measurements. And it does not guarantee a Gaussian state distribution when only Gaussian noise is used for both process noise and measurement noise. Lets look at Figure 6.1, it has five outward pointing corners and one inward pointing corner. Both of these types of corners give certain artifacts shown in Figure 6.6. The inward corner, whose artifact is shown in Figure 6.6b, gives the possibility that when the heading crosses a certain angle, a ray will pass the corner and hit the back wall. The range then becomes very big in comparison with the measurement and that state gets a low likelihood. The artifact of an outward pointing corner, shown in Figure 6.6c, seems to be different variances depending on the state. If for example the vessel is far to the left in Figure 6.1 and a ray hits exactly the top left corner, giving too much heading will change the range measurement a lot, making the likelihood narrow. And too little heading will give a measurement on the top wall, where the measured range will not change as much depending for a small change in heading.

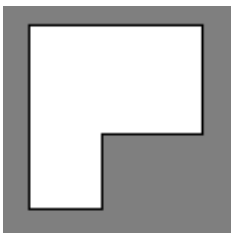
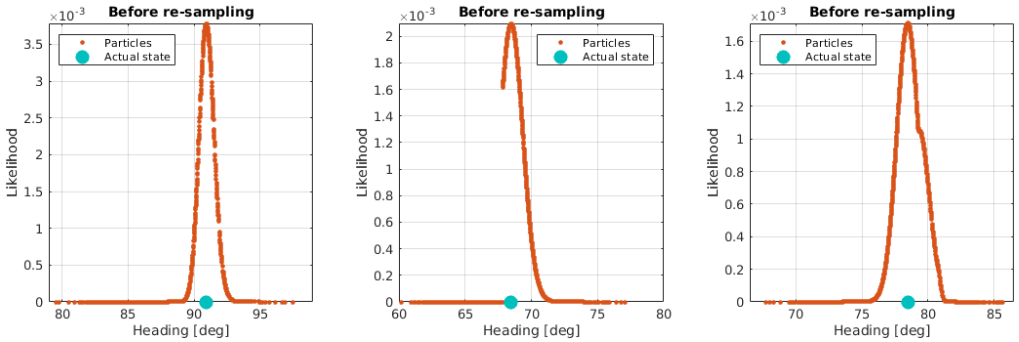


Figure 6.1 revisited.



(a) Usual (b) Inward corner (c) Outward corner

Figure 6.6: Different artifacts using the map in Figure 6.1. The y-axes have different scales in order to compare the distributions.

6.2.2 Bad importance density

In the case of a narrow $p(\boldsymbol{z}_k \mid \boldsymbol{x}_k)$ compared to the process prior $p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})$, using the prior as the importance density might be a bad approximation. Such a case is illustrated in Figure 6.7. It results in few particles in the area of highest likelihood and the particles not representing the distribution well.

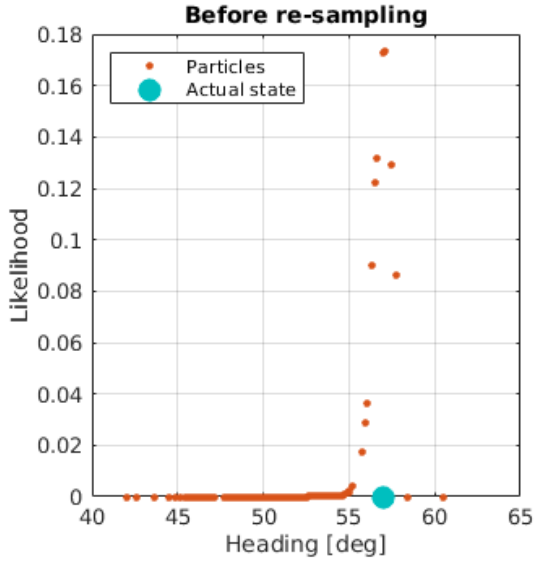


Figure 6.7: Illustrates what might happen when using the prior as the importance density.

6.2.3 Diagonal map

The map used earlier is very simple in that the discretization of the map is not visible. In order to show the discretization the map can be rotated, as shown in Figure 6.8. The sides of this are not straight, but have been pixelated. If the ray tracing finds the closest point on those cells and not the center point, as in our implementation, this will affect the estimation as shown in Figure 6.9. The figure shows how the discretization affects the distribution $p(\boldsymbol{z}_k \mid \boldsymbol{x}_k)$. It becomes less smooth and seems to consist of many combinations of the corner artifacts described earlier. Increasing the measurement variance makes the distribution better represented because it becomes wider and more particles are in the core of the distribution. The increase of measurement noise also increases the spread of the particles via the re-sampling step.

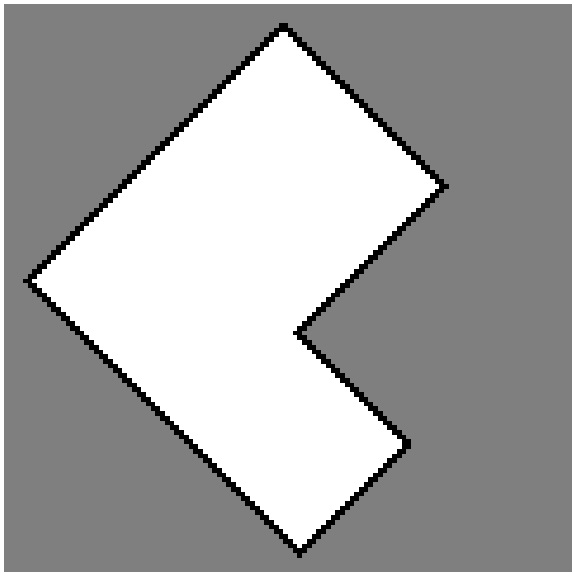
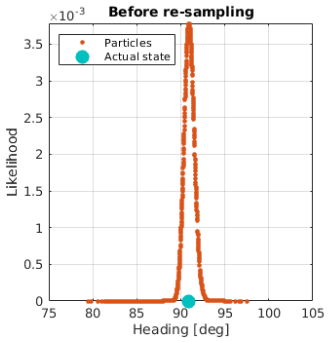
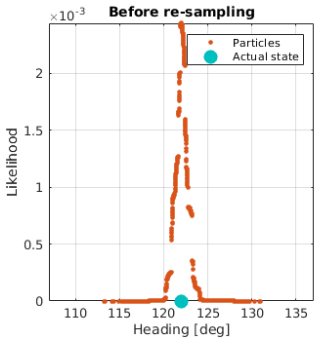


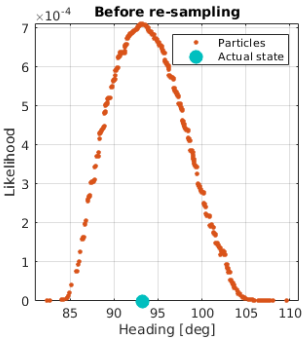
Figure 6.8: Diagonal map with pixelated lines.



(a) Normal map



(b) Diagonal map.

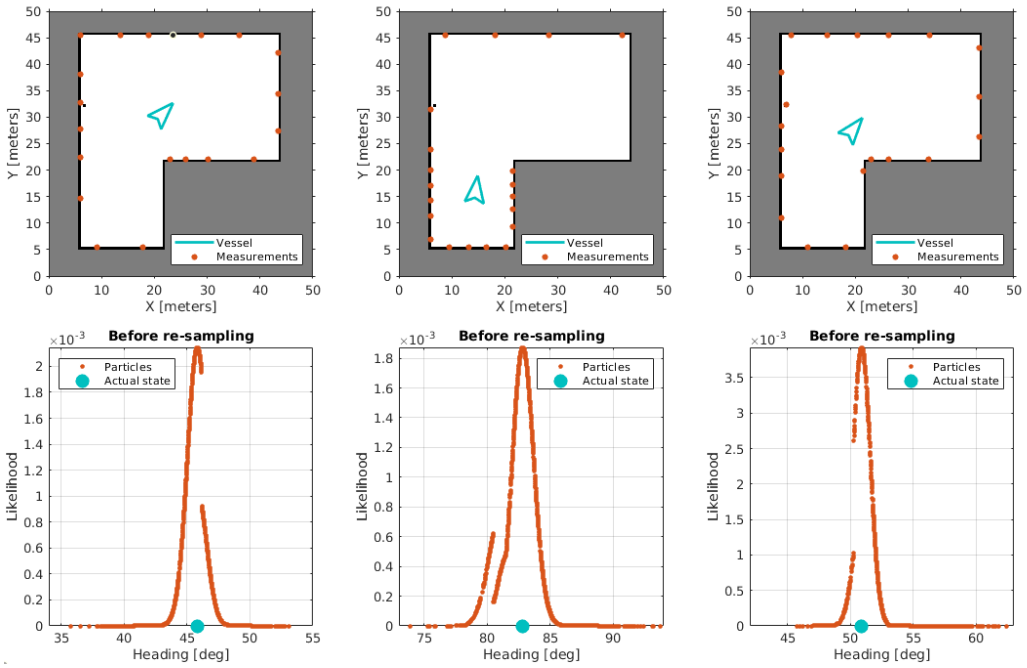


(c) Diagonal map with more measurement noise.

Figure 6.9: Difference between using straight and diagonal lines in a discretized map. The y-axes have different scales in order to compare the distributions.

6.2.4 Noise

Small details in the map can change the distribution by for example creating discontinuities. One such example is shown in Figure 6.10 where one occupied cell is placed to the left in the map. This discontinuity decreases the range measurements when hit. If the discontinuity increases the range instead of decreasing it, the results are assumed to be similar. This can reduce the likelihood of a state if that state hits either on or off the occupied cell, shown in Figure 6.10a and Figure 6.10c respectively. But it is also possible for the new occupied cell to increase the likelihood of a state, as shown in Figure 6.10b. There, a decrease in heading gives reduced likelihood until the occupied cell is hit, the measured range of that ray is reduced a bit, resulting in a more likely state.



(a) Decreasing the likelihood when hitting the discontinuity.

(b) Increasing the likelihood when hitting the discontinuity.

(c) Decreasing the likelihood when missing the discontinuity.

Figure 6.10: How the distribution is changed by details in the map. Note the one occupied cell to the left in the map. The y-axes of the likelihoods have different scales in order to compare the distributions.

The map can be not so smooth in many different ways, but all are in some way a combination of the properties already discussed. Some examples are shown in Figure 6.11. The different figures are:

- Figure 6.11a: Case 1. This shows a regular pattern of occupied cells on the left side of the map. It results in several discontinuities of the distribution.

- Figure 6.11b: Case 2. A lot of small occupied cells are introduced to the map, giving many discontinuities. The corresponding distribution is only high in a very small region.
- Figure 6.11c: Case 3. The map borders have been made into curvatures. The corresponding distribution seems to be in between the normal distribution and the one from the diagonal map.
- Figure 6.11d and Figure 6.11e: Case 4. More realistic structures with curved walls and some bigger structures in the map. The corresponding distribution is mostly similar to Case 3, but might have some more big discontinuities due to the bigger structures.

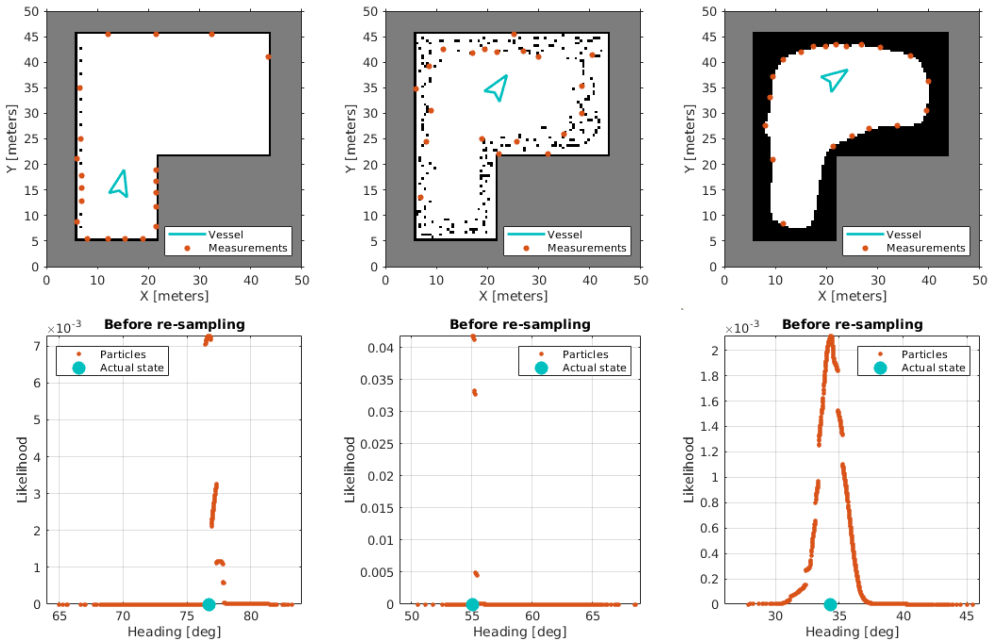
6.2.5 Localize in another map

If we try to use one map for localization, while the actual measurements have noise induced by a noisy map, the result might look similar to Figure 6.12. We see that the distance between the actual state and the mean of the distribution made from noisy measurements depends on the noise.

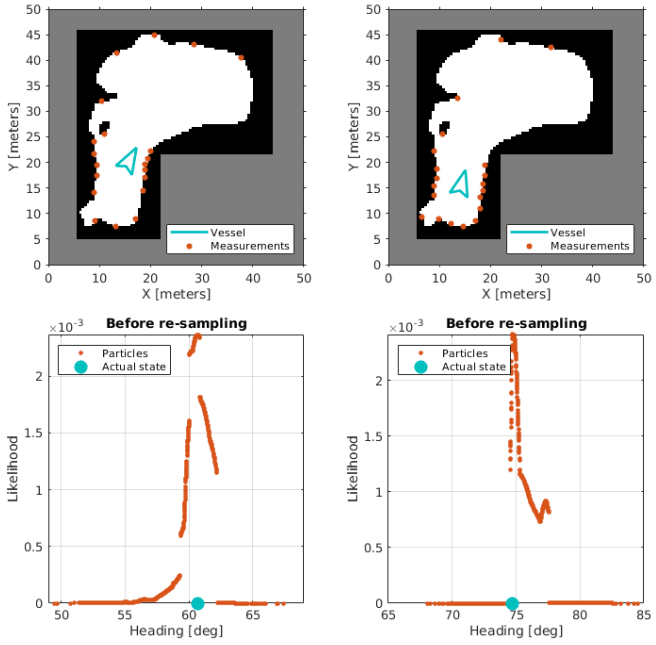
In order for the particles to give higher weight to the actual state in the cases of noise, the parameters in the sensor model from Section 4.5.3 can be tuned, see Figure 6.13. The parameters from Section 4.6 are the ones in Figure 6.13d. We see from the figure how the weighting of the uniform noise affects the distribution. The actual state gets increased weight as the Gaussian weight is reduced and the uniform weight is increased. In the extreme case of $\alpha_{\text{rand}} = 1$ all particles get the same weight. The difference between increasing he variance of the Gaussian and increasing the weight of the uniform distribution is that increasing the variance affects the closest points to the peak of the distribution, while increasing the weight of the uniform part increases all particles equally.

6.2.6 Using the filter map

Attempting to estimate heading using the realistic map is illustrated in Figure 6.14 where the filter map from Section 4.6 is used. Rays that go beyond the maximum range z_{max} are given the value of $z = z_{\text{max}}$. It is possible to see that the result is a combination of the different possible types of noise described earlier. The increased distance to the occupied cells also makes more happen as the heading is changed, explaining the narrower distribution compared to earlier. The map can also contain any sort of noise, and the estimate can contain any of the different artifacts described above.

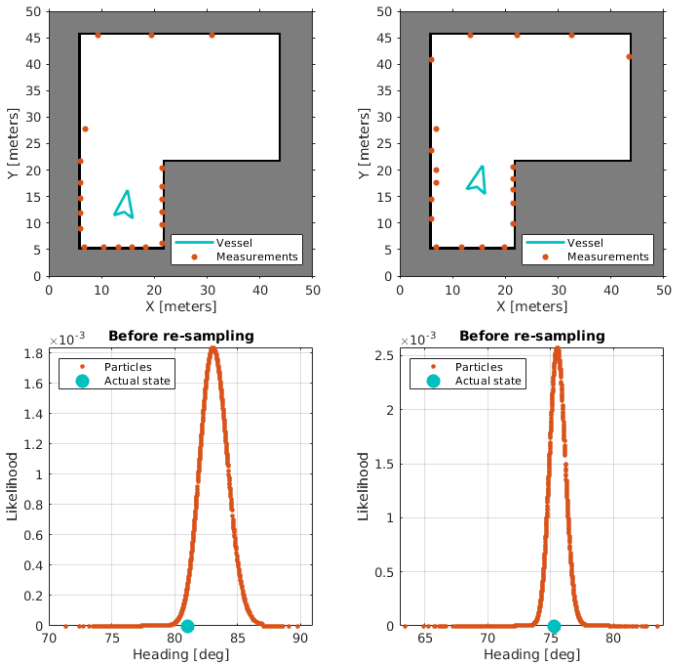


(a) Case 1 (b) Case 2 (c) Case 3



(d) Case 4 (e) Case 4

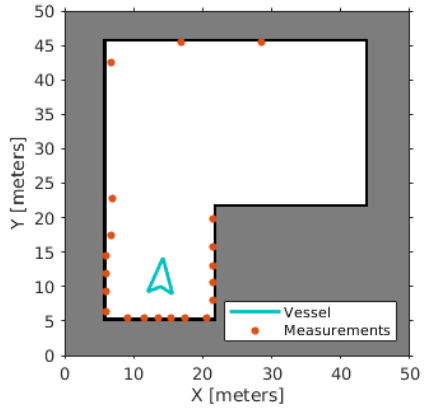
Figure 6.11: Different maps and what the corresponding distributions look like. The y-axes of the likelihoods have different scales in order to compare the distributions.



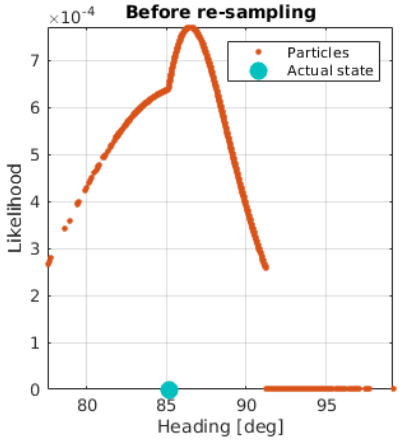
(a) Noticeable error from distribution mean to actual state.

(b) Smaller error from mean to actual state.

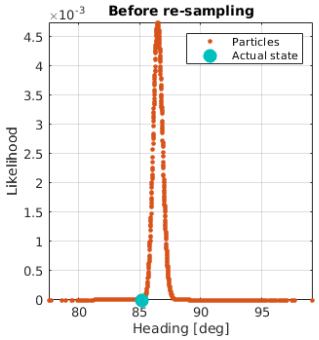
Figure 6.12: Localizing in a different map than where measurements are from. The distance between the actual state and the mean of the distribution changes depending on the noise on the measurements. The y-axes of the likelihoods have different scales in order to compare the distributions.



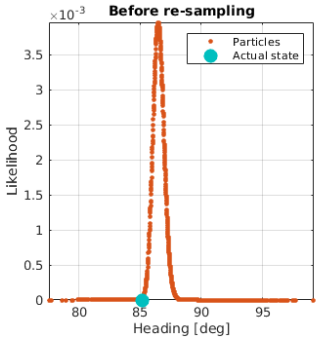
(a) Actual state and measurements



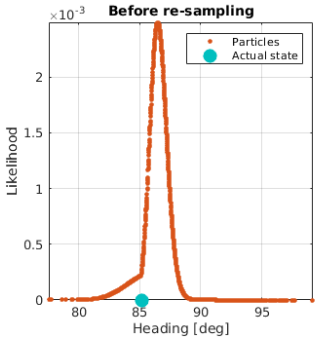
(b) $\sigma^2_{\text{hit}} = 30$, $\alpha_{\text{rand}} = 0$



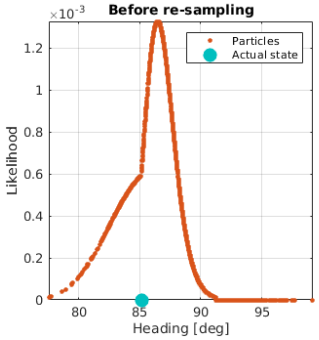
(c) $\sigma^2_{\text{hit}} = 0.7$, $\alpha_{\text{rand}} = 0$



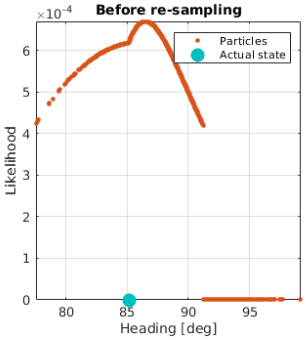
(d) $\sigma^2_{\text{hit}} = 0.7$, $\alpha_{\text{rand}} = 0.3$



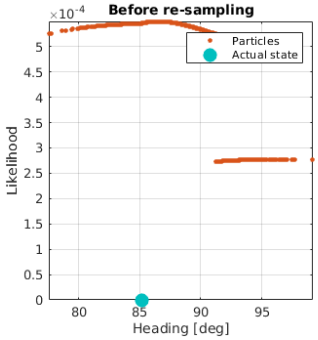
(e) $\sigma^2_{\text{hit}} = 0.7$, $\alpha_{\text{rand}} = 0.7$



(f) $\sigma^2_{\text{hit}} = 0.7$, $\alpha_{\text{rand}} = 0.9$

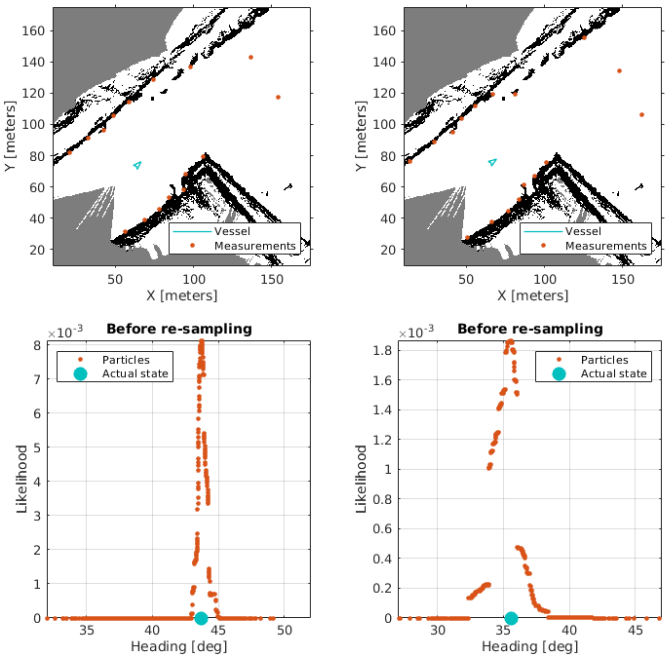


(g) $\sigma^2_{\text{hit}} = 0.7$, $\alpha_{\text{rand}} = 0.99$



(h) $\sigma^2_{\text{hit}} = 0.7$, $\alpha_{\text{rand}} = 0.999$

Figure 6.13: Localizing in a different map than where measurements are from. The different figures are made using different values of σ^2_{hit} and α_{rand}, the tuning parameters for the sensor model in Section 4.5.3. The y-axes of the likelihoods have different scales in order to compare the distributions.



(a) Normal variance. (b) Increased variance.

Figure 6.14: Estimating only heading in the realistic map. The two illustrations have different variance for the sensor noise. The y-axes of the likelihoods have different scales in order to compare the distributions.

When this method is going to be run on estimating the full 3 DOF, the artifacts mentioned will affect all states. The filter will need to have a lot more particles to have as good an estimation of heading. Figure 6.14 shows a distribution that is not exactly Gaussian, and when more states need to be estimated and the particles do not represent the distribution as well, approximating this distribution as Gaussian might not be accurate. This problem is illustrated in the next section.

6.3 Particle filter simulations

This section runs the generic particle filter algorithm shown in Algorithm 6, described and analyzed above, in a more realistic yet simulated run. It uses noise on measurements and weights them using a more realistic uncertainty. Some important parameters mentioned earlier and their values in this simulation:

- $N = 100$: the number of particles. This is not many, but using more means slower computations and using less means not being able to represent the distribution well.
- $N_m = 100$: the number of measurements. This is a lot less than the 1800 rays of the real sensor. But using more means slower computation. Using less means less data to use for estimation. The measurements are made synthetically.
- $\Delta_k = 1\,\mathrm{s}$: time step. This decides how much the process noise will spread the particles, and how much the vessel can move randomly between different time steps.

The filter is initialized around the initial true state distributed with a Gaussian with standard deviation $\sqrt{3}\,\mathrm{m}$, $\sqrt{3}\,\mathrm{m}$ and $\sqrt{0.1}\,\mathrm{rad}$ for x-, y-position and heading respectively. The particles are given equal weights. But the implementation has also proved able to handle a uniform distribution over the whole map, given that enough particles have been used. A constant velocity used a lot in this project is $\boldsymbol{\nu}_{k,\mu} = \begin{bmatrix} 15\,\frac{\mathrm{km}}{\mathrm{h}} & 0\,\frac{\mathrm{km}}{\mathrm{h}} & -2\,\frac{^\circ}{\mathrm{s}} \end{bmatrix}^\mathsf{T}$ based on the authors assumptions on typical speeds for a ship. An example of the localization being run is shown in Figure 6.15. An illustration of all the steps in the algorithm is shown in Figure 6.16 and all steps are discussed below.

6.3.1 Weighting and importance density

In Figure 6.16 after sampling the particles have weights 0.01, since they have been given equal weights after the last re-sampling step. As can be seen in Figure 6.17, the likelihood of different particles is small after weighting. This is probably due to the fact that many narrow Gaussians are implicitly being multiplied together

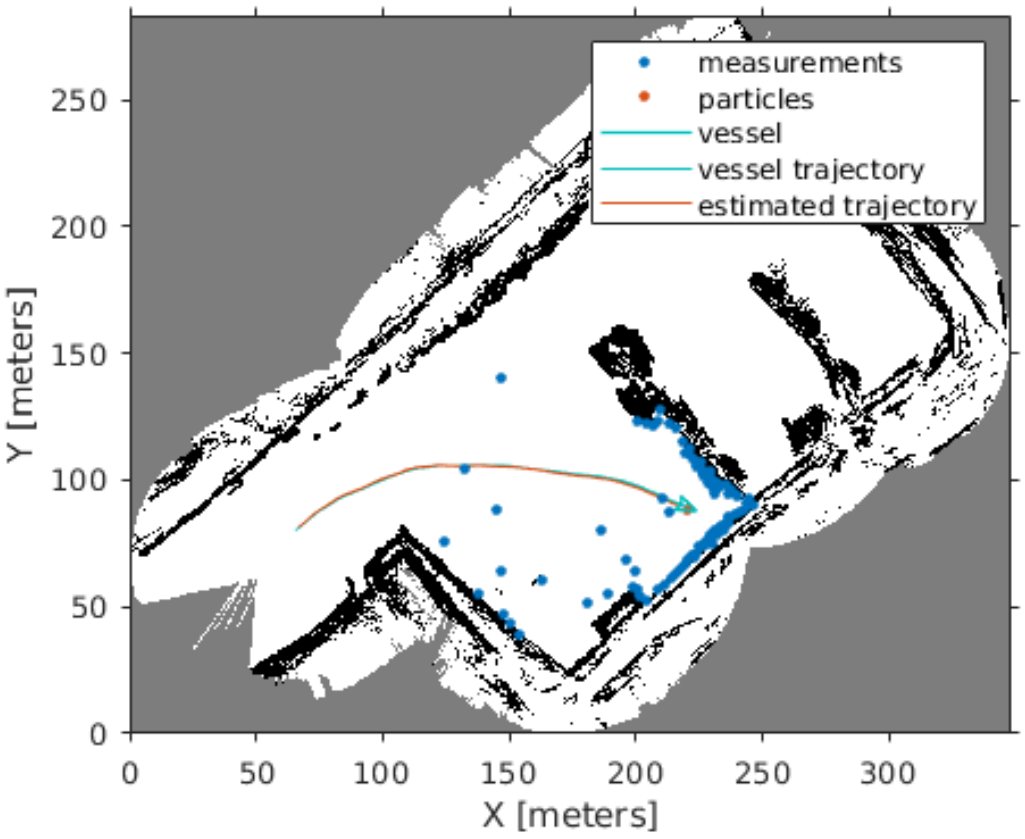


Figure 6.15: The figure shows actual path, estimated path, measurements and particles after re-sampling.

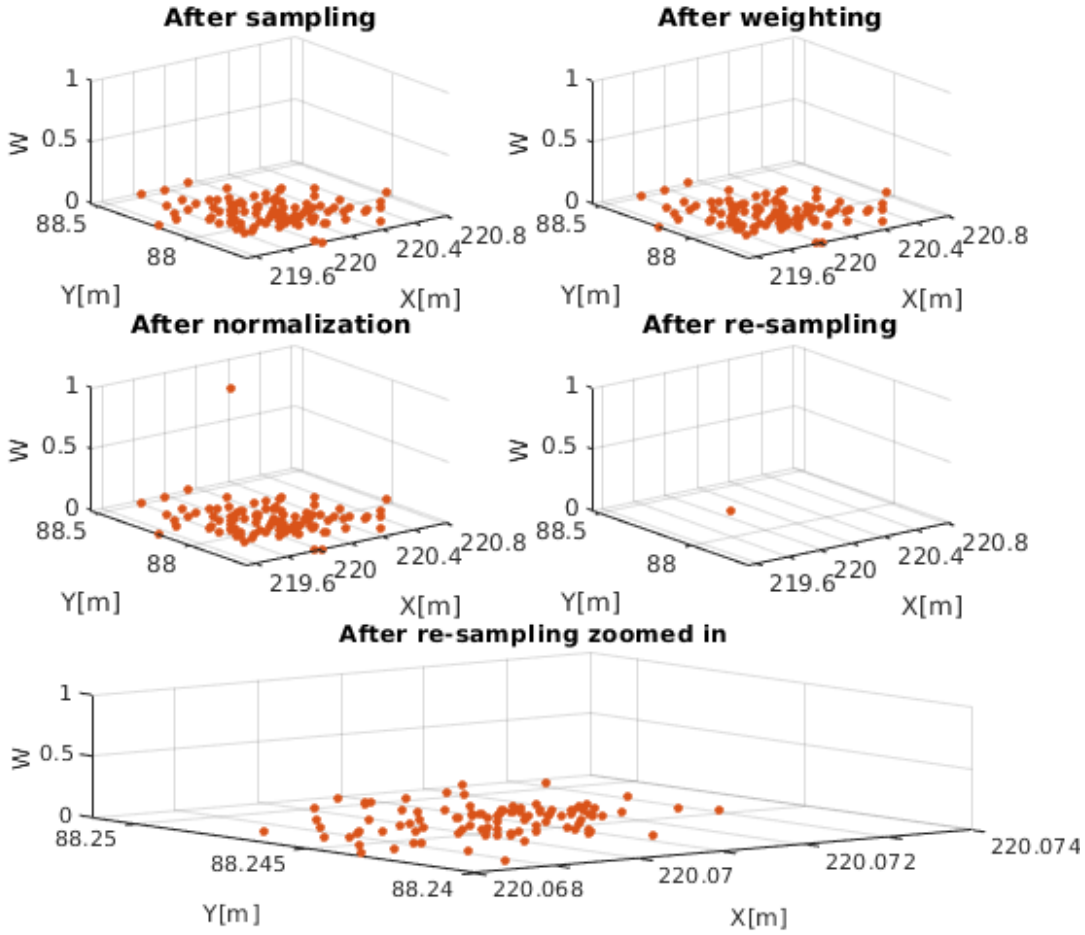


Figure 6.16: The particles after drawing from the prior, weighting, normalization and re-sampling. The particles shown have all the coordinates of the $\boldsymbol{x} = \begin{bmatrix} E & N & \psi \end{bmatrix}^{\mathsf{T}}$, but heading is for simplicity not displayed.

to form a very narrow distribution. So a small deviation in measurements gives a large deviation in likelihood. This also explains why there are only a few particles in Figure 6.16 that are not approximately zero after normalization. The artifact of only a few particles having relatively high weight can also be explained as a combination of artifacts of the sensor model, as mentioned in Section 6.2. In that case, a better importance density might be very difficult to find, which will be discussed in the future chapters Chapter 7 and Chapter 8.

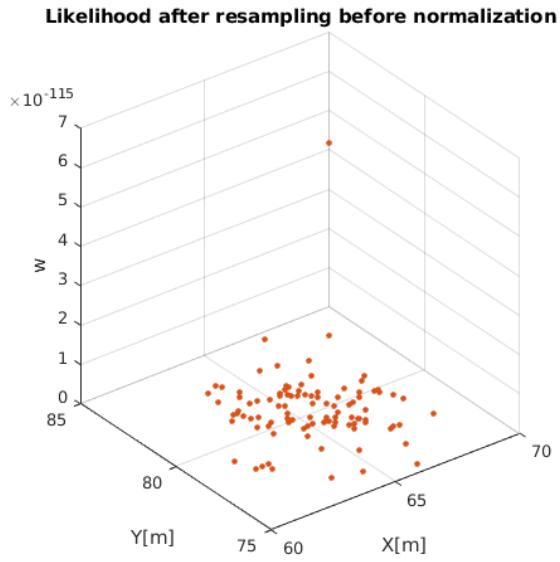


Figure 6.17: Likelihood of different particles after weighting.

6.3.2 Normalization and effective sample size

The normalization is done using the logarithmic weights as described in Section 5.6.1. And the effective sample size is calculated as described in Section 5.6.2. The threshold N_T is chosen to satisfy $\ln(N_T) = 0.01$, which is very small. This is due to the small empirical covariance of the particles.

6.3.3 Re-sampling

One problem in this application is particle impoverishment. Figure 6.16 shows a typical case where only one particle represent most of the distribution. The basic low variance sampling will often only sample this one particle.

The regularized particle filter can be used to partly solve the problem of particle impoverishment, see Section 5.5.4 for theory about the regularized particle filter.

In this implementation, the Gaussian kernel is used and the result is shown in Figure 6.16 after re-sampling. The figure shows that after re-sampling, there is not only one particle, but rather that all particle are distributed around the one sampled particle. But the illustration also shows a problem in this case, where the empirical covariance is small: the regularized re-sampling barely spreads the particles. The regularization does not seem to help much in this case, and the case of low process noise where it is more effective is described in Section 6.1.4.

6.3.4 Point estimation

The method of point estimation used in this implementation is MMSE and is explained in Section 5.6.5. It was chosen because of its simplicity and used in Figure 6.15 to find the estimated trajectory.

6.4 Evaluating the simulations

This section describes how well the particle filter behaves using different values of parameters. To evaluate the simulations, the consistency is tested using ANEES as described in Section 5.7.1. Finding the empirical covariance using (5.13) and estimating the position using (5.12) gave the ANEES as shown in Figure 6.18. Notice that the y-axis is in logarithmic scale, hence these are big numbers. The ANEES seems to not be inside the confidence interval at all. This indicates that the model is not correct. It is likely due to the low empirical covariance resulting from only one particle being very high compared to the others, which makes the particles not represent the distribution well enough. The RMSE is calculated using (5.15).

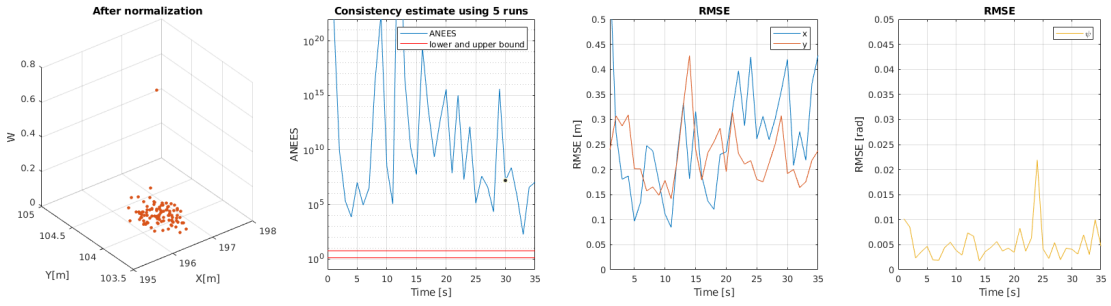


Figure 6.18: Data gathered over 5 simulations using $N = 100$ and $N_m = 100$. The lower and upper bound are the bounds of a 95% confidence interval.

Some things that can be done to get better consistency are

- Use more particles N: This makes the particles able to cover and represent the distribution better.
- Use more measurements N_m: Gives more precise point estimation and makes the method more robust to noise on each measurement.
- Reduce time step Δ_k: If the time step is smaller, the vessel cannot move as much from the last state, meaning less effect of the dynamics uncertainty. The time step is bounded below by the inverse of the frequency of the sensor.

Using more particles can give a better representations of the true state probability density. Using $N = 10000$ particles instead gives Figure 6.19. The particles might represent the true distribution better, but it is still not represented well. On the other hand, both the ANEES and the RMSE seem to be reduced. The consistency is still not very good, as can be seen from noting that the y-axis is still logarithmic scale. The ANEES is often inside the confidence interval, but peaks far above the upper bound several times.

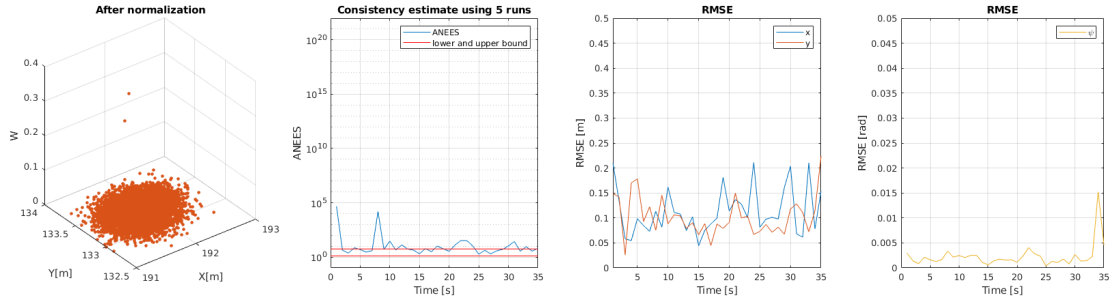
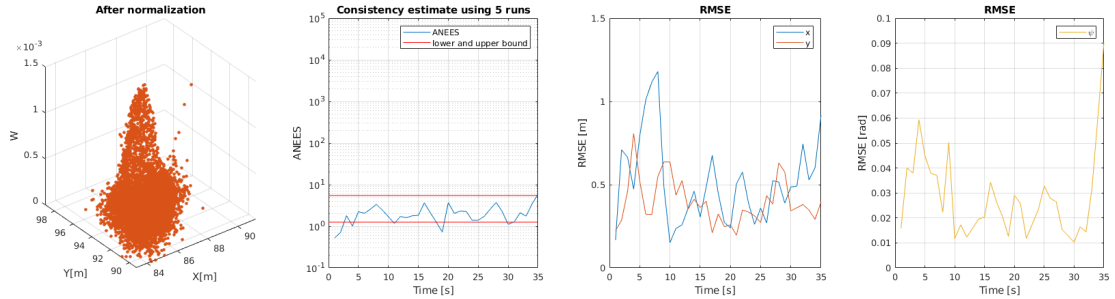


Figure 6.19: Data gathered over 5 simulations using $N = 10000$ and $N_m = 100$. The lower and upper bound are the bounds of a 95% confidence interval. The x- and y-axis for the particles plot are set to the same scale as in Figure 6.18.

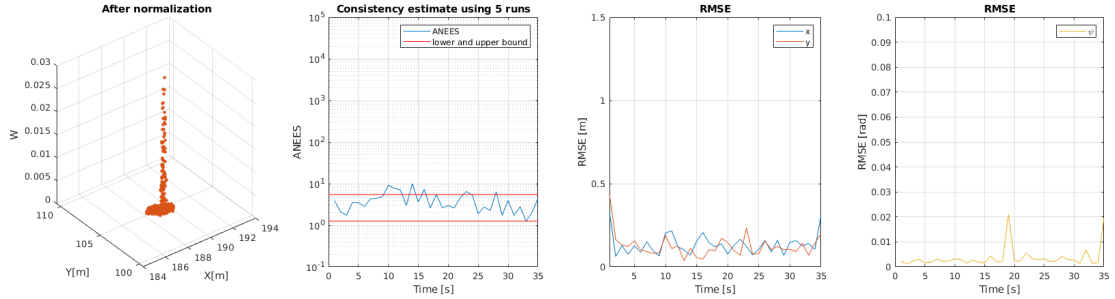
Reducing the amount of measurements used can have a big impact on the consistency of the filter. Figure 6.20 shows the filter being used with $N = 10000$ particles but different amounts of measurements N_m. It shows that increasing the amount of measurements decreases the RMSE. But more measurements also make the true state distribution more narrow and increases the ANEES. The case in Figure 6.20c is similar to the incredibly narrow case in Figure 6.19 but with twice as many measurements, giving a generally higher ANEES. In the case of a Gaussian distribution having a standard deviation σ, the case of N_m such Gaussians multiplied together will have a standard deviation of σ_{N_m}:

$$\frac{1}{\sigma_{N_m}^2} = N_m \frac{1}{\sigma^2} \quad \overset{\sigma, \sigma_{N_m} > 0}{\Longrightarrow} \quad \sigma_{N_m} = \frac{\sigma}{\sqrt{N_m}}.$$

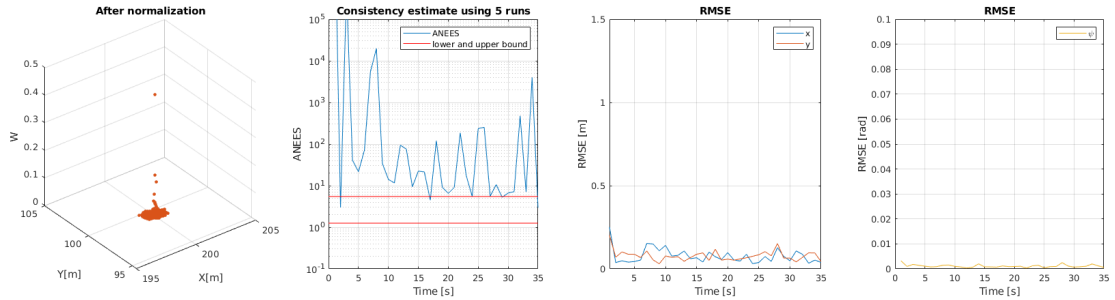
This explains why the particles seem to form a more narrow distribution as N_m increases in the figure.



(a) $N_m = 3$



(b) $N_m = 50$



(c) $N_m = 200$.

Figure 6.20: Data gathered over 5 simulations using $N = 10000$ particles and different number of measurements N_m. The lower and upper bounds are the bounds of a 95% confidence interval. The x- and y-axis for the particles plot are set to the same scale. Different scales on the z-axes of particle plots is set to be able to see the shapes.

An other way of increasing consistency is reducing the time step Δ_k. This reduces the time the vessel is allowed to move between iterations and therefore reduces the noise on the process model. For a real-time application, it means making the code run with higher iteration frequency. That is the opposite of what will happen when increasing either the number of particles or the number of measurements. The running times of the above mentioned simulations are shown in Table 6.1. These numbers might be deemed too big for real-time applications, arguing that $N = 10000$ particles is too many. Only the simulations with $N = 100$ particles had an iteration time of less than $\Delta_k = 1\,\text{s}$. But this one had bad consistency properties. By decreasing Δ_k, the noise on the process is reduced and the particles are less spread after sampling the prior. Making the method run faster is generally a good idea if possible also because any zero-mean noise will be averaged over time and it makes less happen between different time steps. It is also worth mentioning that doing the simulations in Matlab might make them slower than a C++ implementation.

N	N_m	**Time** [s]	**Time** $[\frac{\text{s}}{\text{iteration}}]$
100	100	92	$92/(35 \cdot 5) = 0.53$
10000	3	3449	$3449/(35 \cdot 5) = 19.7$
10000	50	6306	$6306/(35 \cdot 5) = 36.0$
10000	100	8766	$8766/(35 \cdot 5) = 50.1$
10000	200	14030	$14030/(35 \cdot 5) = 80.2$

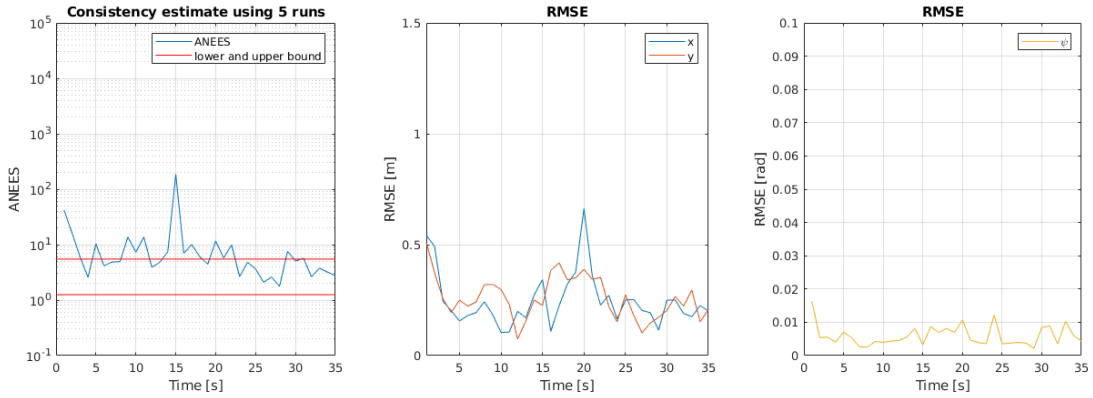
Table 6.1: The time necessary for 5 simulation runs for different choices of parameters. Each simulation run had 35 time steps. N is the number of particles and N_m is the number of measurements.

The threshold N_T for when to re-sample the particles depends on the amount of particles. It should be tuned for each set of parameters, but is for simplicity in the evaluation set to $0.15N$, where N is the number of particles used. This results in not having to re-sample too often for $N_m = 3$, but ends up re-sampling every iteration of the particle filter for more narrow distributions. It might not be a problem in the case of using a better importance density.

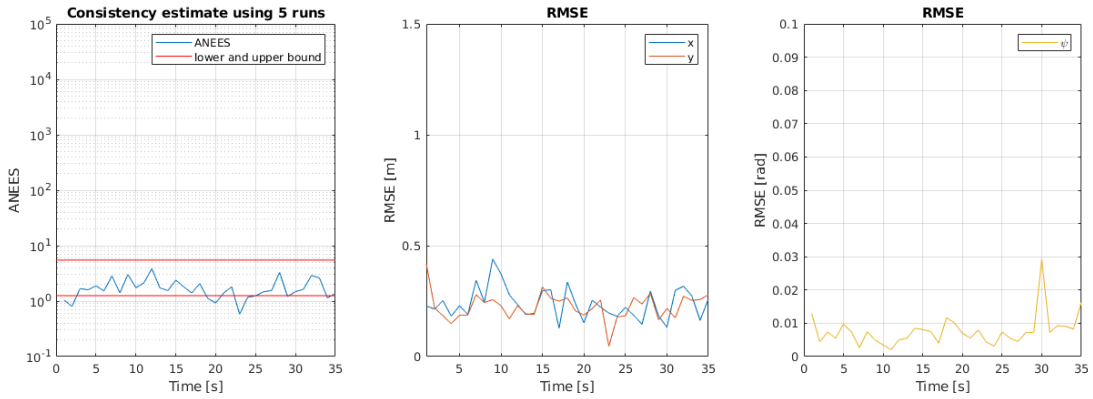
The consistency problem can also be dealt with using a better importance density. If the sampling of points is done using a better importance density, the points are not spread as much, they will cover the relevant area better by not having many points in low probability areas and the amount of particles used does not have to be as high. This would for the particles in Figure 6.20b mean less particles with low weights and more particles representing the core of the distribution, the higher weighted ones in the figure. But the sensor model used here is not linearly dependant on the state, and an optimal importance density, as described in Section 5.5.2, might not be as easy to find.

6.5 Increasing the uncertainty

One way of getting better estimates is increasing either the variance σ^2_{hit} or the weight for the uniform noise α_{rand}. Increasing α_{rand} has been done in Figure 6.21. It shows both a decrease in the RMSE and also a decrease in ANEES.



(a) $\alpha_{\text{rand}} = 0.1$



(b) $\alpha_{\text{rand}} = 0.9$

Figure 6.21: Data gathered over 5 simulations using $N = 1000$ particles $N_m = 20$ range measurements. The lower and upper bounds are the bounds of a 95% confidence interval. Note that one has a logarithmic y-axis for the ANEES.

Chapter 7

UKF-based importance density

For many problems it can be difficult to find the optimal importance density in the PF as described in Section 5.5.2. But it can still be possible to approximate it, for example using an EKF as done in [24]. Using the same sensor model as in Chapter 6, the measurement model is not only nonlinear, but can be difficult to linearize. This is because of discontinuities in the relation between measurements and the state, that is the map. Therefore, the EKF is not well suited for this case. Instead, the UKF [29] can be used as described in this chapter, resulting in the UPF [64]. The methods are explained in detail in Section 5.2 and Section 5.8 respectively.

7.1 UPF method

The simulations are running using simplified versions of the models used in Chapter 6. This has been done to remove uncertainties and make the uncertainties more Gaussian in the hope that it simplifies the tuning of the UKF. More precisely, the measurements now follow instead of (4.9)

$$\boldsymbol{z}_k = \boldsymbol{h}\left(\boldsymbol{x}_k, \boldsymbol{n}_k\right) = \boldsymbol{h}_r\left(\boldsymbol{x}_k\right) + \boldsymbol{n}_k,$$

where the uniform noise has been removed. σ_{hit}^2 is the same as in Section 4.6. This is because the UKF uses models where the noise covariance needs to be specified explicitly. It does not conform with the sensor model containing a combination of a uniform and a Gaussian distribution used earlier. But it is only needed in the UKF and not the rest of the particle filter. For simplicity of tuning, the map used to make both measurements and expected values are the same. The process model is the same as in (4.4).

The UKF implementation used was found online at [30]. The UKF was difficult to tune due to the computational burden, see an explanation of the parameters in

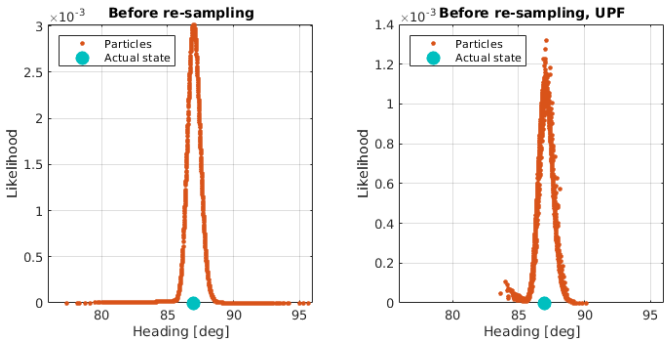
Section 5.2. Some choices of parameters are shown in [25, p. 463], explaining that $\beta = 2$ is a good value for Gaussian distributions and $\kappa = 0$ is typical. The initial attempt $\alpha = 10^{-3}$ was probably a very narrow spread of sigma points because it gave a lot of track loss. Using $\alpha = 1$ instead worked a lot better. The question remains if those are the optimal choice of parameters, which we will come back to in the next section.

7.2 UPF properties

This section looks at how the UPF works by using the simplified map made earlier, see Figure 6.1, similar to Section 6.2. Most figures are made by using the SIR filter until something interesting happens, and the UKF is then used and the results discussed. These properties will be used further to discuss the behavior of the UPF in the three DOF case.

7.2.1 Gaussian distribution

The UPF works as expected for the perfectly Gaussian case, see Figure 7.1. Here, the UKF seems to find the core of the distribution well, and the particles becomes less spread than in the SIR filter. This makes the actual state distribution represented better and might enable a reduction in the amount of particles used.



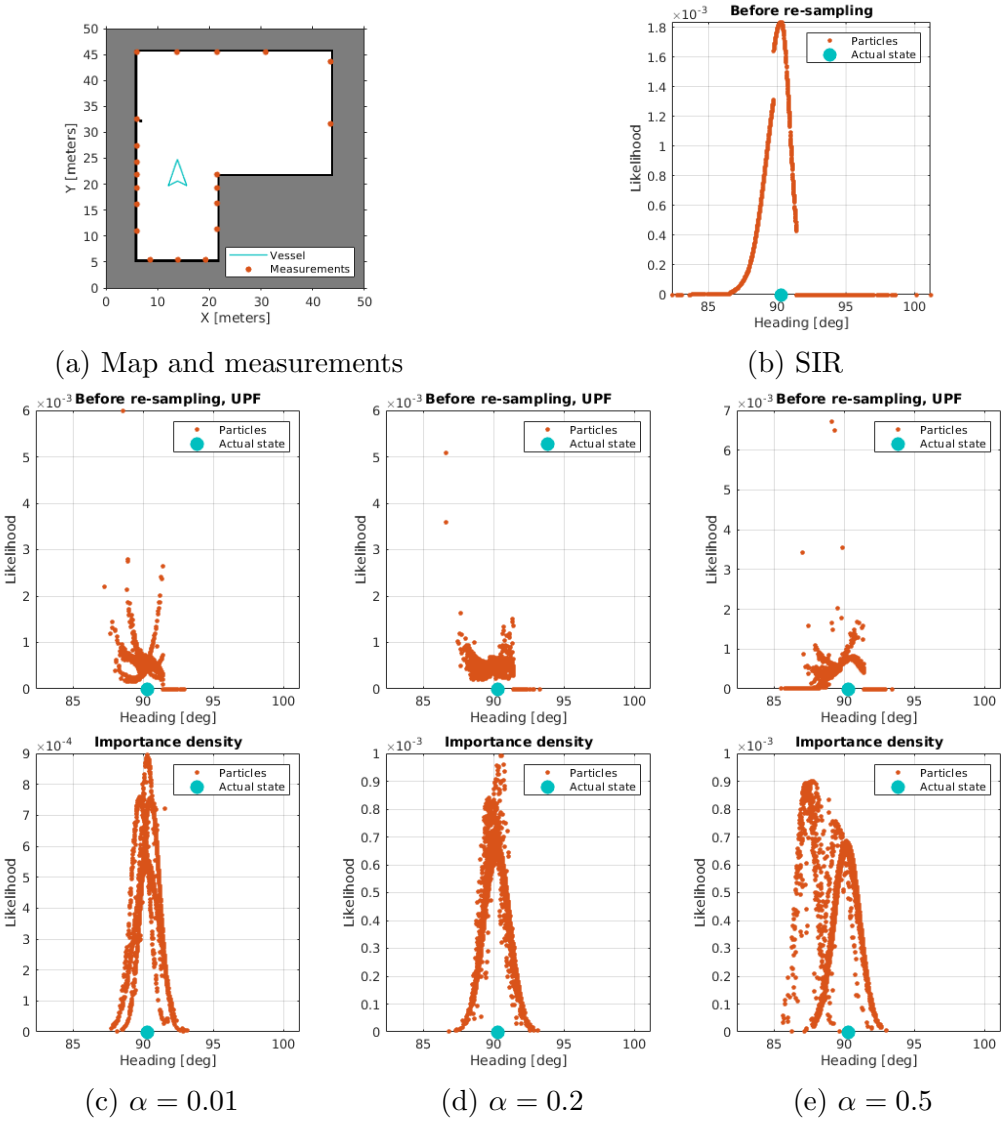
(a) SIR (b) UPF

Figure 7.1: Using UKF to make the importance density.

7.2.2 Tuning

Assuming a Gaussian distribution, the one parameter to tune with regards to the UKF is α. This has been done for a specific scenario in Figure 7.2. It is difficult

to know exactly what the distribution should look like, but it should not value the actual state very low. The best Gaussian importance density is assumed to be similar to the distribution from the SIR filter. Too small α seems to not spread the particles enough to see the correct distribution, giving many different possibilities due to the complexity of the map. And too big an α seems to spread the particles so much that there becomes several different importance densities a big distance from each other. $\alpha = 1$ was also attempted and moved the left Gaussian in Figure 7.2e farther to the left. As we will see later, the best value of α depends on the situation.



(a) Map and measurements (b) SIR

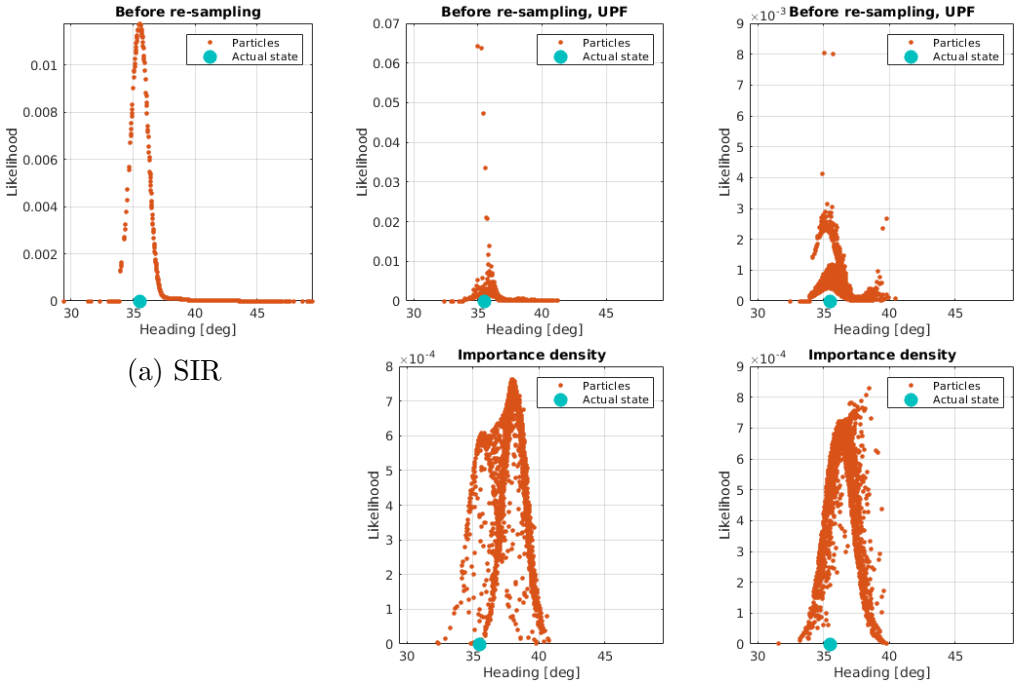
(c) $\alpha = 0.01$ (d) $\alpha = 0.2$ (e) $\alpha = 0.5$

Figure 7.2: How different values of α affect the distribution and the importance density.

It is important to remember the assumption of a Gaussian. As mentioned in [64], the UKF has "[...]the limitation that it does not apply to general non-Gaussian distributions." On the other hand, this does not mean it cannot be a good approximation.

7.2.3 Bad importance density

Figure 7.3 shows how the UPF can make a better importance density than the SIR filter. The SIR filter has relatively few particles in the most important area, where the UPF has placed all its particles. Contrary to the tuning before, here it seems that $\alpha = 0.5$ is better than $\alpha = 0.2$. It therefore seems that the best tuning depends on the situation, which is expected but still not good for this state estimation method.



(a) SIR

(b) $\alpha = 0.2$

(c) $\alpha = 0.5$

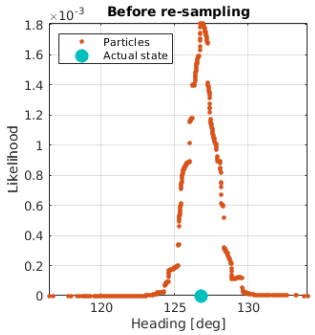
Figure 7.3: Better importance density than SIR.

7.2.4 Diagonal lines

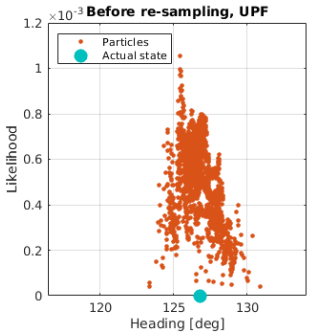
The UPF does not seem to react well to diagonal lines, see Figure 7.4. It does spread the particles less than the SIR and in the correct region. But the weighting is difficult to understand. It might be a good distribution, but it would be more intuitive to have something similar to the core of the SIR distribution. This can be done by using only $p(\boldsymbol{z} \mid \boldsymbol{x})$ in the weighting instead of (5.8).

7.2.5 Corners

The simple map in Figure 6.1 has five outwards pointing corners and one inward pointing corner. Figure 7.5 shows that the UPF can in some cases approximate the



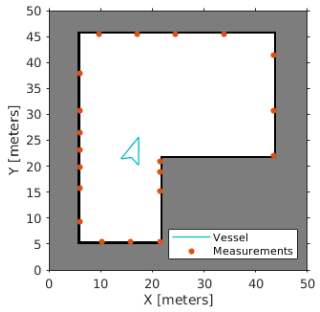
(a) SIR



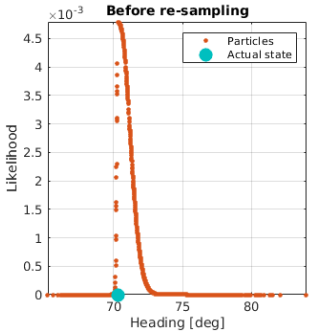
(b) UPF

Figure 7.4: How the UPF reacts to diagonal lines.

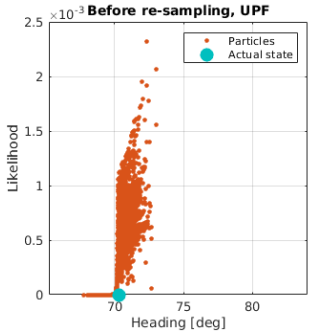
optimal importance density even though the measurements are not pure Gaussians. We can notice that the importance density resembles what the SIR distribution would have looked like if the right and down wall had been all the way to the right.



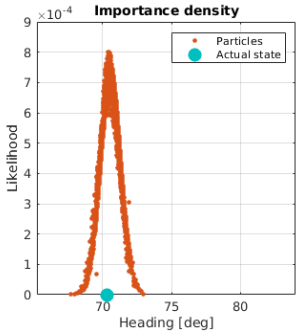
(a) Map and measurements



(b) SIR



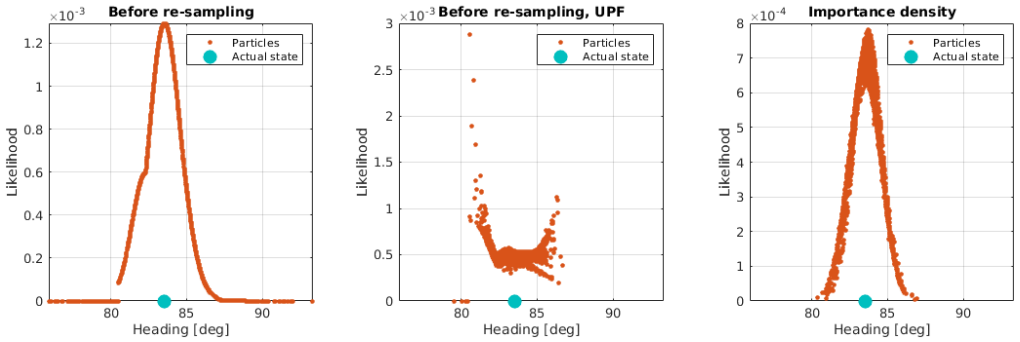
(c) UPF



(d) Importance density

Figure 7.5: Inward corner affecting the UPF distribution.

Also an outward pointing corner can affect the distribution, shown in Figure 7.6. The UPF seems to handle this well. The figure also shows what the importance density does in the weighting by (5.8). Where the importance density is low, the resulting weight becomes big, as we see from many particles in the UPF distribution having high values when it has low values in the SIR distribution. This phenomenon can be viewed on some of the earlier figures, but is very clear in this case.



(a) SIR (b) UPF (c) Importance density

Figure 7.6: Outwards pointing corner affecting the UPF distribution.

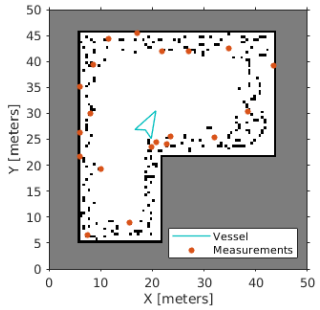
7.2.6 Noise

Many different types of maps get similar distributions as the ones already described. But an important artifact is shown in Figure 7.7 where the map once again has been made to give big changes in range measurements from small changes in state. The states of the highly weighted particles in the SIR filter are now the ones we want the UPF to represent. The figure shows how this depends on the tuning. For the case of $\alpha = 0.2$ the UKF placed all particles in regions that are away from the actual state. This is unfortunate and might make the UPF diverge. Using $\alpha = 0.5$ is better, while increasing α even more can spread the particles more than necessary.

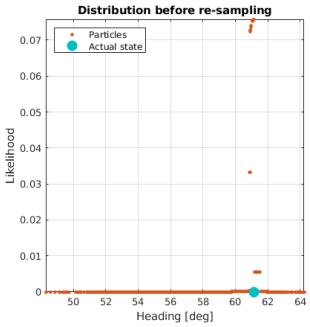
7.2.7 Using the filter map

Until now, the SIR filter has been run first in order to find the interesting artifacts, and then the particles from before last iteration has been used in the UKF to make the importance density. Only a single iteration of the UPF has been run. The UPF will now run by itself.

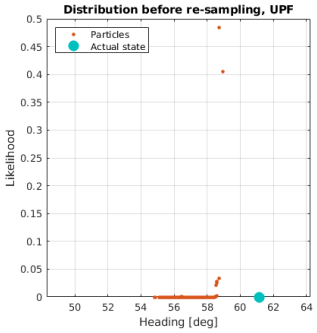
Using the UPF in the filter map, described in Section 4.6, seemed to not give any new insight. Running the UPF on its own can give results like in Figure 7.8. The figure shows that the particles have been spread in a small and correct area as desired. But the particles do not seem to represent any intuitive distribution. When looking closer, the zoomed in figure shows similarities with the SIR weights. It is difficult to say what weighting is best and what the distribution should look like. But the filter seems to not spread the particles too much using UPF and it should be optimal under the assumptions used.



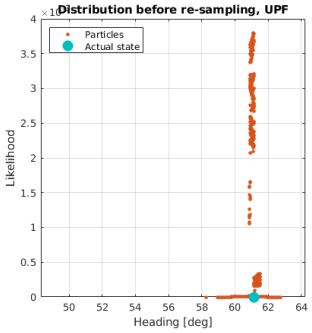
(a) Map and measurements



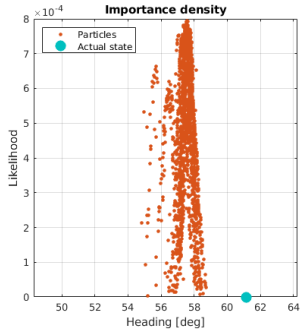
(b) SIR

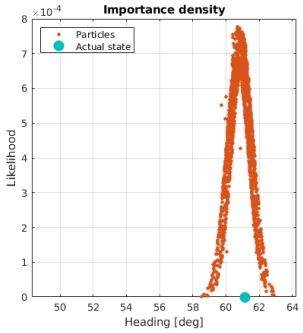


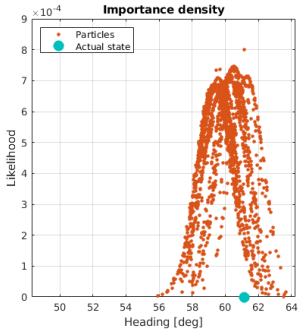
(c) $\alpha = 0.2$



(d) $\alpha = 0.5$

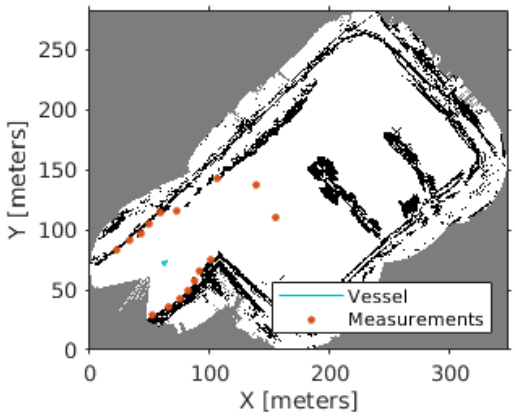




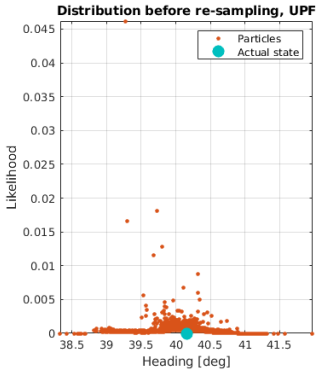


(e) $\alpha = 1$

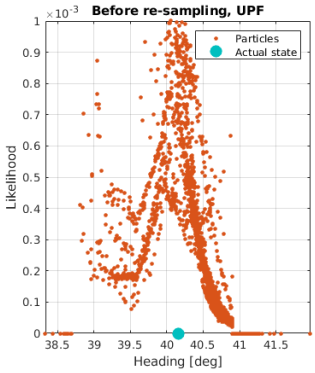
Figure 7.7: How a map with many discontinuities affects the UPF distribution. α is a tuning parameter of the UKF.



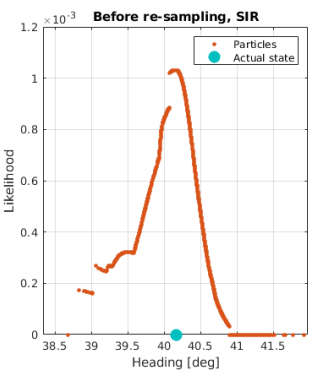
(a) Map and measurements



(b) UPF



(c) UPF zoomed in

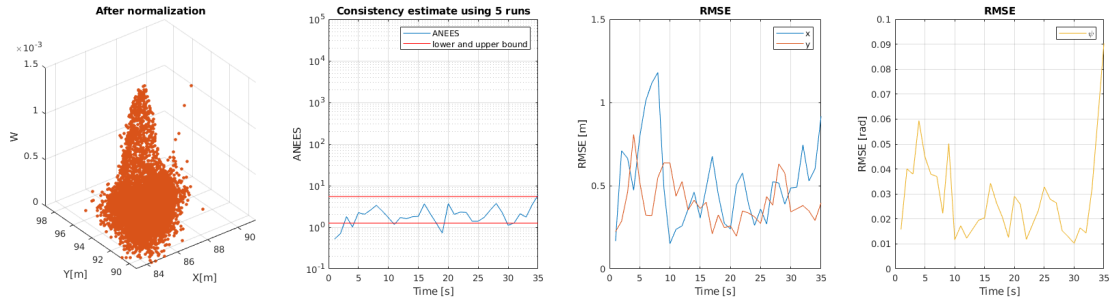


(d) SIR

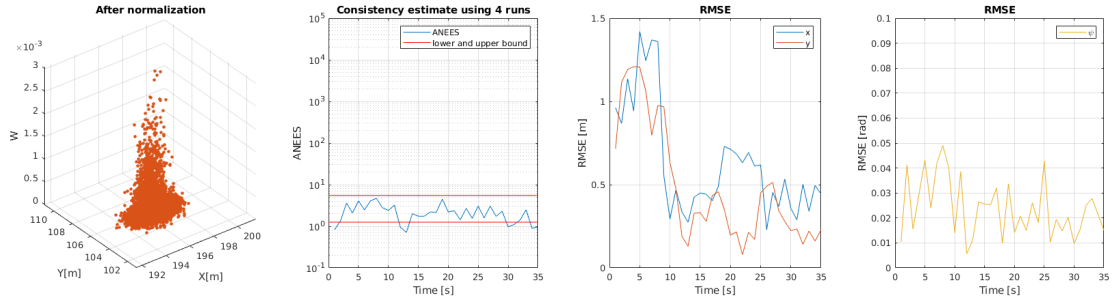
Figure 7.8: The distribution during a realistic run. The SIR weights are using the particles after sampling from the UKF importance densities, it is just a different weighting.

7.3 Evaluation

This section evaluates how well the UPF works for the full three DOF. The UPF was run $N_{\text{runs}} = 4$ times on the full 3 DOF state and the resulting data is shown in Figure 7.9b. It shows that the UPF can at least give similar results as with the earlier particle filter, from now on called the generic particle filter, in Figure 7.9a. But neither the RMSE or the ANEES seems to be significantly better, and the distribution might even look like a worse estimate without a clear peak. The sampled particles are spread a bit less, which is easier to see in Figure 7.10, but not a lot. This might become better with better tuning of the UKF. The big problem is that to run only the $N_{\text{runs}} = 4$ runs, with 35 time steps each, took approximately 15 hours as shown in Table 7.1, making it difficult to tune the UKF. Using $N = 1000$ particles for tuning was tested instead, but as well as still taking a lot of time, it was difficult to see differences in distributions with such few particles, see Figure 7.10.



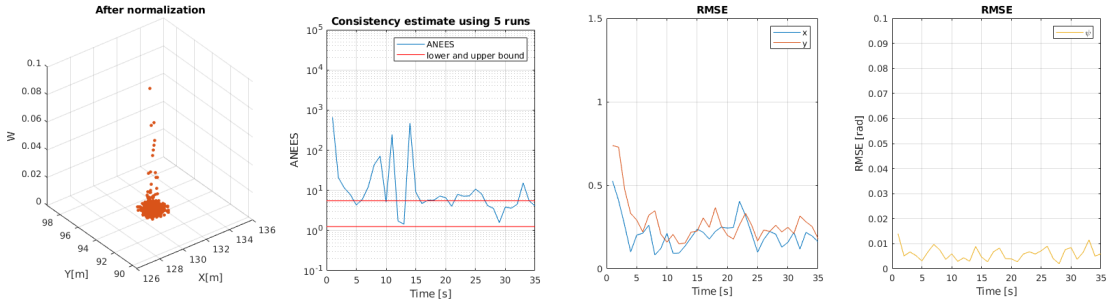
(a) Generic particle filter. Uses $N_{\text{runs}} = 5$. Figure 6.20a revisited.



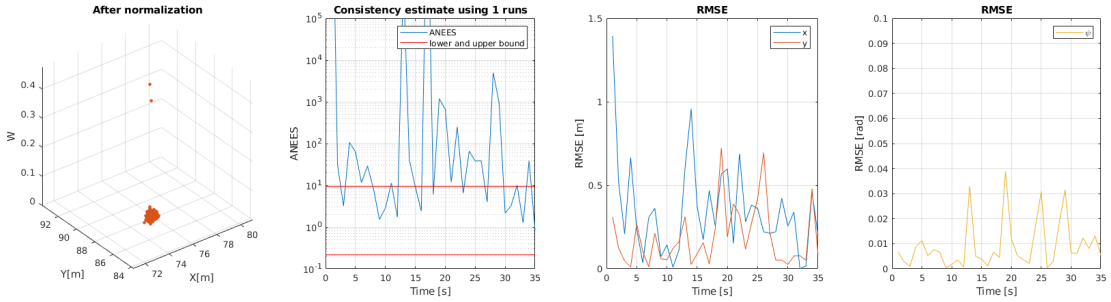
(b) UPF. Uses $N_{\text{runs}} = 4$.

Figure 7.9: Illustrates the difference in distribution, ANEES and RMSE for the use of the generic particle filter and the new UPF using $N = 10000$ particles and $N_m = 3$ rays of measurements. The difference in lower and upper bounds of the 95% confidence interval comes from the difference in number of runs. The scales have been set equal.

Some run times of different simulations are shown in Table 7.1. The table shows that the run times of the UPF is even bigger than for the generic particle filter. The long run times are due to the methods used having to loop over many different states. The UPF has to for each time step iterate over



(a) Generic particle filter. Uses $N_{\text{runs}} = 5$.



(b) UPF. Uses $N_{\text{runs}} = 1$.

Figure 7.10: Illustration of the difference in distribution, ANEES and RMSE when using the generic particle filter and the UPF. Parameters used are $N = 1000$ particles and $N_m = 20$ range measurements. The scales have been set equal. The differences in lower and upper bound of the 95% confidence interval is due to the difference in number of runs N_{runs}.

- N particles. For each particle
- $2(n_x + n_\nu + n_n) + 1$ sigma points, see (5.2). For each sigma point
- calculate N_m measurements using ray tracing.

Notice that $n_n = N_m$ here. There is also a lot more happening, but this gives the time complexity of $\mathcal{O}(NN_m^2)$. Using many measurements therefore makes the UKF computationally demanding. This is the case since the sensor model is based on the physics of the sensor, giving a lot of measurements and doing a lot of computation to find the expected measurements. This argues that a more efficient sensor model would be one using fewer measurements, for example using features from the measured points or doing scan matching.

Which PF	N	N_m	N_{runs}	**Time**[s]	**Time**$\left[\frac{\text{s}}{\text{iteration}}\right]$	
Generic PF	10000	3	5	3449	19.7	From Table 6.1
UPF	10000	3	4	52984	378	
Generic PF	1000	20	5	487	2.8	
UPF	1000	20	1	4654	133	

Table 7.1: The run times of different particle filters with different number of particles N, number of measurements used N_m and number of runs. The time per iteration of the particle filter, that is one time step, is shown. The number of time steps for each run is 35. The time per iteration is calculated as $t/(35 \cdot N_{\text{runs}})$ where t is the time a simulation used.

7.4 Beam model considerations

The UPF seems to be a possible solution to finding a better importance density. But actually finding it in this case has been made very difficult due to the already big computational demand of the particle filter and the sensor model used. The reason for finding a better importance density was to not spread the particles in areas of low likelihood, enabling the use of less particles and use more measurements. This seems to not be a solution because of the big increase in computation when increasing the number of measurements.

One suggestion to make the UPF better is reducing the amount of measurements only for the UKF importance density estimation, but keep using the same amount for the rest of the PF. This would decrease the run time of the method, but it would not necessarily make the estimate better, probably the opposite. It would make the estimates less robust as noise on one measurement will affect the total estimate more than earlier, arguing that a method for which rays to choose is important.

An other suggestion is increasing α_{rand} as done in Section 6.5. This is assumed to be approximately as good as done there, but not actually solving any of the problems discussed in this chapter. An other suggestion is analyzing every one of the 3 dimensions separately to try and figure out how the UPF behaves in the different dimensions.

The beam model is generally slow as it requires ray tracing for each measurement ray, and the lidar has a lot of rays. The particle filter is also known to be one of the slower estimation methods. The closest results to a good real time implementation are the results in Section 6.5. But it is important to remember the methods discussed so far are based on a sensor model where all measurements are evaluated independently. The next chapter tries to mitigate this by using the ICP method which does not handle measurement independently.

Chapter 8

ICP-based importance density

Instead of using every beam of a lidar scan independently, all rays can be used together in scan matching. A scan is a set of measurements taken in succession, often represented as a point cloud. Scan matching is generally the process of finding the rigid transformation of a scan that best aligns it with some other data. This data can have different types, for example a map or an other scan. If the map itself is a point cloud, or the matching is done to an other scan, the process is also called point set registration, where the transformation between two different point clouds is found that optimizes a certain metric. The rigid transformation found when a scan is matched to a map, given that the scan was made in that map, can be used as an estimate of the pose of the scan in that map, and therefore useful in localization. The ICP method, see Section 5.9, is a method that finds such a transform.

8.1 ICP convergence analysis

8.1.1 Point cloud from occupancy grid

The map in Figure 6.1 is used to analyze how well the ICP works. Making this occupancy grid into a point cloud can be done in many ways. Three ways will be described and a short analysis of the differences is made.

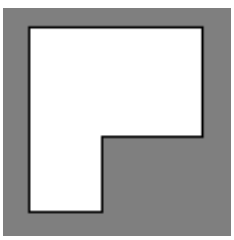


Figure 6.1 revisited.

A quick solution is to simply let the center of every occupied cell be a point and then do ICP with the resulting point cloud as the fixed set. This has been done in Figure 8.1 which shows how a quite big deviation after sampling a particle filter can converge close to the correct state. But it also shows that

the real measurements do not actually coincide with the map
points and can result in the estimated state being farther to the sides where there are more points, farther toward the bottom of the figure. Since the measurements are actually at the sides of the occupied cells in these simulations, placing points on the sides of the occupied cells might be advantageous. This will also enable the use of more points for better estimation. In these simulations there have been no noise on the measurements, but the states have been spread quite a bit to see what happens. Using the center as point gave the convergence of particles shown in Figure 8.2. We see that many particles get close to the actual state, while other seemingly have converged to other local minima. This is also the case for the other ways of creating point clouds, and ways to solve this will be discussed.

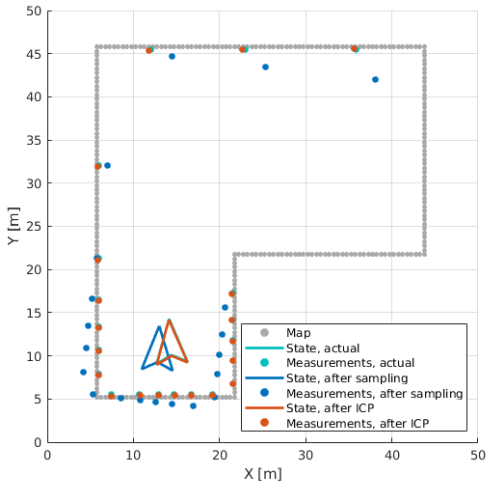
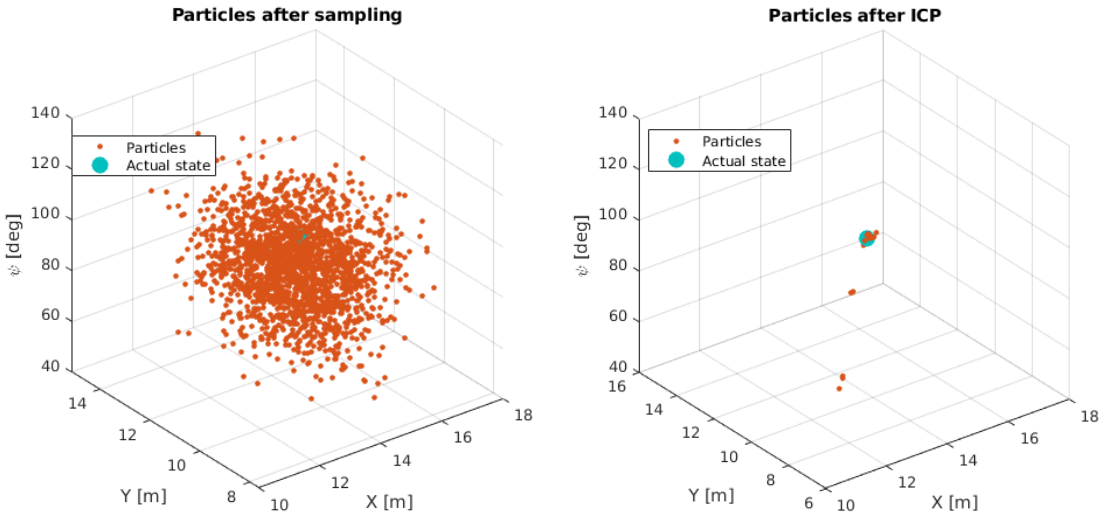


Figure 8.1: The center of occupied cells used as point cloud.

The three different ways of making a point cloud from an occupancy grid discussed here are shown in Figure 8.3 with states of different situations. The cases have been run on different realization of motion noise. Corresponding RMSE values are shown in Figure 8.4 and are given from the ICP implementation in Matlab. It seems the number of different local minima is visible in the RMSE graphs, where the RMSE seem to take one of several values depending on the particle. See for example mainly 3 local minima in Figure 8.2b seem to correspond to 3 different values of RMSE in Figure 8.4a, both figures made from the same run. All maps seem to have the problem of the type of minimum shown in Figure 8.3b. These give the highest RMSE values and are seemingly fixable by using more measurements or more points in the map, this tuning will be discussed later. Using points on each corner of each occupied cell as well as one on each side of each occupied cell gives the map in Figure 8.3a. That figure shows the case of a local minimum that is special for that map. Since the map is no longer thin, but has a thickness of a cell width, the ICP runs into local minima close to the actual state. This is also shown in the RMSE values in Figure 8.4b. More information can be used to solve this. By only placing points on the sides between occupied and free space, the map becomes the



(a) After sampling, that is propagating the particles with a motion model.

(b) After ICP.

Figure 8.2: States converging using ICP.

one shown in Figure 8.3c. This map type seems to give less local minima than the others and has a reduced RMSE. It is also tunable with the amount of points to place on the sides.

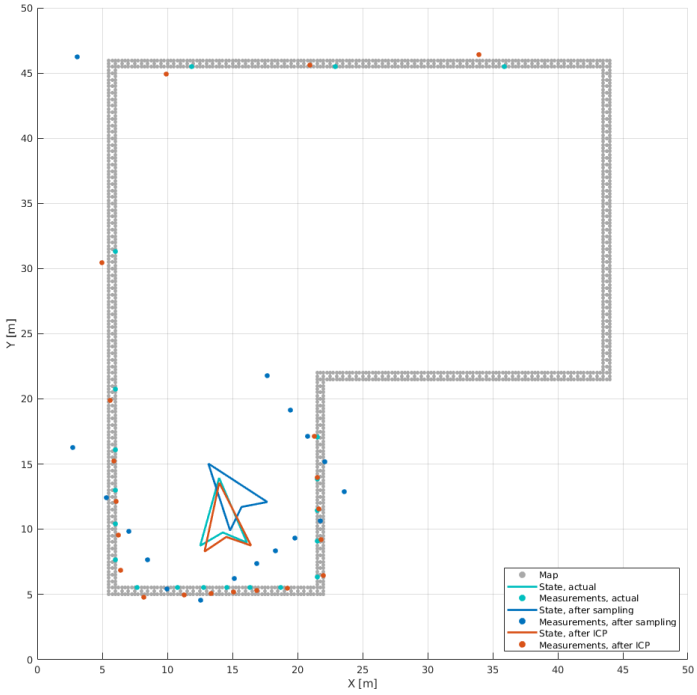
8.1.2 Tuning the ICP

Some tunable parameters in the simulation regarding ICP are:

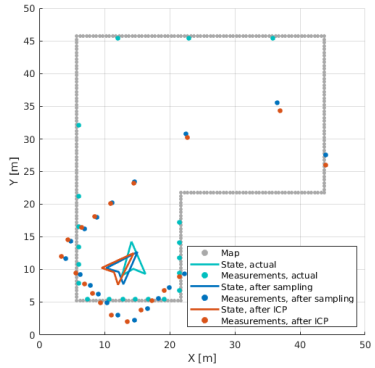
- The number of points the map consists of.
- The tolerance of the ICP method.
- The maximum number of iterations for the ICP method.
- The number of measurements used, N_m.
- The spread of the possible states or the noise of the motion model, $\mathbf{\Sigma}_\nu$.

An easy fix to the problem of the ICP converging to states far from the actual state is to simply not spread the particles a lot, but have all particles close to the true state. This is obviously not a solution, as the motion noise is in real life not tunable and is therefore disregarded here. But the motion noise is still set very high in order to analyze possible outcomes.

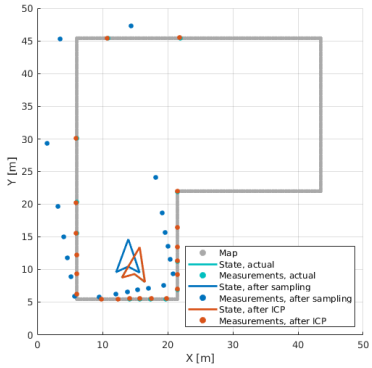
The local minima mentioned earlier and viewed in Figure 8.3b did not disappear by the use of more points in the map, lower ICP tolerance or increasing the number



(a) Points on corners and each side

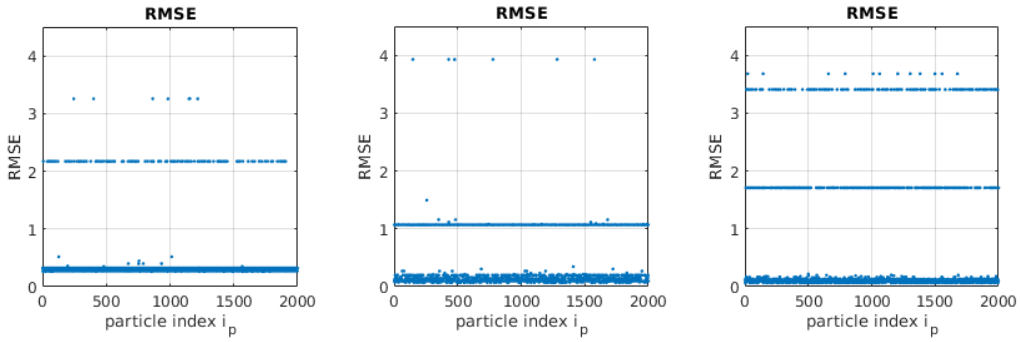


(b) Point at center



(c) Points along sides between occupied and free space

Figure 8.3: Different ways of creating point cloud from occupancy grid for each occupied cell. The figure also shows some different types of local minima and lastly how a good convergence should look.



(a) Point at center

(b) Points on corners and each side

(c) Points along sides between occupied and free space

Figure 8.4: The RMSE using different ways of creating point cloud from occupancy grid for each occupied cell.

of ICP iterations. But increasing the number of measurements from $N_m = 20$ to $N_m = 100$ helped and gave results in Figure 8.5. The maximum RMSE seems to have been reduced compared to earlier, and the particle with highest RMSE is shown in Figure 8.6a. The ICP method does not seem to give many groups of converged particles. This does not solve the problem, since different realizations can still give convergence toward approximately the same states as earlier, where the measurement point cloud is rotated approximately 45° relative to the actual cloud. But it does not happen as much when more measurements are used. Increasing the spread of the particle more makes more particles have that problem. But 45° is a big angle and the problem can be omitted by not artificially spreading the particles as much.

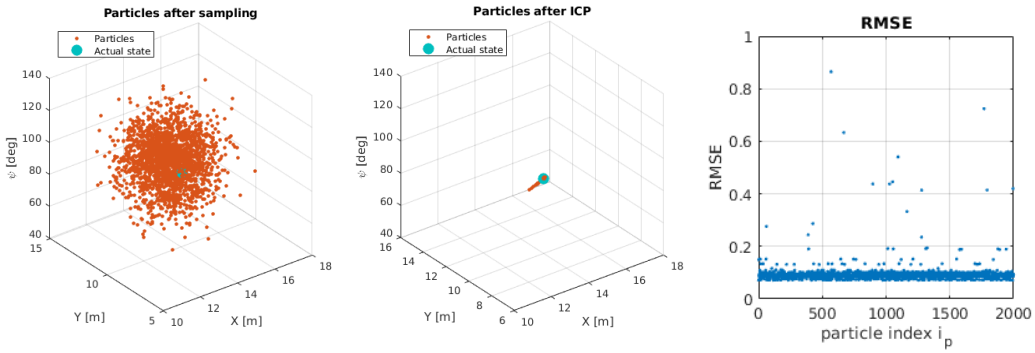
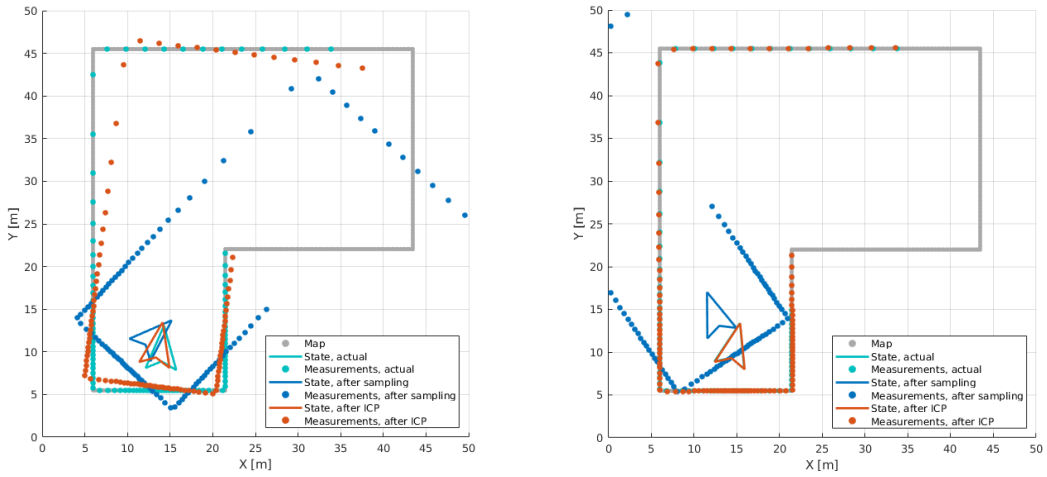


Figure 8.5: Particle convergence and RMSE when increasing the number of measurements.

A new problem occurs where the ICP gives a certain tail of particles close to the actual state. We see from the particle with highest RMSE, in Figure 8.6a, that it does not seem to be a problem of a local minimum, but rather that the ICP did not converge. By increasing the number of maximum ICP iterations, lets call it N_{iter}, from the default $N_{iter} = 20$ to $N_{iter} = 100$, this problem was removed and the



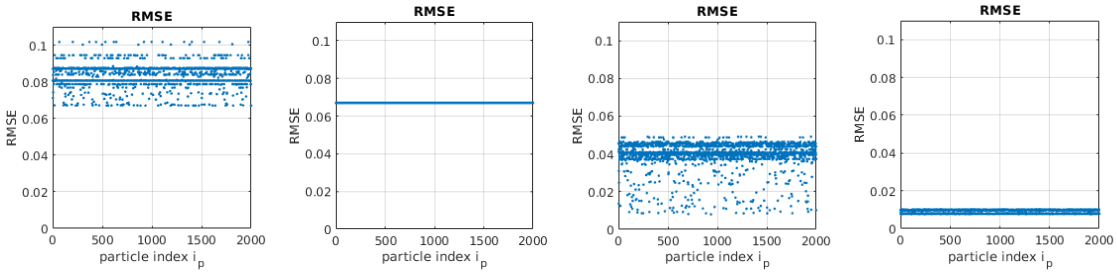
(a) Possible particle when increased N_m.

(b) Typical particle when increased maximum number of ICP iterations.

Figure 8.6: How increasing the number of maximum ICP iterations might affect the estimation.

particles with highest RMSE becomes similar to Figure 8.6b. The RMSE is now similar to Figure 8.7a.

The other parameters to tune enable a reduction in RMSE and more precise convergence. Let N_{map} be the number of points on each side of an occupied cell, $N_{map} = 3$ has been used earlier. The tolerance of the ICP method is a vector of two values, with Euclidean distance threshold for the translation and a rotation threshold in degrees respectively, the default is $(0.01, 0.05)$. The illustrations in Figure 8.7 use the same actual state, measurement and particles before the ICP is done. When the tolerance has been reduced in Figure 8.7b, the RMSE is also reduced and the particles become less spread. This is as expected given that the maximum number of ICP iterations N_{iter} is big enough. When N_{map} is increased in Figure 8.7c, the RMSE is also reduced, but more spread than when the tolerance is decreased. Combining these in Figure 8.7d gives a big reduction in RMSE, and the particles are all very close to the actual state. We can see from Figure 8.8 what the resulting map looks like and how big a transformation the method handles and still converges very well. Increasing N_{iter} did not seem to make any difference, arguing that it is big enough for all particles to converge within the mentioned tolerances. This tuning makes it seem like it is possible to get the estimates as accurate as desired, given that there are no local minima.



(a) $N_{map} = 3$ and the tolerance is $(0.01, 0.05)$.

(b) $N_{map} = 3$ and the tolerance is $(0.001, 0.005)$.

(c) $N_{map} = 20$ and the tolerance is $(0.01, 0.05)$.

(d) $N_{map} = 20$ and the tolerance is $(0.001, 0.005)$.

Figure 8.7: The RMSE using $N_m = 100$ and $N_{iter} = 100$.

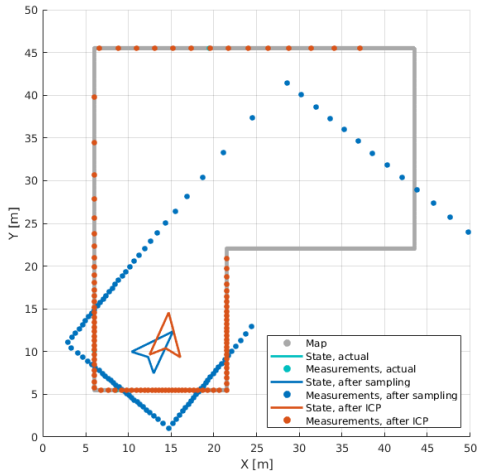


Figure 8.8: A typical convergence using $N_m = 100$, $N_{iter} = 100$, $N_{map} = 20$ and a tolerance of $(0.001, 0.005)$.

8.1.3 Different maps

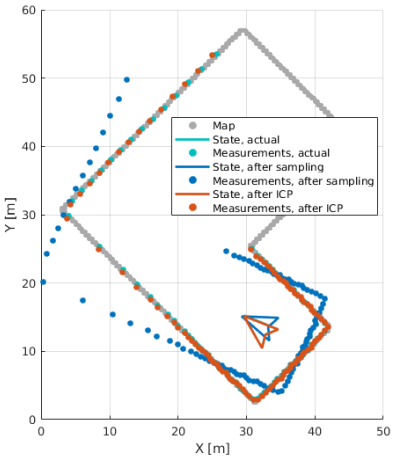
It is interesting to see how the ICP converges when using different maps. Similarly as seen in Figure 6.11, Figure 8.9 shows some different maps and a particle and convergence with relatively high RMSE value. All of these maps can give convergence to a state with heading that is approximately 45°. But it is in the figures only showed in Figure 8.10d where the top particles converged to approximately two different states with high heading error. It seems like the map in Figure 8.9d is the one that is most sensitive to this kind of local minimum. Figure 8.9b and Figure 8.9c gave RMSE and convergence that was similar to the results after the tuning in the simpler map. But looking closely at Figure 8.9a we can see that one of the particles with high RMSE values has converged to what looks like a local minimum, where the orange points are not overlapping the teal points. The same is true for Figure 8.9d. The local minima are also evident from Figure 8.10a and Figure 8.10d for the rotated and the noisy map respectively. They show a bigger RMSE value for the minima that do not coincide with the actual state. The existence of such local minima are expected and lies in the nature of the ICP method.

8.1.4 Measurement noise and realistic map

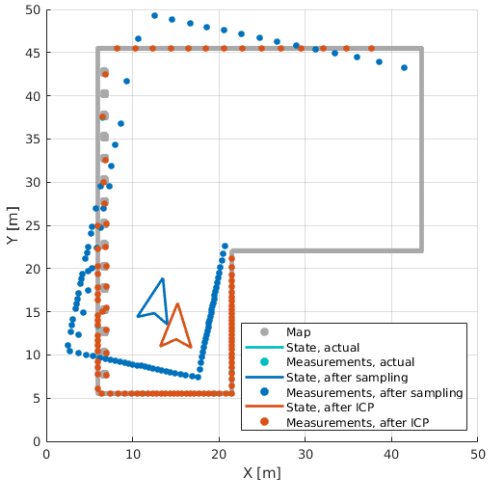
Adding Gaussian measurement noise, and not the uniform noise, with variance $\sigma^2_{hit} = 0.7$ as found in Section 4.6.1, gave the results shown in Figure 8.11. The state after ICP convergence is very similar for all particles, and all particles gave approximately the same RMSE which is high compared to when different maps were used in Figure 8.10. The higher RMSE is expected since it becomes impossible to align the measured point cloud perfectly to the map. Compared to the spread of the particles after sampling, the resulting error after convergence is small, not only in the figure but for more runs of the simulation. That said, there is a consistent error between the particles after convergence compared to the actual state, which is due to the fact that the actual state does not minimize the objective function of the ICP method.

The more realistic map in Figure 4.2 shows a lot of areas of free space that is not necessarily free. These areas make the resulting point cloud contain many more points than necessary. They are removed by making free space that is covered by occupied space into occupied space, see Figure 8.12.

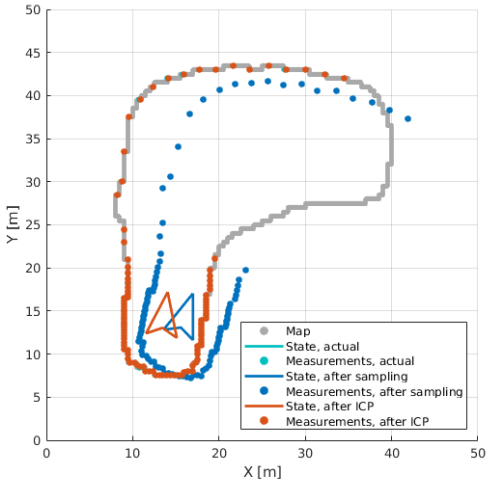
Figure 8.13 shows that local minima can occur for big errors in heading angle also for the realistic map. This can be seen from some particles having high RMSE values in Figure 8.13a which correspond to the particle far away from the actual state in Figure 8.13b. Zooming in, we see that the particles do not seem to converge towards the actual state, but gets an error because of the noise, as described earlier. In the zoomed in Figure 8.13b there is not just one state the particles have converged to,



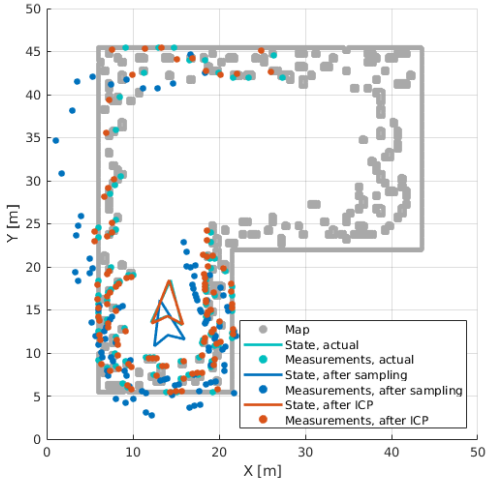
(a) Rotated map



(b) Structured noise map

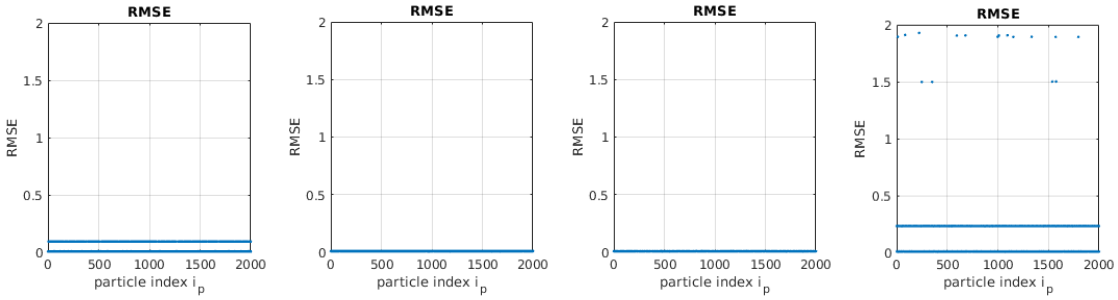


(c) Curved occupied space



(d) A lot of noise in the map

Figure 8.9: Different maps and some converged states with high RMSE values from Figure 8.10. These do not represent the type of local minimum discussed before, where the heading is approximately 45°.



(a) Rotated map (b) Structured noise map (c) Curved occupied space (d) A lot of noise in the map

Figure 8.10: The RMSE from one realization of ICP on $N = 2000$ particles using different maps.

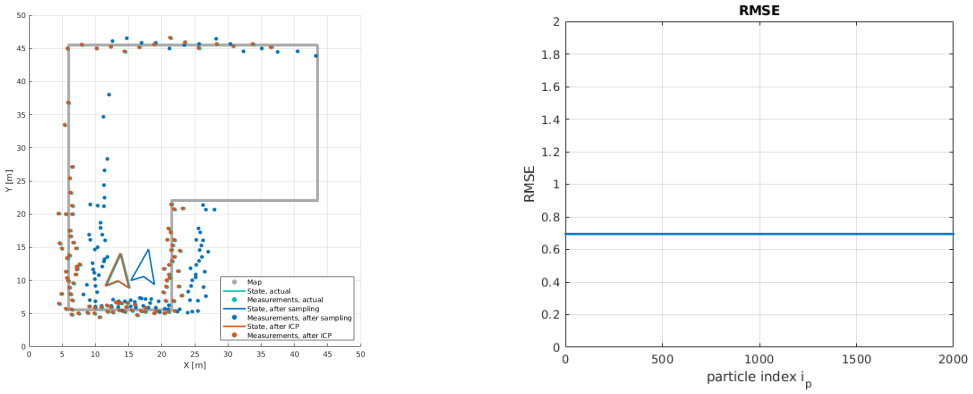
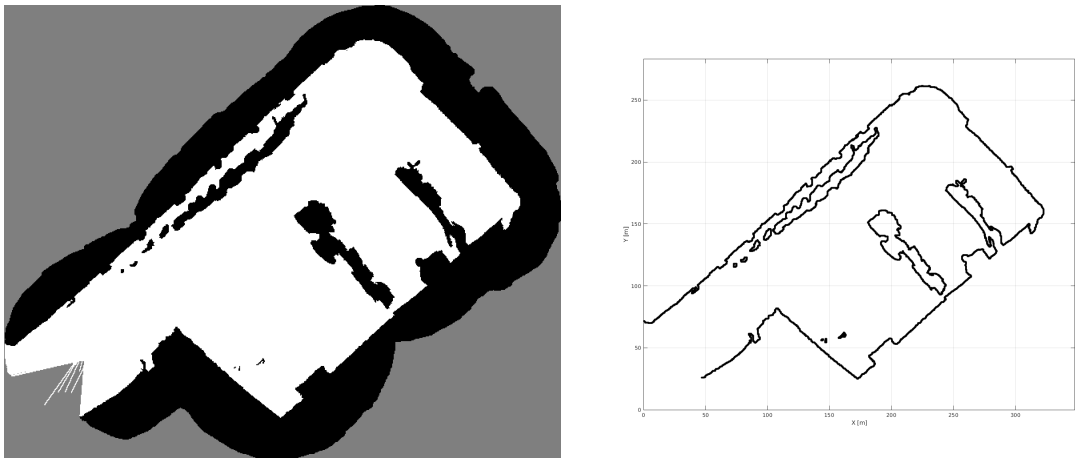


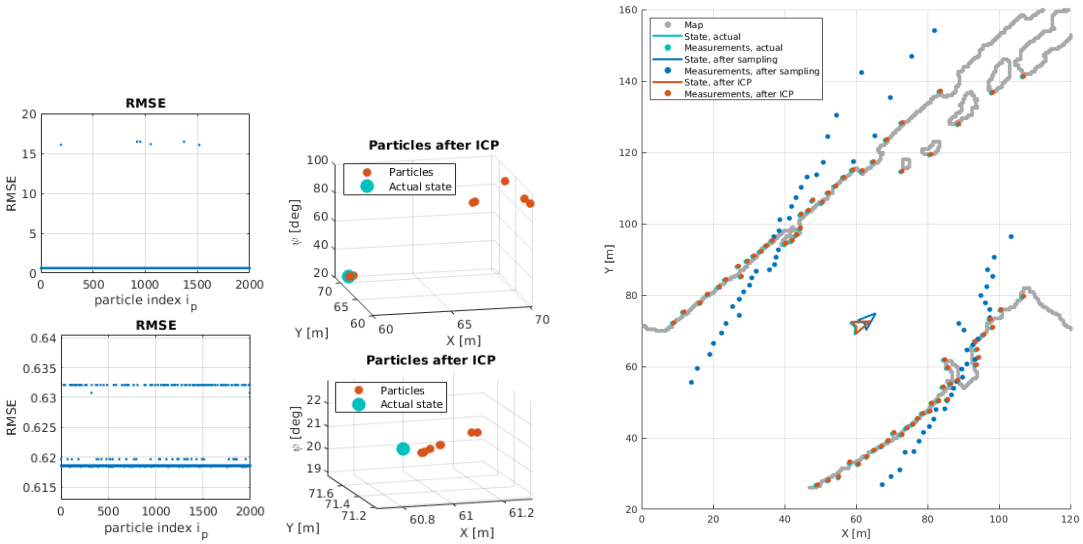
Figure 8.11: How the ICP converges with Gaussian measurement noise.



(a) Map as occupancy grid (b) Map as point cloud

Figure 8.12: The map in Figure 4.2 after removing free space that is covered by occupied space.

but several. This property needs to be taken into account when the ICP method is used further.



(a) RMSE zoomed in

(b) Particles zoomed in

(c) Map and a single ICP convergence

Figure 8.13: Using the realistic map.

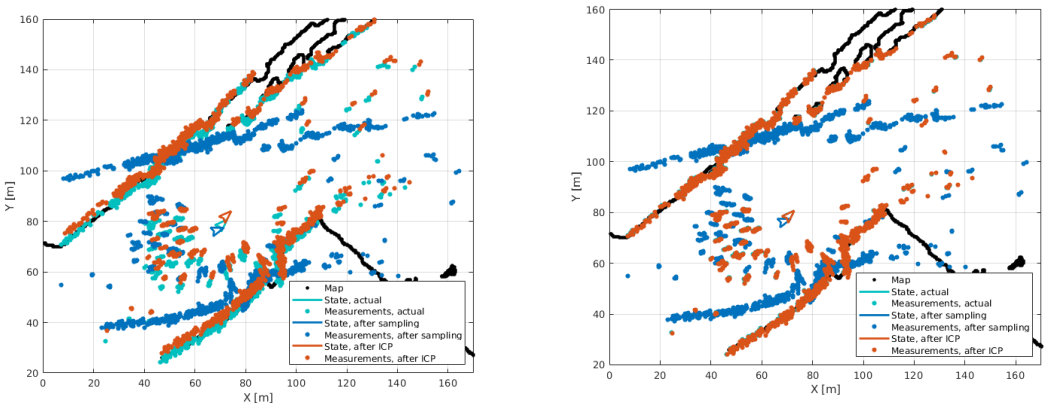
When also including approximate uniform noise, the ICP method consistently gives bad estimates as shown in Figure 8.14a. There we see many measurements being placed inside free space of the occupancy grid and far from the map point cloud. These measurements can make the ICP method converge far from the actual state. But the Matlab implementation used enables the use of outlier detection in the ICP, and the resulting convergence is seen in Figure 8.14b. Points that are far from the map point cloud are then treated as outliers for each iteration of the ICP. The resulting state is then consistently close to the actual state.

8.2 From ICP to importance density

One way of using ICP to make an importance density is by using the model

$$\boldsymbol{x}_k = \boldsymbol{f}_{\text{icp}}\left(\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{v}_{k-1}), \boldsymbol{z}_k, m\right) + \boldsymbol{n}_k^{icp}, \tag{8.1}$$

where $\boldsymbol{x}'_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{v}_{k-1})$ propagates the state through the process model, the function $\boldsymbol{x}_k^{icp} = \boldsymbol{f}_{\text{icp}}\left(\boldsymbol{x}'_k, \boldsymbol{z}_k, m\right)$ represents the ICP method using the measurements $\boldsymbol{z}_k$ and the map m and $\boldsymbol{n}_k^{icp}$ is noise. This means using the ICP method for each particle. But a lot of particles will end up in the same states or very close to one another after the use of the ICP method. That is why one can use less particles only



(a) ICP without outlier detection

(b) ICP with outlier detection

Figure 8.14: Illustration of how the ICP method is affected by outliers and how the use of outlier detection can fix the problem.

to find the importance density and then use all particles in the rest of the particle filter. A benefit from this is that the ICP method can use more measurements than the rest of the particle filter, giving better convergence, while not increasing the run time as much.

The noise $\boldsymbol{n}_k^{icp}$ in (8.1) depends on many factors, such as how many measurements N_m are used and also the noise on both the process and the measurement. It has been estimated for the simulations in Figure 8.15. From these figures, the noise is approximated as Gaussian. We can see a lack of symmetry about zero for all the plots and also a sort of plateau in the x- and y-coordinates, but these are assumed to be specific to the map. The distance of three standard deviations is used which should give a 99.7% confidence interval:

$$\boldsymbol{\Sigma}_{icp} = \begin{bmatrix} \left(\frac{0.3}{3}\right)^2 & 0 & 0 \\ 0 & \left(\frac{0.3}{3}\right)^2 & 0 \\ 0 & 0 & \left(\frac{0.004}{3}\right)^2 \end{bmatrix}.$$

The variance has been set a bit high in order to handle the fact that the noise is not really Gaussian and spread the particles more.

To create the importance density, a mix of Gaussian distributions has been chosen. This is because the noise is approximately Gaussian and that the ICP can converge to different states $\boldsymbol{x}_k^{icp}$. First, K ICP states are sampled from the PF distribution from last time step giving $\boldsymbol{x}_{k-1}^i$, $i = 1, \ldots, K$. These are propagated through the process model giving $\boldsymbol{x}_k^i = \boldsymbol{f}(\boldsymbol{x}_{k-1}^i, \boldsymbol{v}_{k-1}^i)$. Then the ICP method is run on each ICP state, giving $\boldsymbol{x}_k^{icp,i} = \boldsymbol{f}_{\text{icp}}(\boldsymbol{x}_k^i, \boldsymbol{z}_k, m)$. The importance density becomes

$$q(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{z}_k) = \sum_{i=1}^{K} \bar{\pi}^i \mathcal{N}(\boldsymbol{x}_k \mid \boldsymbol{x}_k^{icp,i}, \boldsymbol{\Sigma}_{icp}), \tag{8.2}$$

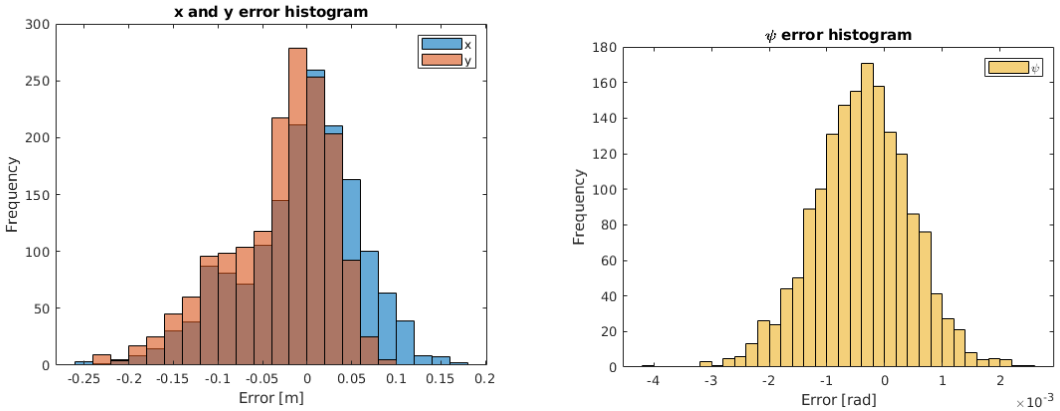


Figure 8.15: Histograms of the error of the state made by propagating the actual state from last iteration through the process model and using the ICP method. This was done for 50 runs with 32 time steps, giving $50 \cdot 32 = 1600$ noise realizations for each coordinate of the state. $N_m = 1800$ measurement were used for the ICP method.

where $\mathcal{N}(\boldsymbol{x}_k \mid \boldsymbol{x}_k^{icp,i}, \boldsymbol{\Sigma}_{icp})$ is a Gaussian distribution with mean $\boldsymbol{x}_k^{icp,i}$ and covariance $\boldsymbol{\Sigma}_{icp}$, $\bar{\pi}^i$ is the normalized π^i. π^i is the weight for each Gaussian chosen to be

$$\pi^i = \frac{1}{r^i} P(\boldsymbol{x}_k = \boldsymbol{x}_k^{icp,i} \mid \boldsymbol{x}_{k-1}),$$

where r^i is the RMSE value from the ICP method and $P(\boldsymbol{x}_k = \boldsymbol{x}_k^{icp,i} \mid \boldsymbol{x}_{k-1})$ is the process model evaluated at the state after the ICP convergence.

When using this importance density, the particles were spread less than in the original particle filter. But the particle distribution once again got the problem of a few particle getting high weights and most particle getting very low weights. Better results were achieved when the weighting did not use the exact importance density in (8.2), but weighted the particles only using the $\bar{\pi}^i$:

$$q(\boldsymbol{x}_k^{\{i_p\}} \mid \boldsymbol{x}_{k-1}^{\{i_p\}}, \boldsymbol{z}_k) = \bar{\pi}^i. \tag{8.3}$$

This means sampling with (8.2) and weighting with (8.3). This is only a trick that gave better results and might be difficult to motivate theoretically. The particle filter weight update equation in (5.8) becomes

$$\bar{w}_k^{\{i_p\}} \propto \bar{w}_{k-1}^{\{i_p\}} \frac{p\left(\boldsymbol{z}_k \mid \boldsymbol{x}_k^{\{i_p\}}\right) p\left(\boldsymbol{x}_k^{\{i_p\}} \mid \boldsymbol{x}_{k-1}^{\{i_p\}}\right)}{\bar{\pi}^i},$$

where $\bar{\pi}^i$ is the Gaussian mixture weight of the ICP state $\boldsymbol{x}_k^{icp,i}$ with index $i \in \{1, \ldots, K\}$ and i_p is the particle index.

8.3 Evaluation

Using the ICP method to make an importance density as described in Section 8.2 including both the approximately uniform noise and the Gaussian noise gave the results in Figure 8.16. The filter still seems to have the problem of spikes in the ANEES with really high values. And it can also go higher than in this figure, which is shown in Figure 8.18. But compared to the implementations earlier, the RMSE is decreased. This is likely due to the high precision and accuracy of the ICP method, and the use of a lot of measurements, some properties of which were described in Section 8.1.

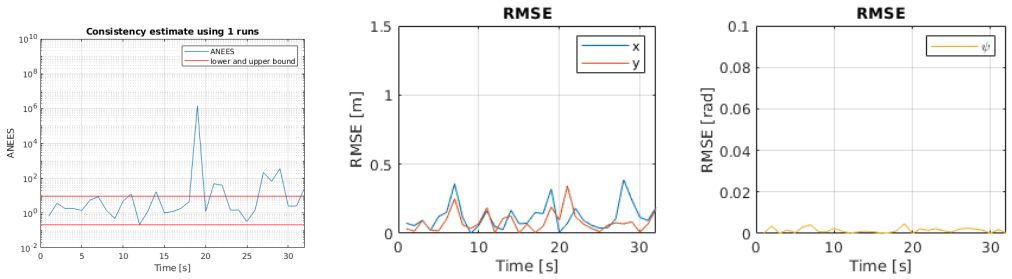
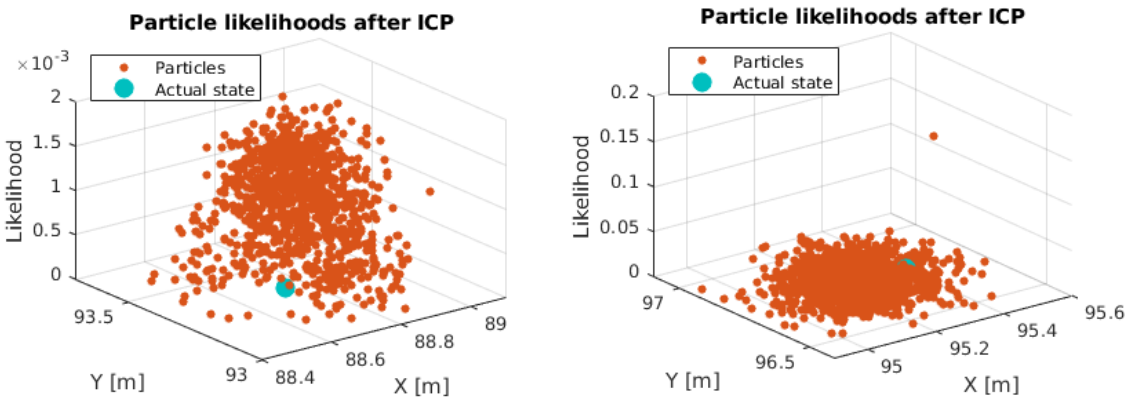


Figure 8.16: ANEES and RMSE for the particle filter after one run of using the ICP method to make the importance density. 1800 measurements were used in the ICP method and $N_m = 3$ measurements were used in the particle filter with $N = 1000$ particles.

At some time steps the particle distribution can have a big empirical covariance while in others it might be very small. This has been visualized in Figure 8.17. The big empirical covariance in Figure 8.17a will for the same error as in Figure 8.17b give a smaller NEES and vice versa. The jumps in empirical covariance can explain the spikes in the NEES in Figure 8.16.

The reason that the particle distribution can get such a low empirical covariance as in Figure 8.17b might be a combination of properties of the sensor model and the map. Section 6.2 discussed some of these properties in one dimension, namely heading. It showed that the likelihood could become quite narrow by introducing noise, and how different types of noise could affect the distribution. Taking into account that the estimation is now in three DOF and not just one, and fewer particles are used, could explain why the filter is even more sensitive to noise. It is now sensitive to the noise in all three DOF and not just one. In that case, trying out different importance densities, as tried here, will not give much better results with the same amount of particles. A possible solution is to make changes to the sensor model or the map used.

The time used by the method is estimated in Table 8.1. We can see that it is faster than the UPF in Table 7.1 and slower than the ordinary particle filter. The consistency results and errors are discussed next.



(a) Big empirical covariance

(b) Small empirical covariance

Figure 8.17: Two different particle distributions from the same run of the particle filter with ICP used to make the importance density. The x- and y-axis of the two figures have the same scale.

Which PF	N	N_m	N_{runs}	**Time**[s]	**Time**$\left[\frac{\text{s}}{\text{iteration}}\right]$
ICP PF	1000	3	5	1279	8.0

Table 8.1: Te run time of the particle filter using ICP to make an importance density, where N is the number of particles used, N_m is the number of measurement used and N_{runs} is the number of times the simulation is run. 1800 measurements were used in the ICP. The time per iteration of the particle filter is shown. The number of time steps for each run is 32. The time per iteration is calculated as $t/(32 \cdot N_{\text{runs}})$, where t is the time a simulation used.

The consistency results after running the simulations $N_{runs} = 5$ times are shown in Figure 8.18. They show that the ANEES can become quite big at the same time as the RMSE is low. This is likely due to the same reasons as mentioned above, the small empirical covariance. One way to solve this problem, when the importance density is made using the ICP method, is to force the particles to a Gaussian distribution with a set covariance around the mean, instead of the regularization step. This would not change the spread of the particles, but it would artificially decrease the ANEES. This is possible because the ICP convergence does not depend on the weights of the particles and is very consistent when many measurements are used. It is on the other hand not a good solution, since it only fixes a symptom and not the problem of bad representation of the actual state probability distribution. In that case, the reasons for using a particle filter disappear and one might instead use a Kalman filter or an other mono-modal method.

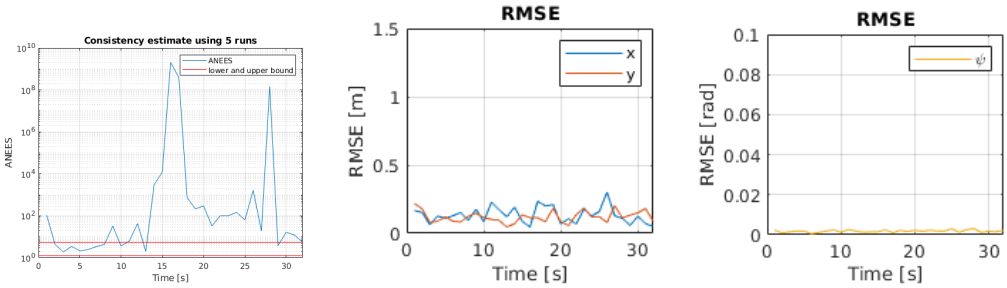


Figure 8.18: ANEES and RMSE after $N_{runs} = 5$ runs of the simulations using the ICP method to make an importance density. 1800 measurements were used in the ICP method and $N_m = 3$ measurements were used in the particle filter with $N = 1000$ particles.

Chapter 9

Discussion

This chapter discusses different topics related to the project.

There is no definitive best way of going forward from this project. Several methods within the given delimitations have been attempted and analyzed, non of which were ideal. But these analyzes give insight for what to embark on. Analyzes of the workings of the particle filter, the UPF and the ICP method with the use of a physically based sensor model in an occupancy grid has been made. These will be discussed and compared.

9.1 Analyzes and evaluations

9.1.1 Generic particle filter

The particle filter analysis, in Section 6.2, is not only useful for implementations of the particle filter. The particle filter can be viewed as a way of showing the state probability distribution, at least in the ideal simulations without noise. Given that one lidar ray has a low range, the state of the vessel could not possibly be in a state that would give any longer range. In this sense, the analysis described a relation between the measurements and the state, via the map. This can for example be useful for the developments of a Kalman filter. That said, the Kalman filter would not solve any problems of the sensor model, but this suggestion will be discussed later.

The analysis of the particle filter gave mostly expected results. For example, when one ray of the lidar passes a corner, the measurement get a sudden increase in range. The particle placed a bit off such that the expected measurement is smaller again will

get low probability values, this is shown in Figure 6.6b. Also how the distribution was affected by changes in the map and increase in uncertainty was mostly expected. They were expected based on the sensor model used and its dependency on every single measurement.

A property that was not expected was the behavior of the particle filter in the full three DOF with fewer particles. The empirical covariance would become very small, only leaving one particle with a high value. It was initially thought that this was because many Gaussian distributions were multiplied together, one for each ray of the lidar. But as mentioned in Section 8.3, when using fewer measurements and a narrower importance density, the particle distribution could still get the problem of low empirical covariance. The best explanation found is that it is because of the noise mentioned in Section 6.2, but affecting all three DOF, and the resulting lower amount of particles per DOF.

Even though the evaluation of the particle filter showed a run time that was too high and difficulties with high ANEES, the filter might be usable. When it comes to the run times, they are assumed to be reduced quite a bit when the method is run in an other programming language or after explicitly trying to make the code more efficient. When it comes to the high ANEES, it could partly be mitigated using synthetically high uncertainty, as shown in Section 6.5. Of all the particle filters implemented in that chapter, that last section probably describes the best one.

9.1.2 UKF-based importance density

The analysis of the UKF-based importance density, in Section 7.2, gave ambiguous results. On the one side, using a simple map and no noise gave much less spread of the particles toward the correct state, exactly the property of a better importance density. The importance density in the figures also seemed to represent a distribution of where the vessel was likely to be. On the other side, the resulting distribution after weighting was rarely intuitive and there was rarely a distinct peak in likelihood at the actual vessel state. This indicates that there might be better ways of weighting the particles, maybe by increasing the uncertainty as done in the generic particle filter.

The biggest problem of the UPF is the run time. The method scales badly with the number of measurement, and with a sensor model using many measurements, this gives long run times. The results in Table 7.1 show that the UPF used well over ten times as long as the generic particle filter did, making it both bad for implementations and not least difficult to work with in simulations. The reasons for the method being slow and suggested solutions were discussed in more detail in Section 7.3 and Section 7.4.

9.1.3 ICP-based importance density

An important part of the localization method is the outlier detection. This is where the measurements that do not give a lot of information, or measurements that are noisy, are not used for estimation. This was not done in all particle filters, but was done in the ICP method, see Figure 8.14. The ICP method worked a lot better with outlier detection and the localization method in general could have the same benefit. If outlier detection was used more in the particle filter, the distributions might look a lot nicer with lower ANEES. This would also fix any problem of measurements hitting on top of the quay, see Figure C.2, or on buildings in the environment instead of the edge of the quay.

The analysis of the particle filter using the ICP method to make an importance density, Section 8.1, is also not only useful for the particle filter. It looks at how the ICP method convergence in the occupancy grid. This can also be useful when the ICP method is used as a sensor model, or equivalently the measurement function. If it is for example used in a Kalman filter, the method will not be exempt from converging to wrong states, which is therefore an important aspect to take into account when designing the filter. This once again goes back to the analyzes not only analyzing the particle filter, but also the relation between the sensor, or the ICP method in this case, and the state via the map.

The act of ray tracing is slow and the physically based sensor model gives a lot of measurements. One possible solution is to use an other sensor model, for example the ICP method. The ICP method gives a single state out, which makes a possible model a linear dependency on the state. If this had been done in the particle filter, the usage of the UKF to make an importance density would go much faster. Since the relation between measurements and state is linear, it might enable the use of a Kalman filter. The benefit of that is that it does not have to use a lot of particles but only gives a single estimate, making it possibly faster than all methods tested in this project.

9.2 Comparison

All methods seemed to have similar consistency properties measured with ANEES. It is assumed that this is because of the same sensor model is being used for the weighting. Different parameters could be used to affect the ANEES with the different methods, but non gave ideal results. It is therefore difficult to say which method was best in this regard.

The method that gave the smallest RMSE was using the ICP method to create an importance density. While the UPF and the generic particle filter had similar

RMSE, the use of many measurements in the ICP method placed the particles closer to the true state.

The method that used the least amount of time was the generic particle filter. It only uses the process model as an importance density, which takes less time than what the UKF and the ICP uses in the creation of importance densities. Especially the UPF used a lot of time. This is mostly relevant for real time implementation, but also for the time it takes to do simulations.

The method that did best in total seems to be the generic particle filter. More specifically, the version that weights the particles with an increased uncertainty, see Section 6.5. By increasing the uncertainty, the ANEES was decreased to a more appropriate level. The RMSE was not so bad as too lose track and the time the method used was low compared to the other methods. This reduction in ANEES shows that the tuning of parameters is important for the performance of localization methods.

9.3 Importance of modeling and tuning

It is important to know how well the models match reality. This requires a lot of experimentation and gathering of data for both the vessel and each sensor. The data from these experiments can then be used to establish how accurate for example the process model is. The lidar data can be useful for modelling how many rays miss, the precision of hits, etc. (See Section 4.5.3 for a list error types and references.)

Good tuning of the methods is very important for good performance. This project has attempted to see what happens when different parameters are tuned, but not all possibilities have been studied. For example the ICP-based particle filter could use less measurements, instead of 1800 use for example 100. This would reduce the run time of the method and probably get it closer to the time the generic particle filter used. If the method kept giving a low RMSE for the vessel state, it had likely taken over as the best method. Also the increasing of the uncertainty for weighting the particles could for the UKF-based method and the ICP-based method have been tested, in which case the ANEES would likely decrease an reach a similar level as the generic particle filter. The reason for not testing all possibilities is both the lack of time and that the focus of this project is on finding a way to go forward.

9.4 Limitations

The delimitations of this project can be regarded as limitations. The delimitation to the use of a particle filter might be unnecessary. One might argue, based on the figures in this thesis, that the property of handling multi-modality was not used. It might also be difficult to see how multi-modality is necessary if the initial position is known, especially if a faster method is run so fast that the process uncertainty gets small. On the one hand, the ICP method gave convergence to what seemed to be several local minima. This was on the other hand taken care of by tuning of different parameters such as the amount of measurements used. If the method does not have to represent multi-modal distributions, it might be possible to use other methods than the particle filter which will decrease the run time significantly. For localization in general, the case of ambiguities in the state might happen, and it is important to take that into consideration.

Only using a single sensor model is a limitation to this project. Only the physically based sensor model described in Section 4.6 was used, which does not give a clear understanding of pros and cons with using different sensor models. For example the likelihood fields method mentioned several times in Chapter 2 could be of interest. The ICP method has on the other hand been analyzed and to a certain degree compared to the physically based sensor model. The use of a physically based sensor model is good for genera lidar localization analysis, but might not be ideal for implementation.

This study was also limited to using an occupancy grid as a map. The flexibility of the map with what structures it can represent is good, and the map can even be made by lidar data, then edited manually, if desired, before it is used for localization. This also enables the use of different maps for different situations. It could be interesting to try similar methods in a 3D occupancy map or a feature-based map. Using a 3D map, it would be possible to establish the difference in performance when for example roll is estimated versus assuming only three DOF for the vessel. The lidar measurements hitting above the quay would also not be as big of a problem since the map would explain where there are structures. The buildings and other structures on land could then be used in a better way for localization. Using a feature-based map would for example reduce the amount of measurements from the number of rays used to the number of features detected. This was once again not done because of limitations in time and comprehensiveness of the project.

Some other limitations are that the methods were only run on simulated data and the analyzes are tightly connected to how well the process and the sensor is modelled. For example the model of the dynamics, the process model. The uncertainties on the vessel velocity might be a lot smaller than what was used in the simulations. In that case, some of the filters with high ANEES would get a lower ANEES in the real application. The properties analyzed in this project are on the other hand more

dependant on the workings of the methods and not necessarily the parameters used.

Lastly, this project has not explicitly compared pure localization to SLAM. Since the application of the methods is an autonomous ferry, where the map is usually known, it might not be necessary with SLAM. SLAM is on the other hand a more flexible method with regards to where it can work, so that the map does not have to be made prior to deployment of the vessel. The vessel would anyway need an understanding of the changing elements of the environment, especially other boats, for this to be employed safely in the water passage, even if this is not done with the use of SLAM.

Chapter 10

Conclusion and future work

10.1 Conclusion

The localization of an autonomous ferry, using a lidar and a map of the environment, with such accuracy that it can dock is seemingly possible. This has more precisely been done in simulations with a physically based sensor model, a particle filter and an occupancy grid.

The available sensor data can be utilized in different ways. While using the physically based sensor model gave the ability to analyze the problem at hand, the use of ICP resulted in lower RMSE. Some of the analyzes conducted in this project have seemingly not been done before. The analysis of the ICP method shows that the ICP method might in itself be a good sensor model, and its linear dependence on the state makes it compatible with more estimation frameworks than the physically based sensor model.

The generic particle filter can be simple to implement, but can simultaneously be difficult to get good performance from. The particle filter using the motion model, the UKF or the ICP method to make an importance density all gave good tracking, but also unexpectedly high ANEES. This is concluded to be because of the sensor model in combination with the occupancy grid. The problem is solved, at least to a certain degree, by the increase of uncertainty when weighting the particles. Using the motion model as importance density gave faster run times, while using ICP to make an importance density gave low RMSE values.

These observations can together be used to find a way to go farther with localization of the vessel, resulting in a ferry that can dock autonomously.

10.2 Future work

Some suggestions of future work are:

- Use the ICP method as the sensor model. This gives the benefit of a linear dependency on the state, enabling the use of a Kalman filter. Even if the method is not ideal with regards to not representing multi-modal distributions, it can still suffice as a benchmark and also supplement other aspects of autonomous ferry development.
- Localize in changing environments. The aspect of other ships moving in a harbour is as big of a change of the environment as cars moving in a parking lot, mentioned in Section 2.8. An implementation of such a method would combine the freedom of manually designing the occupancy grid with the a robustness to changes in the environment.
- Localize in 3D. How important it is to do 3D localization will depend on how much the roll and pitch affect the estimation , and this will have to be analyzed in order to make a decision on 2D versus 3D.
- Experiment to get good sensor models. Some experiment suggestions for analyzing the process model, requiring known vessel velocity and pose, are:
  - Drive the vessel at a constant velocity. Use both zero yaw rate and non-zero yaw rate. Also run with and without speed in the sway direction.
  - Drive the vessel with a varying velocity.
  - Drive the vessel in waves and wind.

  For modelling the sensor, it will be useful to place the sensor a fixed distance from a wall and see how the measurements represent that wall.
- Implement the generic particle filter used in this project on the vessel. It is then suggested to also do the experiments mentioned above and find how well the models fit reality.

Bibliography

[1] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing*, 50(2), 2002.

[2] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-Squares Fitting of 3-D Point Sets. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-9(5), 1987.

[3] Autoferry, NTNU. Autoferry. `https://www.ntnu.edu/autoferry`, mar 2019.

[4] Abraham Bachrach, Ruijie He, and Nicholas Roy. Autonomous Flight in Unknown Indoor Environments. *International Journal of Micro Air Vehicles*, 1(4):217–228, dec 2009.

[5] T Bailey and H Durrant-Whyte. Simultaneous localization and mapping (SLAM): part II. *IEEE Robotics Automation Magazine*, 13(3):108–117, sep 2006.

[6] Yaakov Bar-Shalom, X.-Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., 2001.

[7] Stuart Bennett. Nicholas Minorsky and the automatic steering of ships. *IEEE Control Systems Magazine*, 4(4):10–15, 1984.

[8] Paul J. Besl and Neil D. McKay. A Method for registration of 3-D shapes. In Paul S. Schenker, editor, *SPIE Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, apr 1992.

[9] Morten Breivik and Jon-Erik Loberg. A Virtual Target-Based Underway Docking Procedure for Unmanned Surface Vehicles. *IFAC Proceedings Volumes*, 44(1):13630–13635, jan 2011.

[10] Claus Brenner. Vehicle Localization Using Landmarks Obtained by a LIDAR Mobile Mapping System. *IAPRS*, XXXVIII, P, 2010.

[11] C Cadena, L Carlone, H Carrillo, Y Latif, D Scaramuzza, J Neira, I Reid, and J J Leonard. Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[12] José Luis Blanco Claraco. A tutorial on SE(3) transformation parameterizations and on-manifold optimization. Technical report, Universidad de Málaga, 2019.

[13] Nicholas Dalhaug. *SLAM for Autonomous Docking using Lidar(Not published, specialization project)*. NTNU, Trondheim, 2018.

[14] Raffaello D'Andrea and Assistants. Lecture notes 10 and 11, Recursive estimation 151-0566-00L at ETHZ, 2018.

[15] F. Daum. Nonlinear filters: beyond the Kalman filter. *IEEE Aerospace and Electronic Systems Magazine*, 20(8):57–69, aug 2005.

[16] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[17] H Durrant-Whyte and T Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[18] Ethan Eade. Lie Groups for 2D and 3D Transformations. `http://www.ethaneade.org/lie.pdf`, 2017.

[19] A. Elfes. Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI1990)*, 1990.

[20] Daniele Fontanelli, Luigi Ricciato, and Stefano Soatto. A Fast RANSAC-Based Registration Algorithm for Accurate Localization in Unknown Environments using LIDAR Measurements. In *2007 IEEE International Conference on Automation Science and Engineering*, pages 597–602. IEEE, sep 2007.

[21] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[22] Daniel Frank, Andrew Gray, Kevin Allen, Tess Bianchi, Kipling Cohen, Daniel Dugger, Jake Easterling, Matthew Griessler, Sylvie Hyman, Matthew Langford, Ralph Leyva, Lucas Murphy, Jason Nezvadovitz, Anthony Olive, Blake Peterson, David Soto, Forrest Voight, Daniel Volya, Timothy Williams, Eric Schwartz, Carl Crane, Ira Hill, and Shannon Ridgeway. University of Florida: Team NaviGator AMS. In *RobotX Forum*, 2016.

[23] Christian Gentner, Siwei Zhang, and Thomas Jost. Log-PF: Particle Filtering in Logarithm Domain. *Journal of Electrical and Computer Engineering*, 2018:11, mar 2018.

[24] Fredrik Gustafsson. Particle filter theory and practice with positioning applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):53–82, jul 2010.

[25] Fredrik Gustafsson. *Statistical sensor fusion*. Studentlitteratur, 2:1 edition, 2010.

[26] Eiliv Hägg and Yingzhi Ning. *Map Representation and LIDAR-Based Vehicle Localization(MSc thesis)*. Chalmers, 2016.

[27] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, 2013.

[28] Esa Jokioinen. AAWA Position Paper: Remote and Autonomous Ships - The next steps. Technical report, Rolls-Royce, 2016.

[29] Simon J. Julier and Jeffrey K. Uhlmann. New extension of the Kalman filter to nonlinear systems. In Ivan Kadar, editor, *SPIE. 3068, Signal Processing, Sensor Fusion, and Target Recognition VI*, page 182, jul 1997.

[30] Simon J. Julier and Rudolph van der Merwe. ukf.m. `https://www.cs.cmu.edu/%7Emotionplanning/papers/sbp_papers/kalman/ukf`, apr 2019.

[31] Rickard Karlsson and Fredrik Gustafsson. Bayesian surface and underwater navigation. *IEEE Transactions on Signal Processing*, 54(11):4204–4213, 2006.

[32] Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pages 155–160. IEEE, 2011.

[33] Kystlaget Trondhjem. Fløtmann. `https://www.kystlaget-trh.no/flotmann/`, 2014.

[34] Chee Sing Lee, Daniel E. Clark, and Joaquim Salvi. SLAM With Dynamic Targets via Single-Cluster PHD Filtering. *IEEE Journal of Selected Topics in Signal Processing*, 7(3):543–552, jun 2013.

[35] Zhixiang Liu, Youmin Zhang, Xiang Yu, and Chi Yuan. Unmanned surface vehicles: An overview of developments and challenges. *Annual Reviews in Control*, 41:71–93, jan 2016.

[36] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *ICLR 2017*, 2016.

[37] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* Sampling. In *Advances in Neural Information Processing Systems*, pages 3086—-3094, 2014.

[38] Justin E. Manley. Unmanned surface vehicles, 15 years of development. In *OCEANS 2008*, pages 1–4. IEEE, 2008.

[39] Wim Meeussen. REP 105 – Coordinate Frames for Mobile Platforms. `http://www.ros.org/reps/rep-0105.html`, 2010.

[40] Pierre Merriaux, Yohan Dupuis, Rémi Boutteau, Pascal Vasseur, and Xavier Savatier. Robust robot localization in a complex oil and gas industrial environment. *Journal of Field Robotics*, 35(2):213–230, mar 2018.

[41] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. In *IJCAI*, 2003.

[42] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, and Others. FastSLAM: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[43] Christian Musso, Nadia Oudjane, and François LeGland. Improving regularised particle filters. In *Sequential Monte Carlo methods in practice*, pages 247–271. Springer, 2001.

[44] U.S. Navy. The Navy Unmanned Surface Vehicle (USV) master Plan. Technical report, U.S. Navy, 2007.

[45] Erik Nelson. B(erkeley) L(ocalization) A(nd) M(apping)! `https://github.com/erik-nelson/blam`, 2019.

[46] Marius Strand Ødven and Edmund Førland Brekke. *Lidar-Based SLAM for Autonomous Ferry(MSc thesis)*. NTNU, 2019.

[47] Jason M O'Kane. *A Gentle Introduction to ROS*. Independently published, 2013.

[48] OpenCV team. About - OpenCV library. `https://opencv.org/about.html`, mar 2019.

[49] Panagiotis Papadimitratos and Aleksandar Jovanovic. Protection and fundamental vulnerability of GNSS. In *2008 International Workshop on Satellite and Space Communications, IWSSC'08, Conference Proceedings*, pages 167–171. IEEE, oct 2008.

[50] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[51] Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[52] RoboNation. 2018 Maritime RobotX Challenge Task Descriptions and Specifications. Technical report, RoboNation, 2018.

[53] Philipp Rosenberger, Martin Holder, Marina Zirulnik, and Hermann Winner. Analysis of Real World Sensor Behavior for Rising Fidelity of Physically Based Lidar Sensor Models. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 611–616, jun 2018.

[54] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[55] Brage Sæther and Edmund Førland Brekke. *Navigation and Motion Control for milliAmpère: Theory and Experiments(MSc thesis)*. NTNU, 2019.

[56] Alexander Schaefer, Lukas Luft, and Wolfram Burgard. An Analytical Lidar Sensor Model Based on Ray Path Information. *IEEE Robotics and Automation Letters*, 2(3):1405–1412, jul 2017.

[57] Roland Siegwart, Illah Reza Nourbakhsh, Davide Scaramuzza, and Ronald C Arkin. *Introduction to autonomous mobile robots*. MIT press, 2011.

[58] A. F. M. Smith and A. E. Gelfand. Bayesian Statistics without Tears: A Sampling-Resampling Perspective. *The American Statistician*, 46(2):84–88, 1992.

[59] Joan Solà. Quaternion kinematics for the error-state Kalman filter. *arXiv preprint arXiv:1711.02508*, nov 2017.

[60] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[61] Gian Diego Tipaldi, Daniel Meyer-Delius, and Wolfram Burgard. Lifelong localization in changing environments. *International Journal of Robotics Research*, 32(14):1662–1678, 2013.

[62] Tobias Valentin Rye Torben. *Hybrid Control of Autonomous Ferries(MSc Thesis)*. NTNU, 2018.

[63] Rudolph van der Merwe, Arnaud Doucet, Nando de Freitas, and Eric Wan. The unscented particle filter. Technical report, Cambridge University Engineering Department, 2000.

[64] Rudolph van der Merwe, Arnaud Doucet, Nando de Freitas, and Eric Wan. The Unscented Particle Filter. *Advances in neural information processing systems*, pages 584—-590, 2001.

[65] Velodyne LiDAR. Velodyne LiDAR PUCK ™ Datasheet. Technical report, Velodyne LiDAR, 2019.

[66] Velodyne LiDAR. VLP-16 User Manual 63-9243 Rev. D. Technical report, Velodyne LiDAR, 2019.

[67] Eric A Wan and Rudolph Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium, AS-SPCC 2000*, pages 153–158, 2000.

[68] Andrew Webb, Bradley Donnelly, Jesse Stewart, Jonathan Wheare, Michael Kossatz, Scott Hutchinson, Shane Geyer, and Tenzin Crouch. Development and Testing of the TopCat Autonomous Surface Vessel for the Maritime RobotX Challenge 2016. In *RobotX Forum*, 2016.

[69] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter, 1995.

[70] William Woodall. rviz. `http://wiki.ros.org/rviz`, mar 2019.

[71] Ryan W Wolcott and Ryan M Eustice. Robust LIDAR localization using multiresolution Gaussian mixture maps for autonomous driving. *The International Journal of Robotics Research*, 36(3):292–319, 2017.

[72] Xia Yuan, Chun-Xia Zhao, and Zhen-Min Tang. Lidar scan-matching for mobile robot localization. *Information Technology Journal*, 9(1), 2010.

[73] Chen Zhang, Marcelo H. Ang, and Daniela Rus. Robust LIDAR Localization for Autonomous Driving in Rain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3409–3415. IEEE, oct 2018.

[74] Ji Zhang and Sanjiv Singh. LOAM: Lidar Odometry and Mapping in Real-time. In *Robotics: Science and Systems*, volume 2, page 9, 2014.

Appendix A

Experiment data

This project is based on some experiments done before it in the master's project [46]. The data used is in a broad sense explained in this chapter.

A.1 Pre-project experiments

Data has been gathered for an earlier master's project, see Section 2.3 for information about that master's project. Those experiments were based in the Brattøra harbour, see Section 3.3.2 about the quays. The ferry Milliampere was here used for the purpose of SLAM. The data was gathered using ROS and recorded into several rosbags, see Section 3.5.1 about ROS. Gathering all data in the rosbag format, makes it possible to replay the scenario in real-time. That means that the data can be used and analyzed as if the vessel did the whole procedure exactly the same way, getting the same sensor readings and doing the same actions. This process was also used in the experiments made in this master's project.

The ferry was driven in the pattern shown in Figure A.1. Starting from the dock, it turned around and drove south closer to the structures. Then the ferry moved southwest before it went northeast going into the different quay areas. Lastly it went straight back, stopped and rotated right before it drove straight towards the quay and docked again.

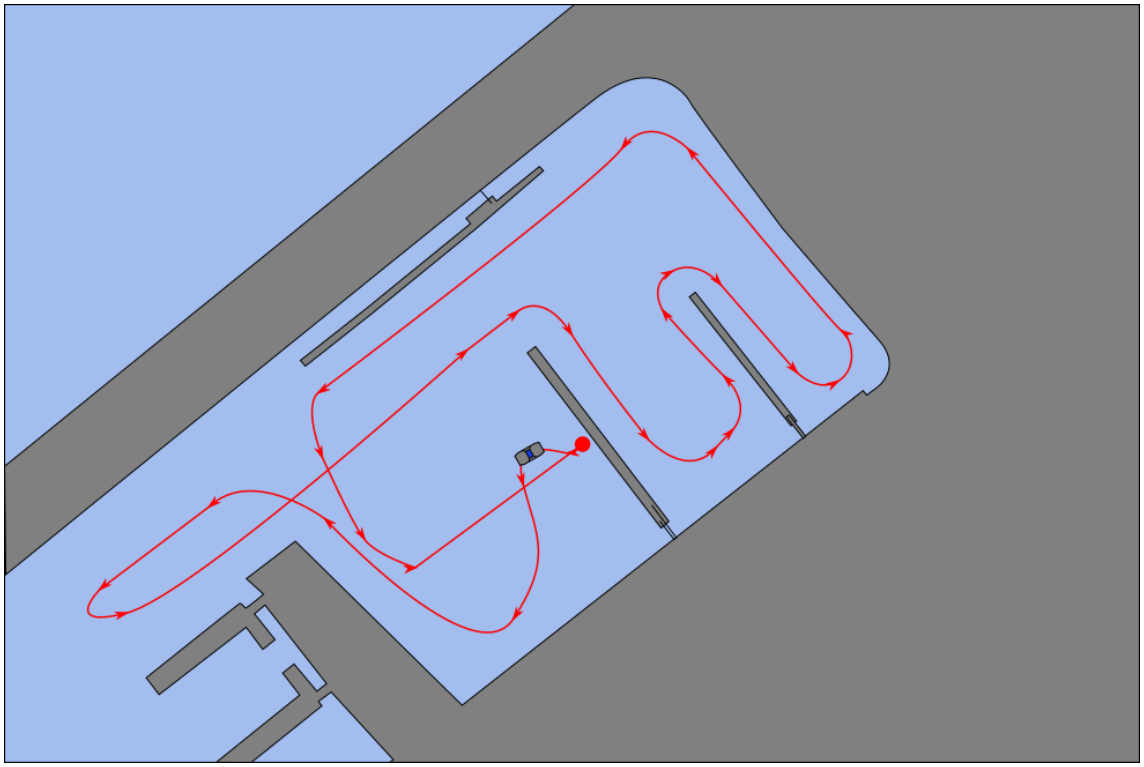


Figure A.1: The path taken by Milliampere in the pre-project experiments. The figure illustrates the Brattøra harbour in gray, water in blue and the path in red. The vessel is shown close to a quay. Other boats are not shown even though they were there, as can be seen in the data.

A.2 Rosbag overview

One of the rosbags given from the earlier experiments is "velodynemasterset.bag". It contains the lidar data and IMU data on the standard ROS message formats "sensor_msgs/PointCloud2" and "sensor_msgs/Imu" respectively. They are sent on the "/velodyne_points" and "/imu_topic" topics respectively. See Listing A.1 for a list of the topics and message types. While the lidar data was publishing data with respect to the reference frame "velodyne", the IMU was with respect to the frame "imu". None of these frames are specified through the "/tf" topic in the other rosbag, see the frames in Figure A.2. That topic is standard for publishing the relation between different frames.

Listing A.1: The different topics and message types in the rosbag "velodynemasterset.bag"

```
path:        velodynemasterset.bag
version:     2.0
duration:    10:36s (636s)
start:       Sep 11 2018 13:17:52.85 (1536664672.85)
end:         Sep 11 2018 13:28:29.12 (1536665309.12)
size:        1.1 GB
messages:    69932
compression: none [1319/1319 chunks]

types:
sensor_msgs/Imu          [6a62c6daae103f4ff57a132d6f95cec2]
sensor_msgs/PointCloud2  [1158d486dd51d683ce2f1be655c3c181]

topics:
/imu_topic           63627 msgs    : sensor_msgs/Imu
/velodyne_points      6305 msgs    : sensor_msgs/PointCloud2
```

The other rosbag, "obccorrectedtime.bag", has topics and message types as shown in Listing A.2. That bag has many different messages and topics, where many of them use custom message types. One of the used topics here is the "custom_msgs/NorthEastHeading" on the "/navigation/eta" topic. It is a custom message consisting of the north, east and heading of the vessel relative to the starting pose. The message type does not include timestamp nor a reference frame frame.

Listing A.2: The different topics and message types in the rosbag "obccorrectedtime.bag"

```
path:        obccorrectedtime.bag
version:     2.0
duration:    10:36s (636s)
start:       Sep 11 2018 13:17:52.69 (1536664672.69)
end:         Sep 11 2018 13:28:29.12 (1536665309.12)
size:        74.4 MB
messages:    553432
compression: none [91/91 chunks]
```

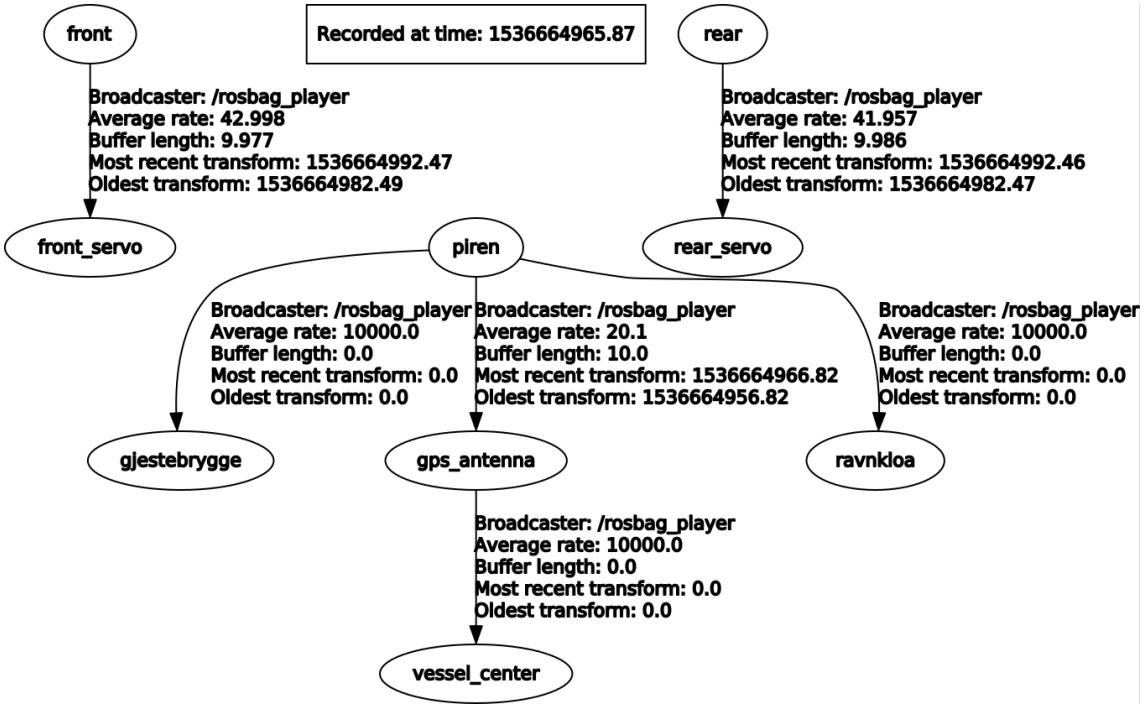


Figure A.2: The different reference frames published in "obccorrectedtime.bag".

```
types:
custom_msgs/ActuatorSetpoints    [2d64702992bfea9b6443fa0b9864979b]
custom_msgs/MotorState           [5ba26985322fe9ed7cb8984452bc3eb4]
custom_msgs/NorthEastHeading     [f8c4d8d295356fc4c7989ce2c0e1b148]
custom_msgs/RemoteControlState   [ceda10c7b37590a99e768bcaea5d1257]
custom_msgs/SurgeSwayYaw         [d86ccd8ad254d064df2af607e63b6ac0]
custom_msgs/ThreeDofForce        [5d59525997a21f5b026ea2746ebcfb38]
custom_msgs/gnssGGA              [76d41a768710775209ac34b58a4ce202]
custom_msgs/gnssHDT              [99a6e32aaf44ebb75a8836a630ce410e]
custom_msgs/orientationEstimate  [8061e110314ddf08ca1dfbc48d314df8]
custom_msgs/rawGPSdata           [9b7b5e623e612f1bf238822a134f2656]
rosgraph_msgs/Log                [acffd30cd6b6de30f120938c17c593fb]
sensor_msgs/Imu                  [6a62c6daae103f4ff57a132d6f95cec2]
sensor_msgs/JointState           [3066dcd76a6cfaef579bd0f34173e9fd]
std_msgs/Int16                   [8524586e34fbd7cb1c08c5f5f1ca0e57]
std_msgs/String                  [992ce8a1687cec8c8bd883ec73ca41d1]
tf2_msgs/TFMessage               [94810edda583a504dfda3829e70d7eec]

topics:
/actuator_ref_1                                  6362 msgs    : custom_msgs/ActuatorSetpoints
/actuator_ref_2                                  6362 msgs    : custom_msgs/ActuatorSetpoints
/actuators/actuator_1/azimuth_angle             55464 msgs    : std_msgs/Int16
/actuators/actuator_1/thruster/motor_state       1536 msgs    : custom_msgs/MotorState
/actuators/actuator_2/azimuth_angle             55248 msgs    : std_msgs/Int16
/actuators/actuator_2/thruster/motor_state       1531 msgs    : custom_msgs/MotorState
/dynamic_positioning/control_action              6362 msgs    : custom_msgs/ThreeDofForce
/guidance/dockpose                                  1 msg     : custom_msgs/NorthEastHeading
/guidance/reference/acceleration                 6363 msgs    : custom_msgs/NorthEastHeading
/guidance/reference/pose                         6362 msgs    : custom_msgs/NorthEastHeading
/guidance/reference/velocity                     6363 msgs    : custom_msgs/NorthEastHeading
/joint_states                                  109678 msgs    : sensor_msgs/JointState
/navigation/eta                                 12725 msgs    : custom_msgs/NorthEastHeading
/navigation/nu                                  12725 msgs    : custom_msgs/SurgeSwayYaw
/rc_state                                        8264 msgs    : custom_msgs/RemoteControlState
/rosout                                            20 msgs    : rosgraph_msgs/Log
/rosout_agg                                         2 msgs    : rosgraph_msgs/Log
/supervisor/mode                                 6275 msgs    : std_msgs/String
/tf                                             67264 msgs    : tf2_msgs/TFMessage
/tf_static                                      19091 msgs    : tf2_msgs/TFMessage
/vectorVS330/GPGGA                              12726 msgs    : custom_msgs/rawGPSdata
/vectorVS330/fix                                12724 msgs    : custom_msgs/gnssGGA
/vectorVS330/heading                            12726 msgs    : custom_msgs/gnssHDT
/xsens/imu                                      63629 msgs    : sensor_msgs/Imu
/xsens/orientation                              63629 msgs    : custom_msgs/orientationEstimate
```

A.3 Lidar data

The lidar data in the message form of "sensor_msgs/PointCloud2" on the topic "/velodyne_points" is a point cloud. This means that calculations are necessary to find which azimuth angle a specific beam was sent out on. The azimuth angle might for example be useful for simulating the lidar in a measurement function. To find the azimuth angle of each ray, trigonometry is once again used, see Figure 4.3 for visualization of how the lidar works:

$$\tan(\alpha) = \frac{x}{y}, \tag{A.1}$$

$$\alpha = \text{atan2}(x, y), \tag{A.2}$$

where α is the azimuth angle. $\text{atan2}(x, y)$ is the tangent inverse which also finds in which quadrant the point is, to give $\alpha \in (-\pi, \pi]$. Notice that this definition of the azimuth angle is the inverse of the usual one, but it is based on the definitions shown in Figure 4.3.

From the data gathered, the settings of the lidar can be found. Using the ROS command "rostopic hz /velodyne_points", the data frequency was measured to be in the range $9.90\,\text{Hz} - 9.97\,\text{Hz}$, which indicates that the setting used for the lidar was $10\,\text{Hz} = 600\,\text{min}^{-1}$, see Section 3.4.1 for a description of the possible angular settings. From (3.1) this corresponds to an angular resolution of $0.199\,\frac{^\circ}{\text{firing cycle}}$.

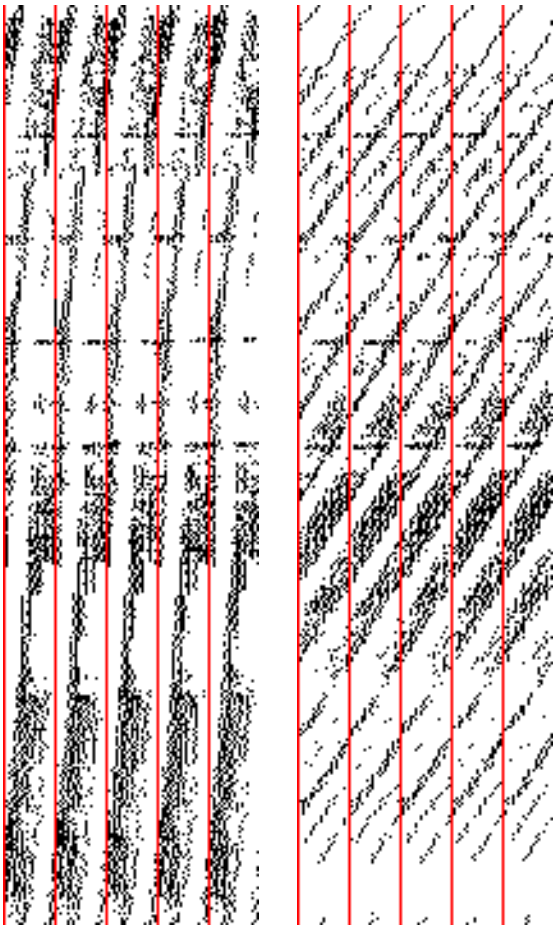
This does not mean that the rays are measured at fixed azimuth angles. Figure A.3 shows that the angles at which the lidar projects varies. The images were made by each image being a pointcloud, one full rotation. For each point in the cloud, the azimuth angle is calculated using (A.1). This may not be a precise way of calculating the azimuth angles, but is the only way when the data is only represented as a point cloud. There seems to be 5 groups for each row in the images, which indicates that the resolution is approximately 0.2°. But as the images have these groups placed differently within the same row, the azimuth angles seem to vary. Also the angles of the stripes change over time, which indicates that the azimuth angles vary. One reason might be that the controller for the sensor does not care about the angles being consistent, but rather that the rotation speed is correct. One resulting problem might be difficulties in simulating the lidar precisely.

We get the number of points by using the angular resolution:

$$\frac{360\,\frac{^\circ}{\text{rot}}}{0.199\,\frac{^\circ}{\text{firing cycle}}} = 1808.45\,\frac{\text{firing cycle}}{\text{rot}}.$$

Having 16 rays gives

$$1808.45\,\frac{\text{firing cycle}}{\text{rot}} \cdot 16\,\frac{\text{ray}}{\text{firing cycle}} = 28\,935.19\,\frac{\text{ray}}{\text{rot}}.$$



(a) One point cloud's azimuth angles.

(b) A point cloud a few seconds later.

Figure A.3: The two images show point cloud azimuth angles at two different measurement times. The black pixels represent angels where there is at least one point. The white pixels are where there are no points. The rows of the images go from 0 to 359 and are different angles with each column corresponding to the decimal 0.00 to 0.99, at a resolution of 0.01°. So these are two images of $360 \cdot 100$ azimuth angles each. The red lines are to separate the angles to intervals of 0.2° and to make them easier to compare. Said differently, the images could each have been one pixel high and 36000 pixels wide, but it would be difficult to compare them.

That is how many points one point cloud might contain. But it is just an approximation since the sensor needs a controller that has a deviation and some rays might not be reflected and give a point. Since the data is unordered, the height field in the point cloud is 1. Using the command "rostopic echo /velodyne_points/width", the number of points was found to oscillate in the range $3000 - 10000$ points. This is a lot less than initially expected, but might be due to the vast number of rays going up into the sky and down into the water, where few to no points reflect back.

A.4 Velocity

The velocity uncertainty is estimated from the topic "/navigation/nu" on the data given. This contains velocity in surge, sway and yaw as shown in Figure A.4. They are given in the NED frame, similarly as for the position data on "/navigation/eta". The uncertainty is difficult to know without more testing, but the velocities are assumed Gaussian with means and standard deviations as shown in Table A.1. These numbers are based on the data series by reading of the plot as in Figure A.4 for the whole duration of the course. The uncertainty is estimated by an approximate measure of two standard deviations, that is approximately a 95% confidence interval. They serve as simple estimates that enable simulations, but are not tested on the actual system.

	μ	2σ
Surge	$0\ \frac{\text{m}}{\text{s}}$ to $2.5\ \frac{\text{m}}{\text{s}}$	$0.5\ \frac{\text{m}}{\text{s}}$
Sway	$-1\ \frac{\text{m}}{\text{s}}$ to $1\ \frac{\text{m}}{\text{s}}$	$0.3\ \frac{\text{m}}{\text{s}}$
Yaw	$-0.3\ \frac{\text{rad}}{\text{s}}$ to $0.3\ \frac{\text{rad}}{\text{s}}$	$0.1\ \frac{\text{rad}}{\text{s}}$

Table A.1: Typical velocity values for the vessel based on the data on the "/navigation/nu" topic. μ is an estimated mean of a Gaussian distribution and 2σ corresponds approximately a 95% confidence interval.

For better estimates of the motion uncertainty, the vessel should be tested with a set velocity in surge, sway and yaw and the error should be measured between the motion model expected state and the ground truth. This would require ground truth position and velocity.

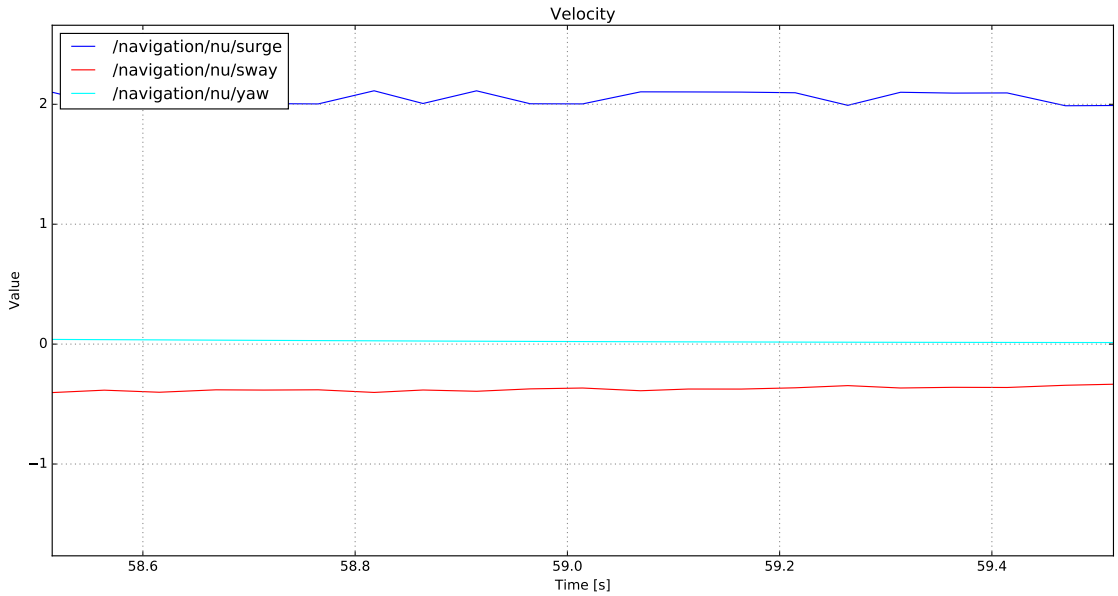


Figure A.4: Values for surge, sway and yaw velocity from the data gathered in the earlier project, on the topic "/navigation/nu". The image is a representative period from the data series. The values are in $\frac{\mathrm{m}}{\mathrm{s}}$ for surge and sway, and $\frac{\mathrm{rad}}{\mathrm{s}}$ for yaw.

Appendix B

Mapping to an occupancy grid

This chapter explains how to take the data from earlier experiments described in Appendix A and make a map.

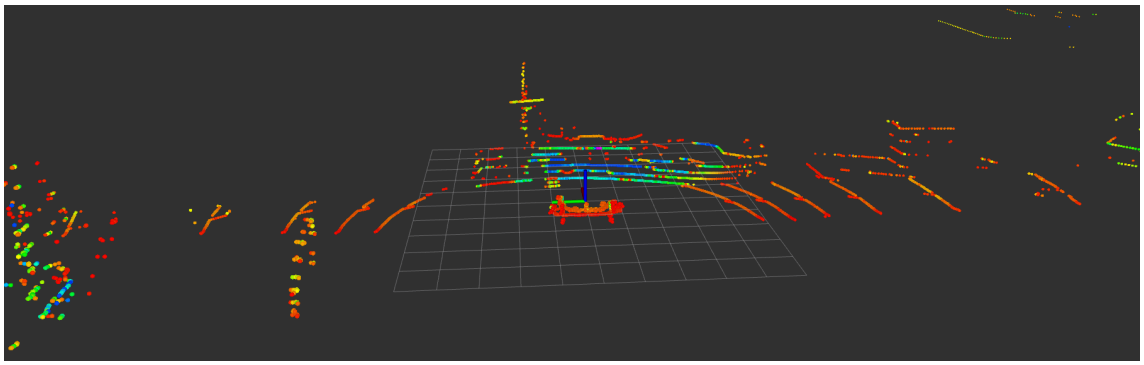
B.1 Point filtering

After initial inspection of the rosbag data, the lidar data was visualized using rviz, see Section 3.5.2 for an explanation of rviz. In order for the lidar point cloud data to be usable by methods that analyzes the environment, the vessel should be removed from the data. This enables all the points in the point cloud to be usable in for example scan matching or SLAM. Assuming a static environment, all moving elements have then been removed.

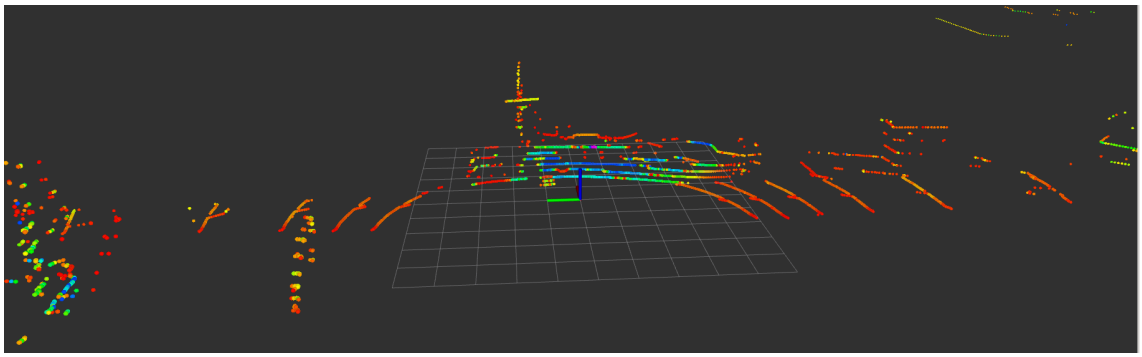
See Figure B.1 for an illustration of difference between the point clouds. On the first image, the equipment on the top of the vessel is visible, whereas it is not on the second image. This was done by first making a ROS node in C++ that subscribes to the lidar data. A box around the lidar was made by inspection, in which all data was removed. The ROS node uses PCL and the conditional removal functionality, see Section 3.5.5 for an description of PCL. The resulting point cloud was then published on another topic.

B.2 Getting a map

The occupancy grid was made using a combination of GPS and lidar, see Appendix A for an explanation of the data. To localize the ferry in the making of the map, the



(a) Before filtering.



(b) After filtering.

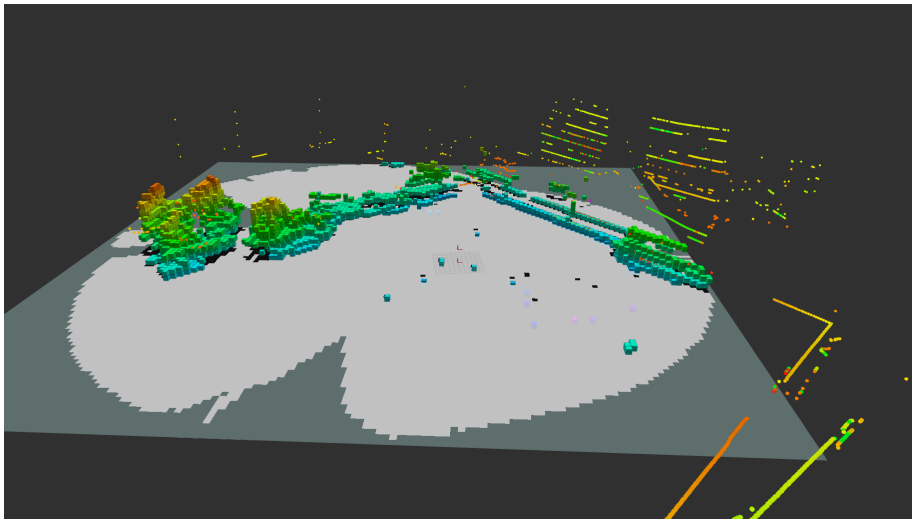
Figure B.1: Illustrations of the point clouds from the lidar before and after filtering. The axes represent the frame of the lidar, with the red axis pointing forward. The other points represent what looks like the quay and some other ships.

pose was given through the earlier mentioned "/navigation/eta" topic. The lidar data was made to an OctoMap, as described in Section 3.5.3. The same software also gives the map in the form of a occupancy grid, with the map frame set to "map". The occupancy grid is published on the "/projected_map" topic. Some parameters of the method are the radius in which to make lidar points specified occupied cells and the resolution of the map. The mapping happens using the point clouds in the frame "velodyne", and the data is combined by reformulating the pose messages into a "tf2_msgs/TFMessage" that represents the transformation from the "map" to "velodyne" frames. The data on the "/navigation/eta" topic is in a NED reference frame and is transformed into a frame with the z-axis upward as specified by the ROS standards described in Section 3.8. That reformulation happens in a new ROS node.

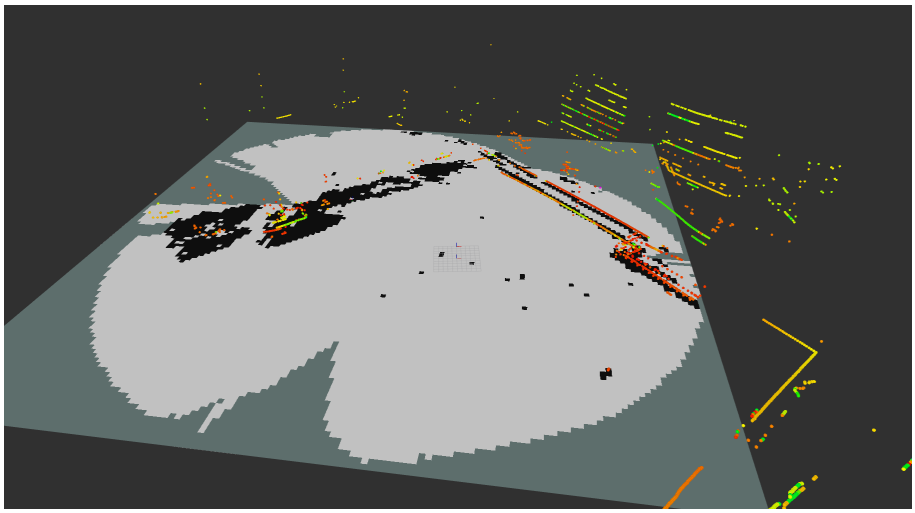
An illustration of the mapping process is given in Figure B.2. The mapping seems to work fine, but there are some misplaced occupied cells especially as the range of the OctoMap method is increased. They can be noise from water. But the jittering of the measurements relative to the map, especially during turns, might be due to the pose estimates and the lidar estimates not being synchronized, or the accuracy of the pose estimates. But since the map is supposed to be used for localization, it is not necessary to have a perfect map and it can be changed in an editing tool later. The resulting map after going around the whole map in the pattern described in Figure A.1 is shown in Figure B.3.

B.3 Saving an occupancy grid as an image

The occupancy grid message type used is "nav_msgs/OccupancyGrid" published on the topic "/projected_map". This is now transformed via a new ROS node to a standardized OpenCV image. Using the nodes "image_transport", "cv_bridge" and OpenCV in ROS, the cell values of the occupancy grid are converted to image colors, for more information on the software see Section 3.5. Also the axes needed to be changed, as the x-axis of the map points north, y-axis of the map points west, x-axis of the image points east and the y-axis of the image points south. The resulting images are then published in ROS as images instead of occupancy grids, see the result in Figure 3.8. The image is then saved with a call to "imwrite($\cdot$)".

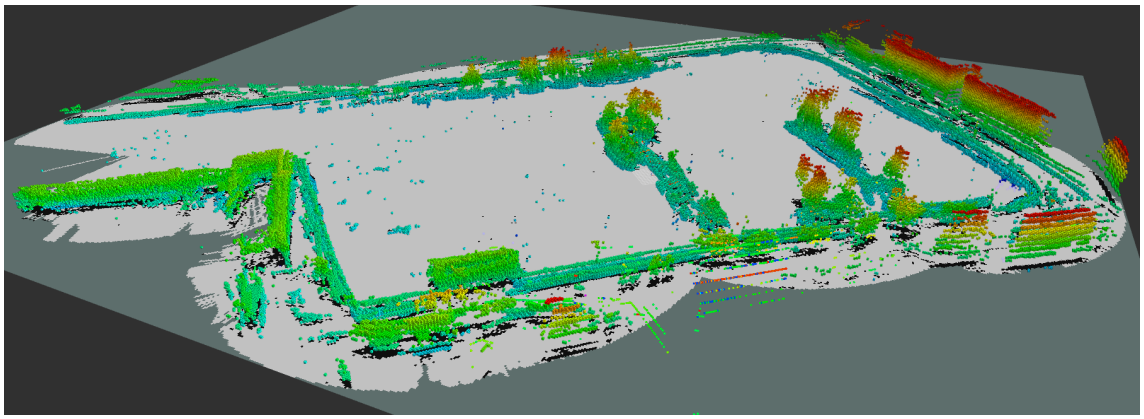


(a) OctoMap and occupancy grid.

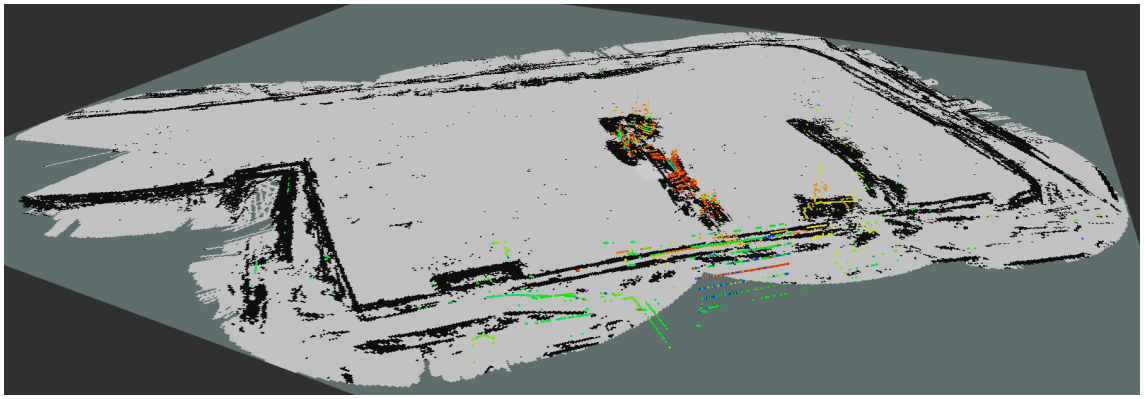


(b) Only occupancy grid.

Figure B.2: The figure illustrates the mapping process, both the OctoMap and the occupancy grid. The occupancy grid has black squares representing the horizontal occupied space. The colored blocks are the pieces of the OctoMap. The colored points are the lidar data in one full rotation. The resolution was here 1 m and the range 50 m.



(a) OctoMap and occupancy grid.



(b) Only occupancy grid.

Figure B.3: The figures show the OctoMap and the occupancy grid resulting from mapping the whole trajectory in Figure A.1 with a resolution of 0.5 m and a range of 50 m. The horizontal plane is the occupancy grid, with light gray squares representing free space, black squares being occupied space and the dark gray being unknown. The colored cubes combined is the OctoMap. The colored points are the lidar data of one full rotation close to the quay after mapping.

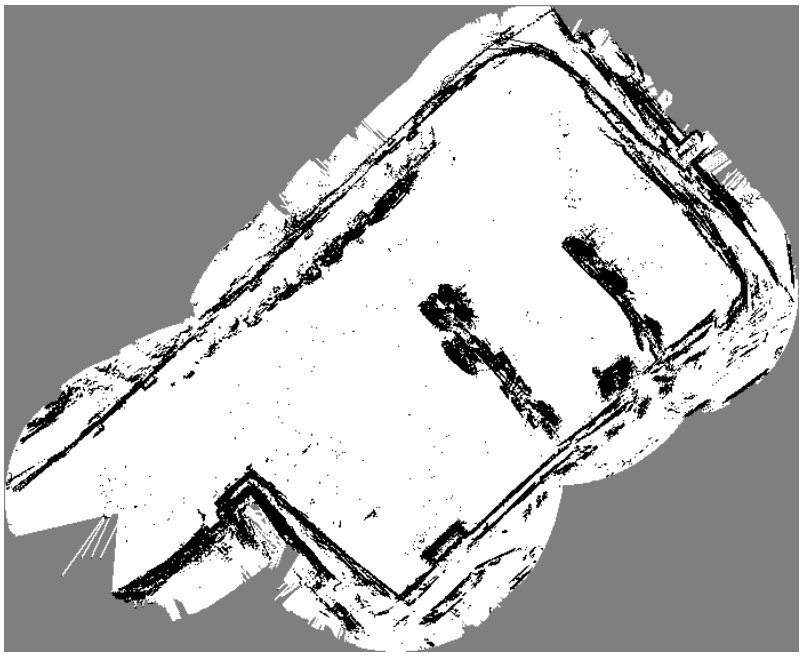


Figure 3.8 revisited.

Appendix C

Implementation

This chapter explains some of the aspects of implementing the suggested methods. More precisely it explains how to remove measurements that are from the vessel itself and how to reduce the 3D measurement to simulate 2D measurements.

To compare the 3D measurements to a 2D map, different methods might be used. Some are described in Section 4.5. The choice of doing ray tracing in the 2D map was made. That makes it possible to make an estimate of what measurements the lidar should get for each pose in the map. But if the vessel is in free cells and a ray goes toward an occupied cell, no measurement will be simulated to be behind that occupied cell in the map. So only the points closest to the lidar will be used, giving an output similar to what a 2D lidar would get. To use this sensor model, the lidar settings needed to be known, and the way to find this is described in Appendix A.3. The ray tracing in a 2D map was chosen because of the simplicity and existence of a 2D map and since it gives a way of estimating the probability of different measurements at different poses. It also reduced the amount of measurements greatly, approximately by 16 because it is the amount of vertically spaced rays in the lidar in this project.

The definition used for the 2D scan is the data type "sensor_msgs/LaserScan" defined in ROS. It contains the angles for each ray as well as the range to object collision for each ray. Contrary to the lidar frame, see Figure 4.3, the azimuth angles are defined with zero at the x-axis and increasing anti-clockwise. So the azimuth angles are defined as

$$\tan(\alpha) = \frac{y}{x}.$$

To find the azimuth angle α from the x- and y-coordinates we do

$$\alpha = \text{atan2}(y, x),$$

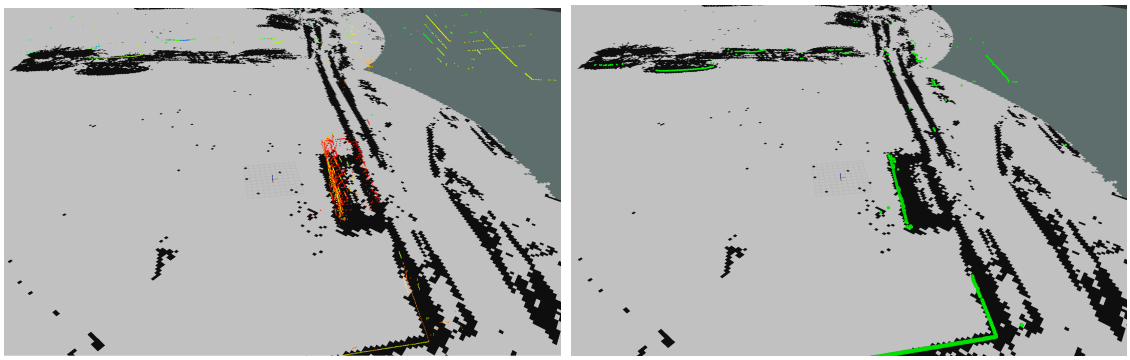
which gives an $\alpha \in (-\pi, \pi]$. The angles at which the lidar projects rays seems to not be constant and it therefore chosen that 1800 evenly spaced horizontal rays are

used. For each 3D point measured, the azimuth angle is calculated and estimated to be the closest one of the $N_m = 1800$ angles. Let $\{\alpha^i\}_{i=0}^{N_m-1}$ be the set of N_m evenly space azimuth angles. The the index i of the azimuth angle a measurement corresponds to is then calculated by

$$i = \left\lfloor \left(\left(\text{atan2}(y, x) + 2\pi + \frac{\alpha_r}{2} \right) \bmod (2\pi) \right) \frac{N_m}{2\pi} \right\rfloor,$$

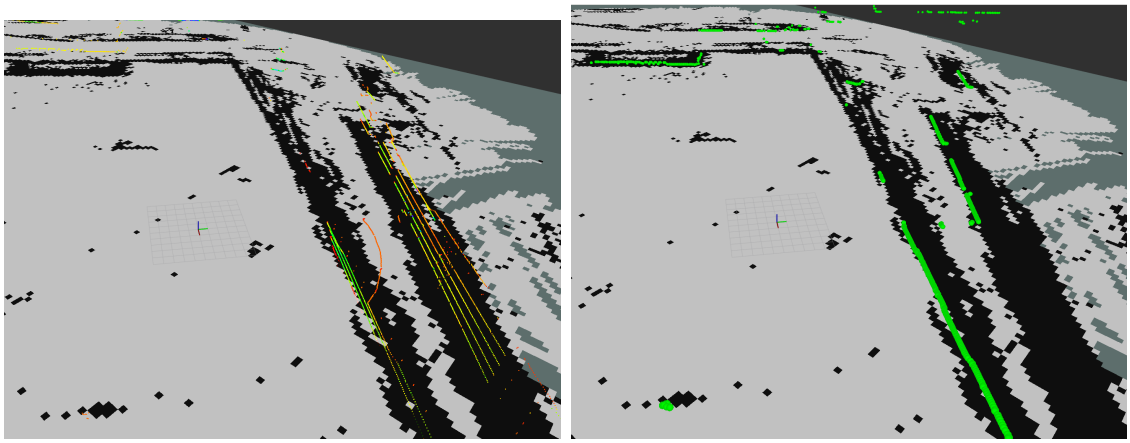
where $a \bmod b$ gives the remainder after dividing a by b, $\lfloor a \rfloor$ rounds a down to the closest integer and α_r is the azimuth resolution of the scan. The measurement is then used if the range is smaller than any other measurement with approximately the same angle, that is the same index i.

The resulting scans look like the ones in Figure C.1. Most of the points seem reasonable and could be estimated from the pose of the vessel and the map. But some points are behind black cells in the map, which is even more clear in Figure C.2. This is due to the geometry of the lidar and the fact that it is a 3D lidar, and not 2D. Some rays might for example go into the water, while the others pass above the quay and hit buildings instead. See Section 3.6 for a more quantitative explanation of this.



(a) Point cloud (b) Laser scan

Figure C.1: Illustration of the resulting laser scan from a point cloud.



(a) Point cloud (b) Laser scan

Figure C.2: Illustration of the resulting laser scan from a point cloud. Here there are many points that might be hard to estimate from the map and the pose of the vessel.

Nicholas Dalhaug

Lidar-based Localization for Autonomous Ferry

NTNU
Kunnskap for en bedre verden