Energy-Efficient Particle Filter SLAM for Autonomous Exploration

Elke Salzmann

TU Delft

Energy-Efficient Particle Filter SLAM for Autonomous Exploration

by

Elke Salzmann

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on the 25th of August 2021.

Student number:	4311450
Project duration:	December 1, 2020 – August 25, 2021
Thesis committee:	dr. ir. J. Dauwels
	dr. R. T. Rajan
	dr. F. Fioranelli

TU Delft

Abstract

Autonomous robots are increasingly used in more and more applications, such as warehouse robots, search-and-rescue robots and autonomous vacuum cleaners. These applications are often in environments where the GPS signals are denied or inaccurate, which makes it difficult to localize the robot in an unknown environment. To overcome this problem the framework of Simultaneous Localization and Mapping (SLAM) is typically used. This solution constructs a map of the environment with the use of cameras or range sensors, while keeping track of the location of the robot in it.

To extend the exploration time of these battery powered robots, the energy consumption of the SLAM algorithm could be reduced. It is assumed that if the computational load of an algorithm reduces, the energy consumption of the algorithm reduces as well.

An existing paradigm to solve SLAM is the use of a particle filter, which tracks the trajectory of the robot and simultaneously maps the environment. The question answered in this thesis is how to make this algorithm more energy-efficient to be able to deploy this framework in more applications and make the existing robots more sustainable.

In this thesis two methods are investigated. In the first method, the information about the landmarks is incorporated in the trajectory estimation as spatial constraints, to try to achieve a higher accuracy with less particles and thus subsequently a smaller computational load. The proposed method is validated by simulations on synthetic datasets. This method shows improvements in terms of the estimation accuracy. However, it is more computational complex than the existing algorithms, so it is considered less energy-efficient. The second method researched in this thesis, is the implementation of a parallelized particle filter. This method processes the observation measurements in parallel for the different particles and communicates the information between the particles efficiently. It should reduce the computational time, to enable partial computation of the algorithm to reduce the computational load. This method is validated on the same datasets as the first method using simulations. This method shows improvements on the run time and thus on the computational load, especially for a larger number of particles and is therefore more energy-efficient.

The two separate methods have been analyzed and compared with state of the art methods. Both methods deliver equally good or better results in terms of accuracy. However, the computation time of constrained FastSLAM does not outweigh the improvement in accuracy. On the other hand, the parallelized particle filter shows significant improvement over the existing solutions.

Preface

After seven amazing years at the TU Delft, it was time to start my master thesis as a final piece to get my master degree and to finish my study Electrical Engineering at the TU Delft. This work has not only been a product of nine months of hard work, reading a pile of papers, writing hundreds of Matlab lines and finetuning the content of this thesis. It has also been a product of the years before, the courses I have followed, the people I have met and the discussions I have had.

I would like to express my sincerest appreciation to my daily supervisor Raj. You gave me the freedom to work on topics I was interested in, but also gave me direction when I needed it! You pushed me to get the most out of it by regularly commenting my work in detail and coming up with great suggestions when I got stuck.

Secondly, I would like to thank Justin Dauwels for being the chair of my thesis committee. You listened carefully to my presentations and came up with interesting angles to work on. My thanks also go to Francesco Fioranelli for being in my committee and the rest of the CAS group and all their members, for letting me feel part of the group, even in this challenging time of working from home. Especially Martijn, Bichi, Felix, Brenda and Calum, thank you for the interesting discussions and chitchat during our virtual coffee meetings.

I am very thankful for my boyfriend Bas for always believing in me, even when I didn't. Without you this work would not have existed.

None of this would have happened if it weren't for my parents and the rest of my family. You encouraged me to do what I loved during my study period, even though that would mean that I moved to Switzerland and doing three gap years. I am grateful that you gave me freedom to enjoy all the possibilities during my study period and always tried to understand what I was working on.

My time in Delft would have never been the same without the amazing experiences I have gained during the last eight years, for those experiences I would like to thank the best Board members I could think of with whom I have had the best Board year ever! And of course my amazing Nuna team members with whom I have experienced the greatest adventures in Delft and Australia.

Finally I would like to thank all my friends and people I have met, who made the years in Delft unforgettable.

Elke Salzmann
Gouda, August 2021

Contents

Abstract iii

List of Figures ix

List of Tables xi

List of Acronyms and Symbols xiii

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Problem definition . . . 1
  - 1.2.1 Scope . . . 2
- 1.3 Organization of this thesis . . . 3

2 Autonomous navigation 5
- 2.1 State space model . . . 5
- 2.2 Inputs . . . 5
  - 2.2.1 Odometry . . . 5
  - 2.2.2 Perception . . . 6
- 2.3 SLAM . . . 8
  - 2.3.1 Formulation . . . 8
  - 2.3.2 Loop closure . . . 11
- 2.4 Filter design for SLAM . . . 11
  - 2.4.1 Kalman filter . . . 12
  - 2.4.2 Particle filter. . . . 12
  - 2.4.3 Graph-based . . . 12
  - 2.4.4 Comparison . . . 13

3 FastSLAM 15
- 3.1 Motivation . . . 15
- 3.2 Robot model . . . 16
  - 3.2.1 Motion model . . . 16
  - 3.2.2 Observation model . . . 17
- 3.3 Particle Filter . . . 18
  - 3.3.1 Rao-Blackwellized particle filter . . . 18
  - 3.3.2 Algorithm . . . 19
- 3.4 Simulation FastSLAM . . . 21
- 3.5 Performance measures . . . 21
  - 3.5.1 Error of the trajectory estimation . . . 23
  - 3.5.2 Error of the map estimation . . . 23
  - 3.5.3 Kullback-Leibler Divergence . . . 23
  - 3.5.4 Time complexity . . . 24
  - 3.5.5 Memory . . . 24

4 Constrained FastSLAM 25
- 4.1 Introduction . . . 25
- 4.2 Mathematical derivation . . . 25
  - 4.2.1 Proposal distribution . . . 26
- 4.3 Simulation setup . . . 26
  - 4.3.1 Datasets . . . 27

4.4 Results . . . 27
4.4.1 Dataset 1 . . . 27
4.4.2 Dataset 2 . . . 28
4.4.3 Dataset 3 . . . 29
4.4.4 Dataset 4 . . . 29
4.4.5 Kullback-Leibler Divergence . . . 30
4.4.6 Computational complexity . . . 32
4.4.7 Memory . . . 32
4.5 Conclusion . . . 33
4.6 Future work . . . 33

5 Parallelized particle filter 35
5.1 State of the art . . . 35
5.2 Robot model . . . 35
5.3 Mathematical derivation . . . 36
5.3.1 Graph theory . . . 36
5.3.2 Laplacian decomposition . . . 37
5.3.3 Comparison . . . 38
5.4 Simulations . . . 38
5.5 Results . . . 39
5.5.1 Dataset 1 . . . 39
5.5.2 Dataset 2 . . . 39
5.5.3 Dataset 3 . . . 40
5.5.4 Dataset 4 . . . 41
5.5.5 Time complexity . . . 43
5.5.6 Memory . . . 43
5.6 Conclusion . . . 43
5.7 Future work . . . 43

6 Conclusion and future work 45
6.1 Conclusion . . . 45
6.1.1 Constrained FastSLAM . . . 45
6.1.2 Parallelized Particle Filter SLAM. . . . 45
6.1.3 Energy efficiency . . . 46
6.2 Future work . . . 46
6.2.1 Constrained FastSLAM . . . 46
6.2.2 PPF-SLAM . . . 46
6.2.3 Multi-agent system . . . 46
6.2.4 Active autonomous exploration . . . 46
6.2.5 Implementing on a platform . . . 47

Bibliography 49

List of Figures

1.1 Block diagram of processes in autonomous navigation [25] 2

2.1 Different representations of an output map. 8
2.2 Dynamic Bayesian Network of the SLAM process. Each node represents a random variable and an edge represents the conditional dependence between the two nodes. The colored circular nodes are the measurements and the white circular nodes are the hidden variables that need to be estimated. [16] 10
2.3 Left: map constructed from odometry data. Right: map build from SLAM [1] 11
2.4 Computational complexity for different solutions of SLAM; Kalman filter, particle filter and Graph-based SLAM as a function of the number of landmarks. 14

3.1 Three-dimensional rotations described in roll, pitch, yaw [19, p. 97]. 16
3.2 3D motion model of robot, where the linear velocity v and angular velocities $\dot{\gamma}$, $\dot{\beta}$ and $\dot{\alpha}$ are denoted. 17
3.3 Importance sampling for particle filter. Samples are drawn from the proposal distribution π (red line) and weighted with the target distribution p (blue line). The weight of the sample is the importance $(p(x_k^{(j)})/\pi(x_k^{(j)}))$. The weighted samples are shown in the lower half part of the figure,where the weight of the samples is represented by the height of the sample. [34, p. 101] 19
3.4 Estimated trajectory and landmark positions using FastSLAM with 10 particles. The red trajectory is the groundtruth and the blue trajectory is the estimated trajectory. 22

4.1 Datasets used for the simulations. The trajectory (blue line) is the same, starting from point (0,0). The number of landmarks (red stars) differ. 28
4.2 Comparison of the performance of unconstrained and constrained FastSLAM on the first dataset containing 10 landmarks averaged over 100 Monte Carlo runs. 28
4.3 Comparison of the performance of unconstrained and constrained FastSLAM on the second dataset containing 50 landmarks averaged over 100 Monte Carlo runs. 29
4.4 Comparison of the performance of unconstrained and constrained FastSLAM on the third dataset containing 100 landmarks averaged over 100 Monte Carlo runs. 30
4.5 Comparison of the performance of unconstrained and constrained FastSLAM on the fourth dataset containing 100 landmarks along the trajectory averaged over 100 Monte Carlo runs. 30
4.6 Estimated mean of the Kullback-Leibler Divergence estimator. The KLD is an average calculated over 100 Monte Carlo runs. 31
4.7 Estimated variance of the Kullback-Leibler Divergence estimator. The KLD is calculated over 100 Monte Carlo runs. 32

5.1 Comparison of the eigendecomposition and power method for 100 iterations of the PPF-SLAM algorithm for $c = 7$. 38
5.2 Comparison of the performance of PPF-SLAM and FastSLAM on the first dataset containing 10 landmarks averaged over 100 Monte Carlo runs. 40
5.3 Comparison between the run time of FastSLAM and PPF-SLAM for the first dataset containing 10 landmarks over 100 Monte Carlo runs. 40
5.4 Comparison of the performance of PPF-SLAM and FastSLAM on the second dataset containing 50 landmarks averaged over 100 Monte Carlo runs. 41
5.5 Comparison between the run time of FastSLAM and PPF-SLAM for the second dataset containing 50 landmarks over 100 Monte Carlo runs. 41

5.6 Comparison of the performance of PPF-SLAM and FastSLAM on the third dataset containing 100 landmarks averaged over 100 Monte Carlo runs. 42
5.7 Comparison between the run time of FastSLAM and PPF-SLAM for the third dataset containing 100 landmarks over 100 Monte Carlo runs. 42
5.8 Comparison of the performance of PPF-SLAM and FastSLAM on the fourth dataset containing 100 landmarks along the trajectory averaged over 100 Monte Carlo runs. 42
5.9 Comparison between the run time of FastSLAM and PPF-SLAM for the fourth dataset containing 100 landmarks along the trajectory over 100 Monte Carlo runs. 43

List of Tables

2.1 Comparison perception sensors for autonomous navigation 7
2.2 Comparison between Kalman filter algorithms 12
2.3 Comparison between the three paradigms for SLAM [26] 13

3.1 Parameters used for the simulation 22

4.1 Parameters used for the simulation 27
4.2 Comparison performance between unconstrained FastSLAM and constrained FastSLAM for 10 landmarks 29
4.3 Comparison performance between unconstrained FastSLAM and constrained FastSLAM for 50 landmarks 29
4.4 Comparison performance between unconstrained FastSLAM and constrained FastSLAM for 100 landmarks 30
4.5 Comparison performance between unconstrained FastSLAM and constrained FastSLAM for 100 landmarks in a structured way 31

5.1 Comparison between FastSLAM and PPF-SLAM 38
5.2 Parameters used for the simulation 39

List of Acronyms and Symbols

List of acronyms

ARMS	time-Averaged Root Mean Squared
DBN	Dynamic Bayesian Network
EIF	Extended Information Filter
EKF	Extended Kalman Filter
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IS	Importance Sampling
KF	Kalman filter
KLD	Kullback-Leibler Divergence
κ**NN**	κ-Nearest Neighbor
LIDAR	LIght Detection And Ranging
MAP	Maximum a Posteriori
MLE	Maximum Likelihood Estimation
PPF	Parallelized Particle Filter
RADAR	RAdio Detection And Ranging
RBPF	Rao-Blackwellized Particle Filter
RMS	Root Mean Squared
SIS	Sequential Importance Sampling
SLAM	Simultaneous Localization and Mapping
UKF	Unscented Kalman filter

List of symbols

In this thesis, matrices are denoted by bold face upper case letters ($\mathbf{A}$), column vectors are denoted by bold face lower case letters ($\mathbf{a}$) and real numbers are denoted by lower case letters (a).

$\hat{}$	Denotes an estimator
$\mathbf{0}$	All zero vector
$\mathbf{A}$	Adjacency matrix
$\mathbf{C}_k$	Spatial constraints on the state space
$\mathbf{D}$	Degree matrix
D_{KL}	Kullback-Leibler Divergence
$\mathbf{D}_k$	Proposal distribution for constrained FastSLAM
$\mathbf{e}_k$	Measurement noise at time k
$\mathbf{F}$	Eigenvector matrix
$f(\cdot)$	Dynamical model of robot
$h(\cdot)$	Sensor model of robot
k	Time step index
$\mathbf{L}$	Laplacian matrix
$\mathbf{m}$	Information of all landmarks
M	Number of landmarks
M_o	Number of observed landmarks
N	Number of particles
N_{eff}	Number of effective particles
$\mathcal{N}(\mu, \Sigma)$	Normal distribution around μ with covariance matrix Σ
$\mathcal{O}$	Big O notation (Time complexity)
$\mathbf{Q}$	Covariance matrix of process noise
r_{ik}	Measured distance between robot and landmark i
$\mathbf{R}$	Covariance matrix of measurement noise
$\mathbb{R}$	Set of real numbers
$\mathbf{S}_k$	Support set for constrained FastSLAM
T	Total time steps robot trajectory
$\mathbf{u}_k$	Input vector at time k
v	Linear velocity
$\mathbf{v}_k$	Process noise at time k
w	Weight of a particle
x_k	x-position of robot at time k
$\mathbf{x}_k$	State vector at time k
$\mathcal{X}$	Set of particles
y_k	y-position of robot at time k
z_k	z-position of robot at time k
$\mathbf{z}_{ik}$	Measurement of landmark i at time k
α	Velocity in yaw direction
$\boldsymbol{\alpha}$	Laplacian transform coefficients
β	Velocity in pitch direction
γ	Velocity in roll direction
$\gamma^{(j)}$	Log-likelihood of the measurements of particle j
θ	Orientation of the robot
$\boldsymbol{\mu}$	Mean vector
$\boldsymbol{\lambda}$	Eigenvalue
$\boldsymbol{\Lambda}$	Eigenvalue matrix
π	Proposal distribution
σ	Standard deviation
$\boldsymbol{\Sigma}$	Covariance matrix
ϕ_{ik}	Bearing to landmark i with respect to the heading of the robot
ω	Angular velocity

1 Introduction

In a world where tasks are increasingly being replaced by robots (like mowing, vacuum cleaning or delivering the mail) we expect the robots to take care of themselves and complete their tasks autonomously. It is impossible nowadays to imagine our society without these autonomously navigated robots. Hereby we expect robots to work faster and better than we do ourselves. The downside is that the robots require a growing amount of resources like hardware and energy. In this thesis, the focus will be on reducing the energy consumption of these robots to reach their full potential.

In this thesis, two improved algorithms for autonomous navigation of mobile robots are proposed to reach this goal. In this chapter the need for improved algorithms in autonomous navigated robots is explained. The state-of-the-art in autonomous navigation is explained in Section 1.1 followed by the problem definition in Section 1.2. The last section introduces the organization of the rest of this thesis.

1.1. Background

If a vehicle or robot is capable of planning and executing a path without intervention of people, it is able to navigate autonomously [31]. To do so, the robot needs sensors aboard to base its decisions on. In addition, some prior knowledge might be available, such as information of the environment or a goal that needs to be reached. The input of the sensors needs to be processed by the robot to localize itself and to be able to avoid obstacles and collisions with moving obstacles or people. This information is then used for the planning of the path and/or to build a map of the environment. The final step is that the robot needs to be able to control the actuators, such as steering and braking to reach the goal or to explore an unknown environment. This process is visualized in Figure 1.1. The inputs of the robot are visualized in the first block, processing these inputs fall under the perception of the robot. Using this processed data the navigation part can map the environment and plan a path for the robot. Finally, this information is used in the vehicle interface part of the robot to control itself.

One of the many challenges with autonomous navigation, is that the robot first needs to know where it is located within the environment to be able to localize itself, only then the planning of a path is possible. But on the other hand, if the robot does not move, it cannot explore the environment outside its observation range. An existing solution to this autonomous navigation problem is simultaneous localization and mapping (SLAM). This method explores a new environment, typically where global positioning systems (GPS) are denied or inaccurate, such as indoors on Earth, or in distant space-missions on other planets. By using this method the robot observes and maps the environment, using sensors, such as cameras or LIDAR, and localizes itself simultaneously in the map using the observations. With every observation the map and trajectory estimation of the robot is expected to be more accurate, because the uncertainty about the observed environment decreases with every measurement.

1.2. Problem definition

Robots that use SLAM to explore the environment are nowadays not only used to explore unknown environments, but also to vacuum clean houses or to mow the grass. Whether the robots work closer

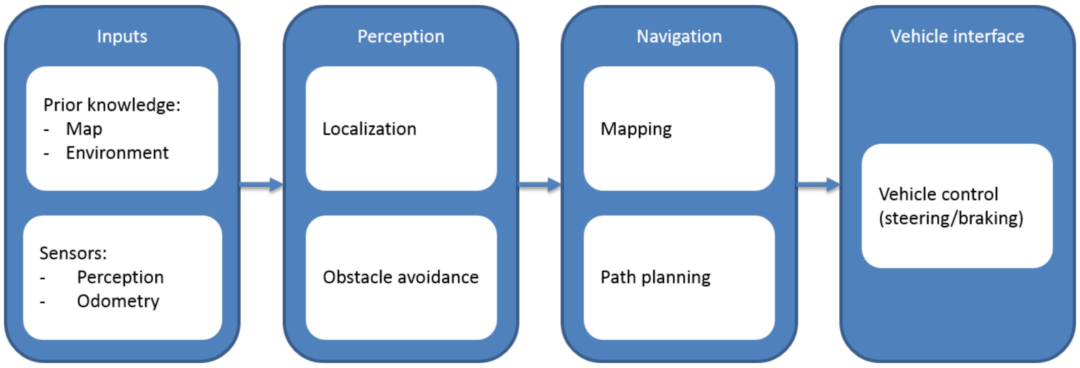


Figure 1.1: Block diagram of processes in autonomous navigation [25]

to home, or farther away, keeping the energy consumption of the robots as low as possible is crucial, since they run on small batteries. These batteries are either charged by solar panels, which can be a bottleneck for the performance of the robot, or by charging systems.

The research in energy-efficient software development is still immature and incorporating this metric during the whole development of software is often overlooked [12]. A proper metric that defines the resource and sustainability of an algorithm does not exist yet. However, broadly speaking, the metrics that influence energy consumption of an algorithm can be divided into multiple categories;

- **Hardware**; related to how the hardware consumes energy for different components
- **Code**; related to the dynamics of the code itself
- **Process**; related to the energy consumption during the development of the algorithm

Finding an overall metric is a subjective matter and out of the scope of this thesis. The algorithms in this thesis will be considered to be language and platform independent, so the energy consumption of the hardware will not be taken into account in this thesis, as well as the process energy consumption.

In this thesis the focus will be on metrics that are directly related to the code. A major part of the research done in this field is summarized in Ergasheva et al. [12]. This paper describes that for example the number of functions, the number of executed instructions and the number of accesses to the memory have shown to influence the energy consumption of a code [3]. These metrics are summarized in this thesis by looking at the computational complexity of the algorithms. The research question of this thesis is: What is an energy-efficient way of implementing a Simulatenous-Localization-and-Mapping-algorithm? The question is split into two key subquestions:

- How can the available information of the landmarks be used more efficiently?
- How can the computational complexity and run time be reduced without compromising the accuracy of the algorithm?

1.2.1. Scope

The research question is broad and can be infinitely complex without any constraints. To focus the results of the thesis to a meaningful realistic situation, the scope is defined by introducing some bounds. Some constraints are defined because going outside that box would have negligible effect on the energy consumption of the algorithm. Other constraints have been defined to keep the simulations tractable, because a Monte Carlo simulation is done for every algorithm. Without these constraints the run time of the algorithms would be too long to complete all simulations during this research. In this thesis the following constraints are used:

Single agent system In this thesis, the environment will be explored by one robot.

Landmarks The number of landmarks for the simulations are up to 100 landmarks, similar to for example trees in a park [24].

Particles The number of particles used for the simulations is up to 1000 particles.

Environment Since the original SLAM solution assumes a static world, this thesis will only focus on non-dynamical objects. So the static world assumption will hold throughout this thesis.

Data association It is assumed that the measurement of a landmark is always associated with the correct landmark. Even though this problem is not arbitrary, there has been done a lot of research to tackle this problem efficiently [22, 34]. Besides that, it is assumed that the research on energy-efficient SLAM can be done independent of this problem.

Known starting position It is assumed that the starting position is known, such that the positions of the robot and landmarks can be defined relative to the starting position. The robot will also not been picked up and placed at an arbitrary location during the exploration. There has been done research on the so called 'kidnapped robot', where the starting position or an intermediate position after relocating the robot to another location also needs to be estimated.

1.3. Organization of this thesis

In the remainder of this thesis the organization is as follows.

In Chapter 2 the background of autonomous navigation is presented. A state space model is presented, which will later be used for the robot model, necessary to solve the SLAM problem. Also the types of inputs of a robot are summarized. The mathematical derivation of SLAM will be given, as well as a short description of the known solutions for SLAM.

Chapter 3 gives an inside in the robot model that is used during the remainder of the thesis and the state-of-the-art of particle filters used for SLAM, known as FastSLAM. A simulation in this chapter of the FastSLAM algorithm will show which parameters play a role in the performance of the algorithm. And finally, some performance measures are presented, which will be used to compare the simulations in the subsequent chapters.

In Chapter 4 the first method is introduced, namely constrained FastSLAM. This method incorporates the information of the landmark observations to the FastSLAM algorithm as spatial constraints. By means of simulations on synthetic datasets, the performance of this method is discussed.

The second method is discussed in Chapter 5, the parallelized particle filter. A graph Laplacian particle filter is used to share information between particles. The performance of this method is also tested using simulations, which are discussed in this chapter as well.

In Chapter 6 the conclusions of both methods are drawn and compared with each other. The shortcomings of the methods are discussed and recommendations to improve the methods are proposed.

2 Autonomous navigation

To design a robot that can navigate autonomously, the design of the algorithm depends on the dynamics of the robot and the type of input sensors. A general framework of an typical autonomous robot will be discussed in this chapter, mainly focused on the sensors of the robot. Besides that, the framework to solve the SLAM problem is explained, including the three popular paradigms to solve the SLAM problem.

2.1. State space model

In general, the autonomous navigation problem for one robot can be written as a state space model, like the following representation.

$$\begin{aligned} \mathbf{x}_k &= f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{v}_k \\ \mathbf{z}_k &= h(\mathbf{x}_k) + \mathbf{e}_k \end{aligned} \tag{2.1}$$

In this representation $\mathbf{x}_k$ is the state of the robot at time k, this can be for example the position, orientation and velocity of the robot. f models the dynamics of the robot and is a function of the previous state and the control input $\mathbf{u}_k$, which is for example the acceleration and steering angle of the robot. Typically, the states of the robot are not measured, but other properties are measured, which are denoted by $\mathbf{z}_k$. h is the measurement model and is a function of the states of the robot. In practice, this model is subject to noise, which is modeled as process noise $\mathbf{v}_k$ and measurement noise $\mathbf{e}_k$, which affect the accuracy of the localization and mapping estimation. These types of noise can have arbitrary properties, but is conventionally assumed to be white Gaussian.

2.2. Inputs

The type of information that is contained in the measurements $\mathbf{z}_k$ and the form of function h depend on the input sensors of the robot. The input sensors can be divided into two types of sensors. The first class of sensors can be used to estimate the relative position with respect to the starting position of the robot, this information is typically encapsulated in the input vector $\mathbf{u}_k$ to calculate the next state of the robot. The second class of sensors are those that can sense the environment, such as cameras or LIDAR, these measurements are denoted by $\mathbf{z}_k$.

2.2.1. Odometry

The first class of sensors are the odometry sensors. For mobile robots, such as wheeled or legged robots, odometry is the use of motion sensors to estimate the position of the robot relative to a known position. This known position can for example be the starting position or known waypoints, such as the charging station. Examples of odometry sensors are rotary encoders or an inertial measurement unit. The early odometry sensors introduced errors due to unequal ground or slip, which typically induced a drift [31, Ch. 4.1]. Consequently, nowadays visual-aided encoders are more commonly used, which compensate for the errors using visual aids [30].

Rotary encoder The change of position can be estimated on the legged joints of a legged robot or wheels of a wheeled/tracked robot by using rotary encoders for example. This can be an encoder that measures the angular position of a shaft or uses the magnetic field of a electric motor to estimate the velocity of the motor. However, this method integrates the velocity to determine the position of the robot, which is sensitive to the introduction of errors [31, Ch. 4.1]. This can only be eliminated using regular calibration of the sensors. Another problem with rotary encoders is that it does not compensate for slip of the wheels or differences in height. This will lead to the believe that the robot is in another position than the actual case.

Visual odometry In case of a less conventional mobile robot, e.g. a combination of legs and wheels on the robot, the rotary encoders may not be usable, so another type of odometry sensors is needed, such as visual odometry. Besides this case, visual odometry also offers a solution to reduce the error of rotary encoders. Most robots already contain some visual sensors, such as cameras or range sensors, which can then also be used to overcome the problem of the inaccuracy of the wheel encoders. Visual odometry uses the images from the environment to match the odometry measurements to get a more precise estimation of the location of the robot. The estimated trajectory computed by visual odometry is more accurate than wheel encoders with a relative position error that ranges from 0.1% to 2% [30].

Since the developments in odometry sensors is continuously increasing, the question remains whether SLAM becomes unnecessary in the future. The answer depends on the application it is needed for. Especially in challenging setups, e.g. no GPS available or low quality visual sensors, SLAM can still provide a better solution than just odometry measurements [1]. Visual odometry only considers local consistency , whereas SLAM considers global consistency [30]. This means that a robot using visual odometry is not able to recognize places it has been before, in SLAM a principle called loop closure is introduced to overcome this problem. This will be further elaborated in subsection 2.3.2.

2.2.2. Perception

The second class of sensors is needed to get a better understanding of the environment. This can serve two goals, the first is to avoid obstacles or avoid collisions with other moving obstacles. The second goal here is to map the environment for exploration.

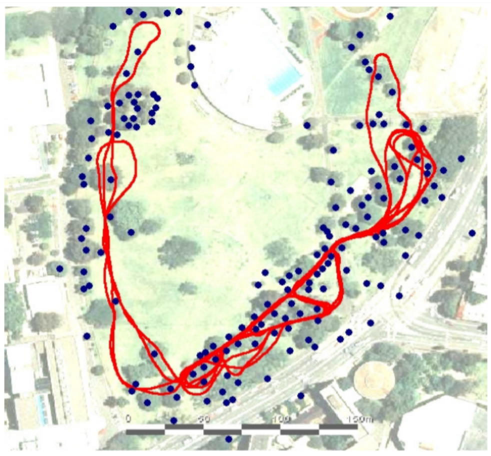
The perception sensors can be roughly divided into groups: range sensors and visual sensors. Range sensors transmit a wave into a medium and measure the time of flight after receiving the echo back. The distance can then be calculated using prior knowledge of the transmitted wave and medium [31, Ch. 4.1]. A relative velocity with other moving obstacles can also be obtained with this type of sensors by doing multiple measurements. The other type of perception sensor, the visual sensors, uses images to sense obstacles or moving objects. This can either be a camera in the visual range, or outside the visual range, depending on the application. This type of sensor is not very useful in dark environments or bright lights, like a sunset [10]. Besides these two classes, other available signals, such as wifi, or sensors, e.g. tactile sensors, can be used for exploration or combined with other sensors to improve the accuracy of the map estimation.

The different classes of sensors can be used for different purposes. For example two robots are identical, except the first uses a laser scanner and the second uses a camera. They are both used to map an office, with similar rooms and corridors. The first robot might often think that two rooms look identical and mark them as the same room, which highly influences the performance of localization and mapping. However, the second robot with a camera might discern those two rooms, from visual cues, such as different paintings on the wall or a different wall color. So the choice of the perception sensors might effect the performance of the map estimation depending on the type of perception sensors.

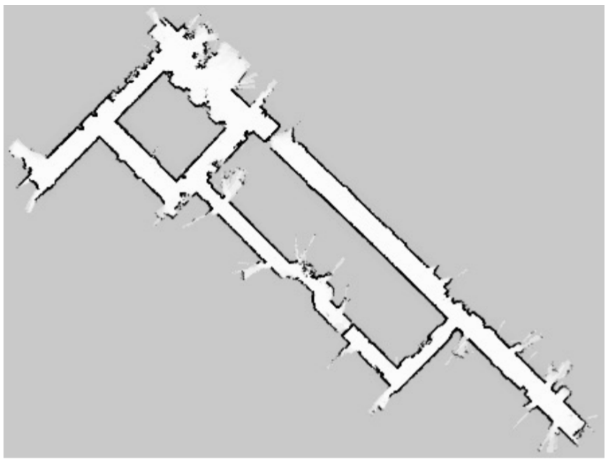
The different types of sensors are compared in Table 2.1.

Table 2.1: Comparison perception sensors for autonomous navigation

Type of sensor	Description	Typical range [15]	Advantages	Challenges
Range sensors				
Sonar/ultrasonic		up to 4m		Not very useful for detection in a narrow angle
RADAR	Transmits radio waves	up to 200m		
LiDAR	Transmits beams of light	up to 200m	more accurate than ultrasonic	
Visual sensors		up to 80m		
Monocular [10]	This set-up only has one camera. It either assumes that bigger objects are closer by or it estimates the distance to an obstacle by the relative movement		Quite easy method to identify objects and classify it	High complexity to compute the distance to an obstacle.
Stereo vision [10]	Set-up of two cameras with known distance between them, which capture each a frame simultaneous.		The distance to an obstacle can be calculated.	
Miscellaneous	Other kind of sensors are also possible, but not commonly used, due to limitations of the techniques			
WiFi [13]			Very helpful for loop closure	Only works if wifi is available and usually combined with other sensors to ensure precision
Tactile [14]	A tactile sensor makes direct contact with the environment	no range	Useful in environments where visual or range sensors are useless, like smoke-filled search-and-rescue operations or covert sensing is required	It takes a lot of time to explore the environment



(a) Feature based map. The red line is the estimated robot trajectory and the blue dots are estimated landmarks in the Victoria Park, in this case trees [34, p. 430].



(b) Occupancy grid map [34, p. 291]. Black squares correspond to occupied cells with a high probability. White squares correspond to a free cell with high probability. The gray cells are still uncertain.

Figure 2.1: Different representations of an output map.

2.3. SLAM

The goal of a SLAM algorithm is to construct a map of the environment and estimate the trajectory of the robot in the map. So the output of the algorithm is a global map, where the representation of the map depends on the environment.

In the case of an outdoor environment with landmarks, such as trees or rocks, a feature-based map is a natural choice, as visualized in Figure 2.1a. This representation is compact and after multiple observations of the same landmark, the estimation improves [34, Ch. 6]. In practical cases, however, a feature detector is needed, which classifies whether an object is a feature or not.

On the other hand, when exploring urban or indoor environments, a grid map (2D) or volumetric map (3D) is a more useful representation, as shown in Figure 2.1b. This representation is useful for larger landmarks, such as building, walls etc. It discretizes the world into cells, where each cell is either occupied, free or not-observed. Therefore a feature detector is not needed. The disadvantage of this technique is that is requires a big memory, which needs to be bigger compared to a feature-based map for large areas.

Constructing the unknown map has two advantages besides exploring the environment itself [1]. First, it improves the performance of the trajectory estimation. The other advantage is that it helps planning the optimal trajectory if autonomous environment exploration is needed. When a new environment needs to be explored autonomously, this is called active SLAM. In contrast with passive SLAM, where the robot either moves randomly or is controlled remotely. Active SLAM normally yields better localization results than passive SLAM, however, requires more autonomy of the robot and thus more computation power [34, Ch. 7.1].

2.3.1. Formulation

To construct a map with a number of unknown landmarks the input sensors of the robot are used. Prior knowledge of the location or type of landmark is not necessary. Simultaneously, the trajectory of the robot is estimated. The formulation to solve this problem is defined as follows [11]:

- $\mathbf{x}_k$ is the state vector at time step k containing the pose (both the location and the orientation) of the robot. Other states of interest might also be included in this vector, such as the velocity of the robot or parameters needed for calibration of the sensors.
- $\mathbf{u}_k$ is the control vector, this is the input applied at time $k-1$ to the robot, which has driven it to state $\mathbf{x}_k$ at time step k.

- $\mathbf{m}_i$ is a vector containing the true location of the ith landmark.
- $\mathbf{z}_{ik}$ is an observation of landmark i by the perception sensor(s) of the robot at time step k. This is either the distance from the sensor on the robot to a landmark or the relative distance between two landmarks.

These vectors are commonly stacked into the following matrices:

- $\mathbf{X}_{0:k} = [\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k]$, in other words the history of the vehicle locations. This information needs to be estimated.
- $\mathbf{U}_{1:k} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_k]$, in other words, the history of the control inputs.
- $\mathbf{m} = [\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_n]$, which is the information of all landmarks. If there is no prior information about the landmarks, the information in this matrix and the dimensions of the matrix are unknown and need to be estimated.
- $\mathbf{Z}_{1:k} = [\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_k]$, which is the information of all landmark observations. The individual observation $\mathbf{z}_k$ is a stacked vector of all landmark observations $\mathbf{z}_{ik}$.

To solve the SLAM problem three paradigms exist, a Kalman filter (KF), FastSLAM based on a particle filter and graph-based SLAM. These paradigms differ in the filter design to represent the robot, but the theory behind these paradigms is for all three the same. Due to inherent noise in both the sensors of the odometry and the observations, the solution of SLAM is formulated using the probability distribution; the distribution of the robot's trajectory and map of the environment, given the observations and the controls:

$$p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k}, \mathbf{U}_{1:k}, \mathbf{x}_0), \tag{2.2}$$

this is the probability that the robot at time k is in a certain state $\mathbf{x}_k$ in a map $\mathbf{m}$ given the observations $\mathbf{Z}_{1:k}$ and input controls $\mathbf{U}_{1:k}$ upto time step k and the starting pose $\mathbf{x}_0$. This distribution needs to be computed for all time steps k, where it is assumed that the starting pose $\mathbf{x}_0$ is known and certain. This is split into two parts: the observation (or measurement) model and the motion model. The observation model assumes that the pose of the robot and the location of the landmarks are known and can be described in the following form:

$$p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) \tag{2.3}$$

On the other hand, the motion model only considers the previous state and the current input, so it is a state transition, which is assumed to be a Markov process. In a Markov process, the future states only depend on the past through the current state. So it can be described in the following form:

$$p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{2.4}$$

The solution to this problem is implemented in a recursive two-step form. The first step is a time-update, a prediction of Equation 2.2. The second step is the correction step, which uses the observations $\mathbf{z}_k$ of the robot at time step k.

Time update

$$p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k-1}, \mathbf{U}_{1:k}, \mathbf{x}_0) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) p(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{1:k-1}, \mathbf{U}_{1:k-1}, \mathbf{x}_0) d\mathbf{x}_{k-1} \tag{2.5}$$

Measurement update

$$p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k}, \mathbf{U}_{1:k}, \mathbf{x}_0) = \frac{p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k-1}, \mathbf{U}_{1:k}, \mathbf{x}_0)}{p(\mathbf{z}_k|\mathbf{Z}_{1:k-1}, \mathbf{U}_{1:k})} \tag{2.6}$$

Since the information of the robot pose and the landmark observations is noisy, the exact locations are hard to estimate. However, the relative location between two measured landmarks is known with high accuracy if they are observed at the same time step. So the error in the location of the landmarks are highly correlated [11]. An important realization in the solution of SLAM is that the correlations between the estimation of landmarks increase monotonically [8]. Thus, the map converges to the real map by

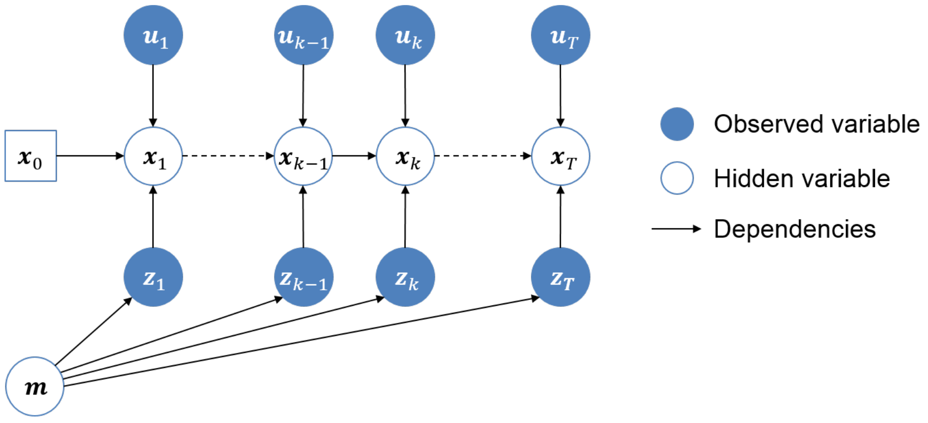


Figure 2.2: Dynamic Bayesian Network of the SLAM process. Each node represents a random variable and an edge represents the conditional dependence between the two nodes. The colored circular nodes are the measurements and the white circular nodes are the hidden variables that need to be estimated. [16]

obtaining more observations at different locations, which are considered as nearly independent measurements.

To solve the SLAM problem two important assumptions are made, the assumption that the robot is a Markov process and the static world assumption [1]. The second assumption assumes that there are no moving obstacles in the world. This assumption is hard to fulfill in urban environments or warehouses, tackling this problem is a big research topic [17, 20, 35], but is outside the scope of this thesis.

The solutions to the SLAM problem can be divided into two classes: online SLAM and full SLAM.

Online SLAM The approach to solve online SLAM estimates only the most recent location and orientation of the robot:

$$p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k}, \mathbf{U}_{1:k}, \mathbf{x}_0) \tag{2.7}$$

With the Markov assumption and static world assumption, the complexity of online SLAM can be decreased, which results in a particular structure named dynamic Bayesian network (DBN), which is visualized in Figure 2.2. This structure describes a stochastic process as a directed graph. The known variable $\mathbf{x}_0$ is positioned in a square. There is a circular node for each random variable, the blue nodes are the observed variables (the measurements of the landmarks $\mathbf{z}_k$ and the odometry measurements $\mathbf{u}_k$), the white circular nodes are the hidden variables $\mathbf{x}_k$ and $\mathbf{m}$, which need to be estimated. Between the nodes the directed edges indicate a dependence between two nodes. This model is in particular used for filtering methods [11], such as the Kalman filter (discussed in subsection 2.4.1) or a particle filter (discussed in subsection 2.4.2).

Full SLAM In a full SLAM algorithm the entire trajectory of the robot is estimated at once:

$$p(\mathbf{X}_{0:T}, \mathbf{m}|\mathbf{Z}_{1:T}, \mathbf{U}_{1:T}, \mathbf{x}_0), \tag{2.8}$$

where T is the final time step of the robot. This type of SLAM is used in graph-based SLAM (discussed in subsection 2.4.3).

So the difference between the two methods is that the trajectory and map construction in online SLAM is done during the exploration, while the construction in full SLAM is done after the exploration is finished. A full SLAM solution requires an inversion of the whole information matrix, which requires a comparatively big computational load at the end of the exploration. This matrix contains the x- and y-positions of the whole trajectory (T) of the robot and the x- and y-positions of the observed landmarks (M) on the diagonal, so it is a $(2T + 2M) \times (2T + 2M)$ matrix. To inverse this matrix a computational complexity of $\mathcal{O}((2T + 2M)^3)$ is required.

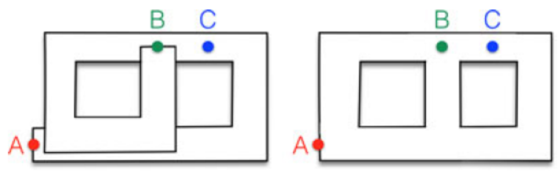


Figure 2.3: Left: map constructed from odometry data. Right: map build from SLAM [1]

2.3.2. Loop closure

The concept of loop closure ensures that the robot does not think it will drive through an endlessly long corridor [1]. Using the measurements SLAM ensures that a loop is detected and implemented in the estimated map. This principle is illustrated in Figure 2.3. The principle is inherently different than the visual odometry as described in Section 2.2.1, since in both cases the observations are not stored, so the exact observations cannot be matched with previous observations to predict whether a location has already been observed before. A SLAM algorithm associates measured data with already observed data to see if a feature has already been seen before, this is called data association. If the robot recognizes a feature it has returned to, it closes the loop, which results in a reduction of the uncertainty in the map estimation. Without loop closure the uncertainty of the exploration only increases. In this thesis the landmark observations are associated with certainty to the right landmark. A lot of research has already been done on data association and loop closure [38, 39], but will be considered outside the scope of this thesis.

2.4. Filter design for SLAM

To solve the SLAM problem an optimal filter can be designed. This filter estimates the state of the time-varying system, where the states are indirectly observed by noisy measurements. The states should be estimated using the observed measurements. In a Bayesian framework, this means that the posterior distribution in (2.9) needs to be computed [33, Ch. 4].

$$p(\mathbf{X}_{0:T}, \mathbf{m}|\mathbf{Z}_{1:T}, \mathbf{U}_{1:T}, \mathbf{x}_0) = \frac{p(\mathbf{Z}_{1:T}|\mathbf{X}_{0:T}, \mathbf{m})p(\mathbf{X}_{0:T}, \mathbf{m})}{p(\mathbf{Z}_{1:T})}, \tag{2.9}$$

where

- $p(\mathbf{X}_{0:T}, \mathbf{m})$ is the prior distribution defined by the dynamic model and the available landmark information,
- $p(\mathbf{Z}_{1:T}|\mathbf{X}_{0:T}, \mathbf{m})$ is the likelihood model for the measurements given the information about the trajectory and the landmarks,
- $p(\mathbf{Z}_{1:T})$ is the normalization constant defined as

$$p(\mathbf{Z}_{1:T}) = \int p(\mathbf{Z}_{1:T}|\mathbf{X}_{0:T}, \mathbf{m})p(\mathbf{X}_{0:T}, \mathbf{m})d\mathbf{X}_{0:T} \tag{2.10}$$

Solving this full posterior distribution every time a new observation comes in requires a lot of computational power. One way to solve this problem is to only compute this posterior distribution after the exploration of a whole map, which is done in full SLAM. How this is solved will be discussed in Section 2.4.3. The second solution to this computational problem is to relax the full posterior distribution, this is done in online SLAM. A solution of online SLAM with a closed form solution is the Kalman Filter (KF). The Kalman Filter is a method to solve a linear Gaussian filtering problem.

However, if a closed form solution is not available, because the problem is not linear or the noise is not Gaussian for example, optimal filtering equations are computationally intractable. Other methods then approximate the closed form solution. The extended Kalman Filter or unscented Kalman Filter are approximations based on the Kalman Filter that can deal with non-linear and non-Gaussian models. How these methods can solve SLAM problems is discussed in Section 2.4.1.

Type	Linearization	Parametrization
EKF	Taylor approximation	moments
UKF	Unscented transformation	moments
EIF	Taylor approximation / unscented transformation	canonical

Table 2.2: Comparison between Kalman filter algorithms

Another filtering technique is based on a particle filter, also called a sequential Monte Carlo method. This method represents the distribution as weighted samples, so this method can be used for any arbitrary distribution [33, Ch. 4]. This method is also used to solve SLAM problems, which is discussed in more detail in Section 2.4.2 and Chapter 3. In the next three sections the three paradigms will be discussed in more detail and will be concluded in Section 2.4.4.

2.4.1. Kalman filter

The first solution to the online SLAM problem is the Kalman filter, which is a Bayesian filter. It assumes that both the process noise on the motion model and the measurement noise on the observation model is Gaussian distributed [33, 34]. The most commonly used Kalman filter for SLAM is the extended Kalman filter (EKF), which accounts for non-linear functions by local linearization using the Taylor expansion. An improved approximation for non-linear models can be achieved by using the unscented Kalman filter (UKF), which makes use of the unscented transformation instead of the Taylor expansion [34, Ch. 7.7]. However, computationally speaking, this algorithm is slower than the EKF. A third method is the extended information filter (EIF), which uses the information matrix (inverse of covariance matrix) and information vector instead of the covariance matrix and mean vector [34, Ch. 12]. The information vector and information matrix are called the canoncial parametrization, in contrast to the moments, which contain the mean and covariance. Conversion between the moments and canonical matrices are expensive, so this makes the EKF more popular for application compared to the EIF.

The different types of Kalman filters are summarized in Table 2.2 and will be compared with the other paradigms in Section 2.4.4, to be able to choose one of the algorithms as a baseline for the proposed algorithm.

2.4.2. Particle filter

A particle filter is a nonparametric implementation of a Bayes filter to approximate the posterior by a finite number of samples, also called particles. The key idea is to represent the posterior distribution by a set of randomly drawn samples. Every particle predicts the current state given the previous state and input, so every particle represents an estimation of the state of the robot. After the prediction step every particle updates its prediction with a likelihood weight that represents the probability of being correct. This update step uses the odometry and perception measurements to generate a probability density function, where the likelihood weights are derived from. All particles together represent the distribution of the estimated state, whereas with an increasing number of particles, the actual distribution is depicted more closely. To avoid particle degeneracy resampling can be used. This method can deal with any arbitrary posterior distribution. Another advantage of this method is its ability to represent nonlinear models [34, Ch. 13].

This methodology has already been used a lot for SLAM, where it is referred to as FastSLAM. FastSLAM is discussed in more detail in Chapter 3.

2.4.3. Graph-based

Graph-based SLAM is a full SLAM solution, which means that the map and trajectory estimation is done after the whole exploration has been finished. In this case, the SLAM problem is represented by a graph, where the nodes represent the poses of the robot at different time steps and the links between them represent constraints between the nodes. These constraints are introduced by the odometry and perception measurements. The goal of this method is to build the map, represented by a graph, that minimizes the error that is introduced by the constraints [16]. The problem can be translated to a leastsquares problem.

This method makes no distinction between the motion and observation model, unlike online SLAM. In this method, the problem is divided into two parts; the front-end and the back-end. The front-end interprets the observations made by the sensors and does some data association, such as loop closure. This part is thus based on the kind of sensor used and is also called graph construction. The back-end then uses this abstract data to estimate the states of the robot and the location of the obstacles. This part is sensor agnostic and determines the most likely graph given the constraints of the front-end, so this part is called the graph optimization. This estimation is typically done using the Maximum a Posteriori (MAP) estimation. In case there is no prior knowledge available about the pose of the robot and the location of the landmarks, this MAP estimation reduces to a Maximum Likelihood Estimation (MLE) [16].

2.4.4. Comparison

These three algorithms have their own advantages and drawbacks, also depending on the type of environment and application. They will be compared in this section.

The advantage of a Kalman filter is the relatively easy implementation of a linear or linearized system with Gaussian noise, especially if the features are distinct. However, the robustness of the EKF is low, since a faulty data association will result in an error in the future. Furthermore the computational complexity increases cubic with the number of landmarks.

A particle filter is less sensitive to wrong data associations due to the particle based algorithm. The disadvantage of a particle filter is the computational load of the algorithm [33, Ch. 7]. However, for the SLAM application a so called Rao-Blackwellized particle filter (RBPF) can be used to reduce this load. This will be further discussed in Section 3.3.

Also graph-based SLAM is more robust to wrong data associations, since it can revisit data associations and reexamine them [1]. For environments with many landmarks, this method requires a big computational load and memory, since the stored data of the landmarks needs to be inverted, so this method is in particular interesting if the robot only explores the environment and a separate computer will do the computations afterwards.

The three paradigms are summarized in Table 2.3, where the computational efficiency for all three methods are shown. The computational complexity of all methods depends on the number of landmarks (M). The computation of the particle filter also depends on the number of particles (N). In Figure 2.4 the complexity is visualized as function of the number of landmarks. This shows that for a large amount of landmarks a particle filter is most efficient.

Table 2.3: Comparison between the three paradigms for SLAM [26]

Method	**Pros**	**Cons**	**Efficiency** [34]	**References**
Kalman filter	Works well when features are distinct.	Adding new features requires quadratic time.	$\mathcal{O}(M^3)$	Durrant-Whyte and Bailey [11]
Particle filter	Adding features only requires logarithmic time and there is no dependence on parametrization of motion model.	Loop closure performance depends highly on the particle set.	$\mathcal{O}(Nlog(M))$	Durrant-Whyte and Bailey [11]
Graph based	Also previous poses of the robot are updated for post-processing.	Computationally more expensive.	$\mathcal{O}(M^2)$	Cadena et al. [1]

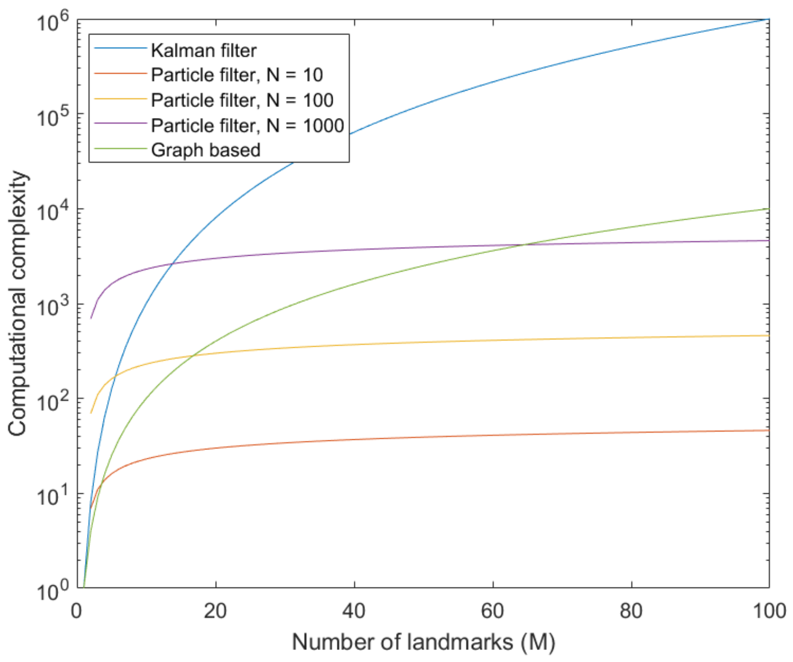


Figure 2.4: Computational complexity for different solutions of SLAM; Kalman filter, particle filter and Graph-based SLAM as a function of the number of landmarks.

3
FastSLAM

In this thesis two improved algorithms based on a particle filter SLAM method are proposed to explore an unknown environment using an autonomous robot that is energy-efficient. In this chapter the robot models that will be used, will be discussed, as well as the FastSLAM algorithm that will be used as a baseline.

The motivation for choosing a particle filter will be discussed in the next section, after which a robot model will be introduced. This model entails the dynamics of the robot and the physics of the measurement model, which is both needed to solve the SLAM problem. In the third section the mathematical derivation of a particle filter is shown, which will be used for the simulation in the next section. This chapter concludes with a section about the performance metrics that are used to measure how good the proposed algorithms will perform compared to FastSLAM.

3.1. Motivation

SLAM can be used for active exploration of new environments, for example in caves, on the moon or Mars. It is also often used indoors for vacuum cleaners or warehouse robots. To achieve the goals of these robots, the autonomous robot needs active SLAM to map the environment and localize itself. These applications yield for an online SLAM solution, since the autonomous navigation and localization can be done simultaneously on the robot. The online SLAM solutions are either an extended Kalman filter or a particle filter.

An advantage of FastSLAM over EKF-SLAM and graph-based SLAM is the lower computational complexity of the algorithm with an increasing number of landmarks, as shown in Table 2.3. The high computational complexity of EKF-SLAM is due to the enormous measurement updates every time step for all landmarks that have been observed so far [34, Ch. 10]. In FastSLAM these measurement updates are only done for the observed landmarks, which requires a smaller computational complexity. This difference in update step yields for a solution with FastSLAM to build an energy-efficient SLAM algorithm.

The first idea to improve SLAM with a particle filter is to incorporate the landmark observations in the trajectory as spatial constraints. The robot is namely not controlled remotely and need to be able to plan its own trajectory. To plan this path, the robot needs to avoid obstacles, which may lead to an advantage for the localization problem. The robot can use the information about landmarks to improve the performance of the SLAM algorithm. To do so, spatial constraints can be added to the SLAM algorithm. However, this leads to non-Gaussian posterior distribution. Since a particle filter can better deal with any arbitrary distribution compared to the EKF, the FastSLAM algorithm is used as starting point for this improved constrained problem.

A disadvantage of FastSLAM is the memory needed for the number of particles [34, Ch. 13]. Since all particles store their own map, too many particles can cause problem in memory load. On the other

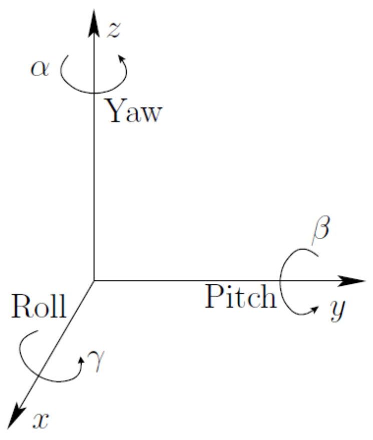


Figure 3.1: Three-dimensional rotations described in roll, pitch, yaw [19, p. 97].

hand, too little particles will not give an accurate map of the environment as result. This drawback will be discussed in more detail in Section 3.5.5.

The output map of the algorithm does not depend significantly on the algorithm of a SLAM problem. For all algorithms either a feature based map or a grid map can be used as map representation. Since a feature based map requires less computation and less memory of the SLAM algorithm, this type of map is used.

3.2. Robot model

As described in Section 2.1, a model for a mobile robot can be written as the state-space model in (2.1). This model can be separated into a motion model, describing the dynamics of the robot, and an observation model, describing the perception measurements of the robot. Both models are needed to solve the SLAM problem and will be discussed in more detail in the next sections.

3.2.1. Motion model

The dynamics of the robot are modeled in the motion model, where the robot obeys the following dynamics:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{v}_k \tag{3.1}$$

where the state vector $\mathbf{x}_k$ of the robot contains the x-position, y-position, z-position and the orientation γ, β, α with respect to the three axes, as denoted in Figure 3.1. So the state vector can be written as (3.2). These states can be controlled by the actuators of the robot. The input vector is written as velocities in four degrees of freedom; the linear velocity v and the angular velocities in every orientation $\dot{\gamma}$, $\dot{\beta}$ and $\dot{\alpha}$. The assumption is that the velocities are constant during a time period. The input vector can be written as (3.3).

$$\mathbf{x}_k = \begin{pmatrix} x_k & y_k & z_k & \gamma_k & \beta_k & \alpha_k \end{pmatrix}^\top \tag{3.2}$$

$$\mathbf{u}_k = \begin{pmatrix} v & \dot{\gamma} & \dot{\beta} & \dot{\alpha} \end{pmatrix}^\top \tag{3.3}$$

In the simulations, the robot is assumed to be holonomic. Mathematically, this means that the dynamics of a robot can be described without nonholonomic kinematic constraints. A nonholonomic kinematic constraint requires a differential relationship, such as the velocity. Informally, this means that the robot can move sideways and can reach any point in space in any way in any orientation [31, pp. 75-77]. This assumption is not really realistic, but will give a good inside in the operation of the algorithm without getting lost in the details of the robot dynamics.

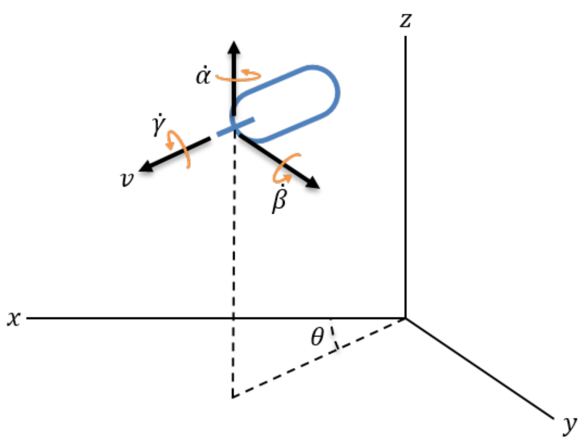


Figure 3.2: 3D motion model of robot, where the linear velocity v and angular velocities $\dot{\gamma}$, $\dot{\beta}$ and $\dot{\alpha}$ are denoted.

The function $f(\cdot)$ in (3.1) is a linear function that translates the linear and angular velocities into the robot states. The next state can be written as a function of the previous state and inputs as shown in (3.4).

$$\begin{pmatrix} x_k \\ y_k \\ z_k \\ \gamma_k \\ \beta_k \\ \alpha_k \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \\ z_{k-1} \\ \gamma_{k-1} \\ \beta_{k-1} \\ \alpha_{k-1} \end{pmatrix} + \begin{pmatrix} \cos(\theta)\cos(\beta) & 0 & 0 & 0 \\ \sin(\theta)\cos(\beta) & 0 & 0 & 0 \\ \sin(\beta) & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{pmatrix} T + \mathbf{v}_k \tag{3.4}$$

where T is the sampling time [21].

The process noise of the linear and angular velocity is assumed to be white Gaussian $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$, where the correlation between the linear and angular velocities is assumed to be zero [31]. So the covariance matrix $\mathbf{Q}$ looks like (3.5).

$$\mathbf{Q} = \begin{pmatrix} \sigma_v^2 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{\gamma}}^2 & 0 & 0 \\ 0 & 0 & \sigma_{\dot{\beta}}^2 & 0 \\ 0 & 0 & 0 & \sigma_{\dot{\alpha}}^2 \end{pmatrix} \tag{3.5}$$

where $\sigma_{\{\cdot\}}$ is the standard deviation of the velocity.

3.2.2. Observation model

The measurements of the perception sensors are modeled in the observation model of the robot [28], which looks like (3.6)

$$\mathbf{z}_{ik} = h(\mathbf{x}_k) + \mathbf{e}_k, \tag{3.6}$$

where

$$h(\mathbf{x}_k) = \begin{pmatrix} r_{ik} \\ \phi_{ik} \end{pmatrix} = \begin{pmatrix} ||(x_k, y_k)||_2 \\ \tan^{-1}(x_k, y_k) \end{pmatrix} \tag{3.7}$$

Both the distance r_{ik} and the bearing ϕ_{ik} from the robot to a landmark i is measured at a time step k. Using a range sensor, these metrics are the output of the sensor measurements. If a camera is used, it is assumed that this is the output after the image processing, which is outside the scope of this report. The measurement noise is assumed to be white Gaussian $\mathbf{e}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ with covariance matrix:

$$\mathbf{R} = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{pmatrix}, \tag{3.8}$$

where σ_r is the standard deviation of the distance measured from the sensor to the landmark and σ_ϕ is the standard deviation of the bearing measured from the sensor to the landmark. These parameters are sensor dependent, so are constant over time.

3.3. Particle Filter

The robot model can then be incorporated in the algorithm to solve the SLAM problem, in this case a particle filter SLAM. A particle filter is a Monte Carlo method. These types of methods refer to methods where samples (also called particles) are drawn from a distribution to estimate the state by certain quantities that the samples contain. In a perfect approximation N independent particles are drawn from a distribution $\mathbf{x}^{(j)} \sim p(\mathbf{x}_k|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k), \quad j = 1, ..., N$. However, in general it is not possible to draw samples from the distribution p, due to the arbitrary form of this distribution. A solution to this problem is to propose a known importance distribution $\pi(\mathbf{x}_k|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k)$, from which samples can easily be drawn [33]:

$$\mathbf{x}_k^{(j)} \sim \pi(\mathbf{x}_k|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k), \quad j = 1, \ldots, N \tag{3.9}$$

The particles are then weighted with the target (true) distribution. This weight represents the likelihood to what extent the estimated state is correct. The weight is then considered as the fraction between the target distribution and the proposal distribution:

$$w_k^{(j)} = \frac{\text{target}(\mathbf{x}_k^{(j)})}{\text{proposal}(\mathbf{x}_k^{(j)})} = \frac{p(\mathbf{x}_k^{(j)}|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k)}{\pi(\mathbf{x}_k^{(j)}|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k)} \tag{3.10}$$

This method is known as importance sampling (IS) and is visualized in Figure 3.3. The target (blue) distribution is an arbitrary distribution. There will be sampled from the (red) proposal distribution, which is a Gaussian distribution, commonly chosen as proposal distribution.

When this importance sampling is done every time step, a sequential version of the importance sampling is used, so called sequential importance sampling (SIS). At time step k a weight $w_k^{(j)}$ is computed for each particle $j = \{1, \ldots, N\}$, which is then used in the next time step to compute an updated weight. In this case a particle is represented as a weighted set $\mathcal{X} = \{(w_k^{(j)}, \mathbf{x}_k^{(j)}) : j = 1, \ldots, N\}$. The algorithm for SIS in the SLAM problem is shown in Algorithm 3.1.

Algorithm 3.1 Sequential Importance Sampling [33]

```
1: Draw $N$ samples $\mathbf{x}_0^{(j)}$ from the prior and set weight to $1/N$
2: for $j = 1$ to $N$ do
3:     $\mathbf{x}_0^{(j)} \sim p(\mathbf{x}_0)$
4:     $w_0^{(j)} = 1/N$
5: end for
6: At every time step draw a sample from proposed distribution
7: for $k = 1$ to $T$ do
8:     $\mathbf{x}_k^{(j)} \sim \pi(\mathbf{x}_k|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k), \quad j = 1, \ldots, N$
9:     $w_k^{(j)} \propto w_{k-1}^{(j)} \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(j)})p(\mathbf{x}_k^{(j)}|\mathbf{x}_{k-1}^{(j)})}{\pi(\mathbf{x}_k^{(j)}|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k)}$
10: end for
```

While using the sequential importance sampling principle almost all particles will converge to a weight of nearly zero. This degeneracy problem can be solved by resampling the particles. In this procedure N new samples are drawn from the old particle set $\mathcal{X}$, where the weight of a particle represents the probability it will be redrawn. The old set is then replaced by the new set of N particles with weight $1/N$. This procedure is not necessarily done every time step, but might depend on various requirements. This will be discussed in more detail in Section 3.3.2.

3.3.1. Rao-Blackwellized particle filter

The SLAM problem is a high dimensional problem, which makes it computationally infeasible to directly implement a particle filter [11]. To improve the efficiency of a particle filter as discussed in Section 2.4.2 Rao-Blackwellization can be used. If possible, some of the filtering equations can be computed analytically and others by sampling. This reduces the computational complexity and improves the performance of a particle filter, since an estimator with less variance can be achieved [33].

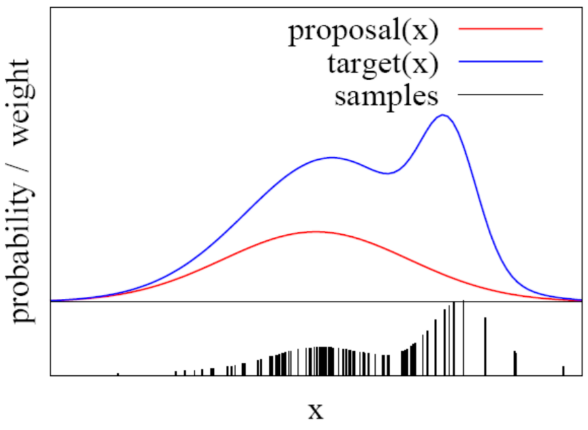


Figure 3.3: Importance sampling for particle filter. Samples are drawn from the proposal distribution π (red line) and weighted with the target distribution p (blue line). The weight of the sample is the importance $(p(x_k^{(j)})/\pi(x_k^{(j)}))$. The weighted samples are shown in the lower half part of the figure,where the weight of the samples is represented by the height of the sample. [34, p. 101]

To achieve this increase in performance, a part of the distribution needs to be calculated analytically. To be able to separate the problem into two parts, (2.9) can be partitioned according to the product rule [33]:

$$p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k}, \mathbf{U}_{1:k}, \mathbf{x}_0) = p(\mathbf{m}|\mathbf{X}_{0:k}, \mathbf{Z}_{1:k})p(\mathbf{X}_{0:k}|\mathbf{Z}_{1:k}, \mathbf{U}_{1:k}, \mathbf{x}_0) \tag{3.11}$$

The Rao-Blackwell-theorem states that if the second term $p(\mathbf{X}_{0:k}|\mathbf{Z}_{1:k}, \mathbf{U}_{1:k}, \mathbf{x}_0)$ can be represented analytically, only the first term $p(\mathbf{m}|\mathbf{X}_{0:k}, \mathbf{Z}_{1:k})$ needs to be sampled.

This Rao-Blackwellized particle filter is commonly used to solve the SLAM problem and is the basis for the FastSLAM algorithm, explained next.

3.3.2. Algorithm

The FastSLAM algorithm consists of 3 steps:

- Sampling from proposal distribution
- Weighting with measured distribution
- Resampling of particles

As discussed in Section 2.4.2, the jth particle maintains information on the estimated state of the robot ($\hat{\mathbf{x}}_k^{(j)}$) as a weighted ($w_k^{(j)}$) set $\mathcal{X}$. Besides the weight and estimated state, the estimated landmark information is also stored in every particle, so every particle maintains its own map. This information is stored as a 2×2 EKF, which means that the estimated x- and y-position as mean $\boldsymbol{\mu}$ and the covariance $\boldsymbol{\Sigma}$ are stored for every observed landmark i. In conclusion a particle j maintains the following information in a set at time step k:

$$\mathcal{X}_k = \{(w_k^{(j)}, \hat{\mathbf{x}}_k^{(j)}, \hat{\boldsymbol{\mu}}_k^{(j)}, \boldsymbol{\Sigma}_k^{(j)}) : \quad j = 1, \dots, N\} \tag{3.12}$$

The FastSLAM algorithm is shown in Algorithm 3.2. The sensor model $h(\cdot)$ is described in (3.7) and the covariance matrix $\mathbf{R}$ is defined in (3.8).

It is assumed that the robot knows whether it has detected a landmark before or not, so there is no data association needed. In the case that it has detected a new landmark i, the robot initializes the mean ($\boldsymbol{\mu}_{ik}^{(j)}$) and covariance matrix ($\boldsymbol{\Sigma}_{ik}^{(j)}$) for every particle, which is described in Algorithm 3.3, with the mean and Jacobian as described in (3.13) and (3.14) respectively, where h^{-1} is the inverse of the observation model h and h' is the derivative of the observation model h with respect to the estimated

Algorithm 3.2 FastSLAM [23, 34]

```
1: Initialization starting point robot $\mathbf{x}_0$, time step $k = 0$
2: $\mathcal{X}_0$ = $\{w^{(j)}, \hat{\mathbf{x}}^{(j)}, \hat{\boldsymbol{\mu}}^{(\mathbf{j})}, \boldsymbol{\Sigma}^{(j)}\} = \{1/N, \mathbf{x}_0, \mathbf{0}, \mathbf{0}\}, \quad j = 1, \ldots, N$    ▷ Initialize the first particle set
3: while true do
4:     for $j := 1$ to $N$ do    ▷ For all particles
5:         $\hat{\mathbf{x}}_k^{(j)} \sim p\left(\mathbf{x}_{k-1}^{(j)}, \mathbf{u}_k\right)$    ▷ Sample new robot pose $\mathbf{x}_k^{(j)}$
6:         $\mathbf{z}_{ik}^{(j)} = h\left(\mathbf{x}_k^{(j)}\right)$    ▷ Observe landmarks
7:         if $\mathbf{z}_{ik}^{(j)}$ is never observed before then
8:             add_new_landmark$\left(\mathbf{z}_{ik}^{(j)}, \mathbf{x}_k^{(j)}, \mathbf{R}\right)$
9:         else
10:            update_landmark$\left(\boldsymbol{\mu}_{i,k-1}^{(j)}, \boldsymbol{\Sigma}_{i,k-1}^{(j)}, \mathbf{x}_k^{(j)}, \mathbf{z}_{ik}^{(j)}, \mathbf{R}\right)$
11:        end if
12:    end for
13:    $\mathcal{X}_k$ = Resample($\{w^{(j)}, \hat{\mathbf{x}}_k^{(j)}, \hat{\boldsymbol{\mu}}_k^{(j)}, \boldsymbol{\Sigma}_k^{(j)}\}_{j=1,2,\ldots,N}$)    ▷ Resample particles
14:    $k = k + 1$
15: end while
```

landmark position $\boldsymbol{\mu}$ and the estimated robot pose $\mathbf{x}$.

$$\boldsymbol{\mu}_{ik}^{(j)} = h^{-1}(\mathbf{z}_{ik}^{(j)}, \mathbf{x}_k^{(j)}) = \begin{pmatrix} \hat{x}_k^{(j)} + r_{ik} \cdot \cos(\phi_{ik}) \\ \hat{y}_k^{(j)} + r_{ik} \cdot \sin(\phi_{ik}) \end{pmatrix} \tag{3.13}$$

$$\mathbf{H} = h'(\boldsymbol{\mu}_{ik}^{(j)}, \mathbf{x}_k^{(j)}) = \begin{pmatrix} \cos(\phi_{ik}) & -r_{ik} \cdot \sin(\phi_{ik}) \\ \sin(\phi_{ik}) & r_{ik} \cdot \cos(\phi_{ik}) \end{pmatrix} \tag{3.14}$$

Algorithm 3.3 Add new landmark

```
1: function add_new_landmark($\mathbf{z}_{ik}^{(j)}, \mathbf{x}_k^{(j)}, \mathbf{R}_k$)
2:     $\boldsymbol{\mu}_{ik}^{(j)} = h^{-1}(\mathbf{z}_{ik}^{(j)}, \mathbf{x}_k^{(j)})$    ▷ Initialize mean
3:     $\mathbf{H} = h'(\boldsymbol{\mu}_{ik}^{(j)}, \mathbf{x}_k^{(j)})$    ▷ Calculate Jacobian
4:     $\boldsymbol{\Sigma}_{ik}^{(j)} = \mathbf{H}^{-1}\mathbf{R}_k(\mathbf{H}^{-1})^\top$    ▷ Initialize covariance
5:     return $\boldsymbol{\mu}_{ik}^{(j)}, \boldsymbol{\Sigma}_{ik}^{(j)}$
6: end function
```

In case the robot has seen the landmark before, the landmark is updated for every particle. This update step also uses a Jacobian matrix :

$$\mathbf{H} = \begin{pmatrix} \frac{\Delta x}{d} & \frac{\Delta y}{d} \\ -\frac{\Delta y}{d^2} & \frac{\Delta x}{d^2}, \end{pmatrix} \tag{3.15}$$

where Δx and Δy are the x- and y-distances from the robot to the observed landmark and d is the distance between the estimated robot pose and estimated landmark position. The update step is described in Algorithm 3.4

Resampling is not performed every time step, but only when needed. The criteria for when it is needed depends on the implementation of the SLAM algorithm. One method of resampling is to resample every ith time step, with a fixed i. This method is unbiased, but might be computationally inefficient, since it is not always needed. A second method is adaptive resampling, where the number of effective particles determines whether resampling is needed. This number is calculated as:

$$N_{eff} = \frac{1}{\sum_{j=1}^{N} \left(w_{nom,k}^{(j)}\right)^2} \tag{3.16}$$

where the weights $w_{nom,k}^{(j)}$ are the normalized weights such that the weights of all particles sum up to 1. Resampling is then performed if this number is lower than a predefined threshold, for example $N_{eff} < 0.75N$ [33]. The algorithm of resampling is shown in Algorithm 3.5.

Algorithm 3.4 Update landmarks

```
 1: function update_landmark($\boldsymbol{\mu}^{j}_{i,k-1}, \boldsymbol{\Sigma}^{(j)}_{i,k-1}, \mathbf{x}^{(j)}_{k}, \mathbf{z}^{(j)}_{ik}, \mathbf{R}$)
 2:     $\hat{\mathbf{z}}^{(j)} = h(\boldsymbol{\mu}^{(j)}_{i,k-1}, \mathbf{x}^{(j)}_{k})$        ▷ Predict measurement
 3:     $\mathbf{H} = h'(\boldsymbol{\mu}^{(j)}_{i,k-1}, \mathbf{x}^{(j)}_{k})$
 4:     $\mathbf{R} = \mathbf{H}\boldsymbol{\Sigma}^{(j)}_{i,k-1}\mathbf{H}^{\top} + \mathbf{R}$
 5:     $\mathbf{K} = \boldsymbol{\Sigma}^{(j)}_{i,k-1}\mathbf{H}^{\top}\mathbf{R}^{-1}$
 6:     $\boldsymbol{\mu}^{(j)}_{ik} = \boldsymbol{\mu}^{(j)}_{i,k-1} + \mathbf{K}(\mathbf{z}_{ik} - \hat{\mathbf{z}}^{(j)})$
 7:     $\boldsymbol{\Sigma}^{(j)}_{ik} = (\mathbf{I} - \mathbf{K}\mathbf{H})\boldsymbol{\Sigma}^{(j)}_{i,k-1}$
 8:     $w^{(j)} = |2\pi\mathbf{R}|^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{z}_{ik} - \hat{\mathbf{z}}^{(j)})^{\top}\mathbf{R}^{-1}(\mathbf{z}_{ik} - \hat{\mathbf{z}}^{(j)})\right]$
 9:     return $\boldsymbol{\mu}^{(j)}_{ik}, \boldsymbol{\Sigma}^{(j)}_{ik}, w^{(j)}$
10: end function
```

Algorithm 3.5 Resample [33]

```
1: function resample($\langle w^{(j)}, \hat{\mathbf{x}}^{(j)}_{k}, \hat{\boldsymbol{\mu}}^{(j)}_{k}, \boldsymbol{\Sigma}^{(j)}_{k}\rangle_{j=1,\ldots,N}$)
2:     $N_{eff} = 1/\sum_{j}(w^{(j)})^{2}$
3:     if $N_{eff}$ < threshold then
4:         $\mathcal{X}$ = Resample $\mathbf{x}^{(j)}_{k}$ from $\mathcal{X}$ with probability $p = w^{(j)}_{k}$
5:     end if
6:     $w^{(j)}_{new} = 1/N, j = 1, \ldots, N$
7:     return $\mathcal{X}_{new}$
8: end function
```

3.4. Simulation FastSLAM

The models described before (the robot model and SLAM algorithm) will be validated using simulations. To research the performance of a new algorithm, the behavior of the robot is simulated. The robot model is now projected on a 2D map, so the state vector is simplified to only the x- and y-position and orientation θ of the robot. The input vector is then the linear velocity v and angular velocity ω of the robot. In Figure 3.4 the simulation is visualized for a particle filter with 10 particles, where both the true trajectory (red robot) and predicted trajectory (blue robot) are visualized. The environment contains five landmarks that need to be mapped. To map the landmarks, the robot navigates to an endpoint via three different waypoints (red crosses), while it localizes itself in the map. For this simulation the number of dimensions, described in Section 3.2, is reduced from 3 dimensions to 2 dimensions. So the state space and input space are reduced as follows:

$$\mathbf{x}_k = \begin{pmatrix} x_k & y_k & \theta_k \end{pmatrix}^{\top} \tag{3.17}$$

$$\mathbf{u}_k = \begin{pmatrix} v & \omega \end{pmatrix}^{\top} \tag{3.18}$$

where θ_k is the orientation at time k and ω is the angular velocity. This also results in a reduction of $\mathbf{Q}$, which is now written as:

$$\mathbf{Q} = \begin{pmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{pmatrix} \tag{3.19}$$

The simulation parameters are listed in Tab. 3.1, where v is the linear velocity, dt is the sampling time and N_{eff} is the number of effective particles, needed for the resampling criteria.

3.5. Performance measures

To measure the performance of the algorithm, different metrics of the algorithm will be measured. The types of errors that are induced for FastSLAM can be divided into two classes: the estimation error for both the trajectory and the landmark positions, and the error related to the variation inherent in random sampling. The position estimation error of the trajectory estimation will be measured using the time-averaged root mean squared (ARMS) error, discussed in Section 3.5.1. The position estimation error

Table 3.1: Parameters used for the simulation

Parameter	Description	Value
v	Velocity robot	$30m/s$
obs_range	Observation range of the sensors of the robot	$30m$
dt	Sampling time	$0.25s$
N_{eff}	Effective number of particles	7.5
σ_v	Standard deviation velocity robot	$1.2m/s$
σ_θ	Standard deviation angle robot	$12\pi/180^\circ$
σ_r	Standard deviation distance measurements	$0.5m$
σ_ϕ	Standard deviation bearing measurements	$20\pi/180^\circ$

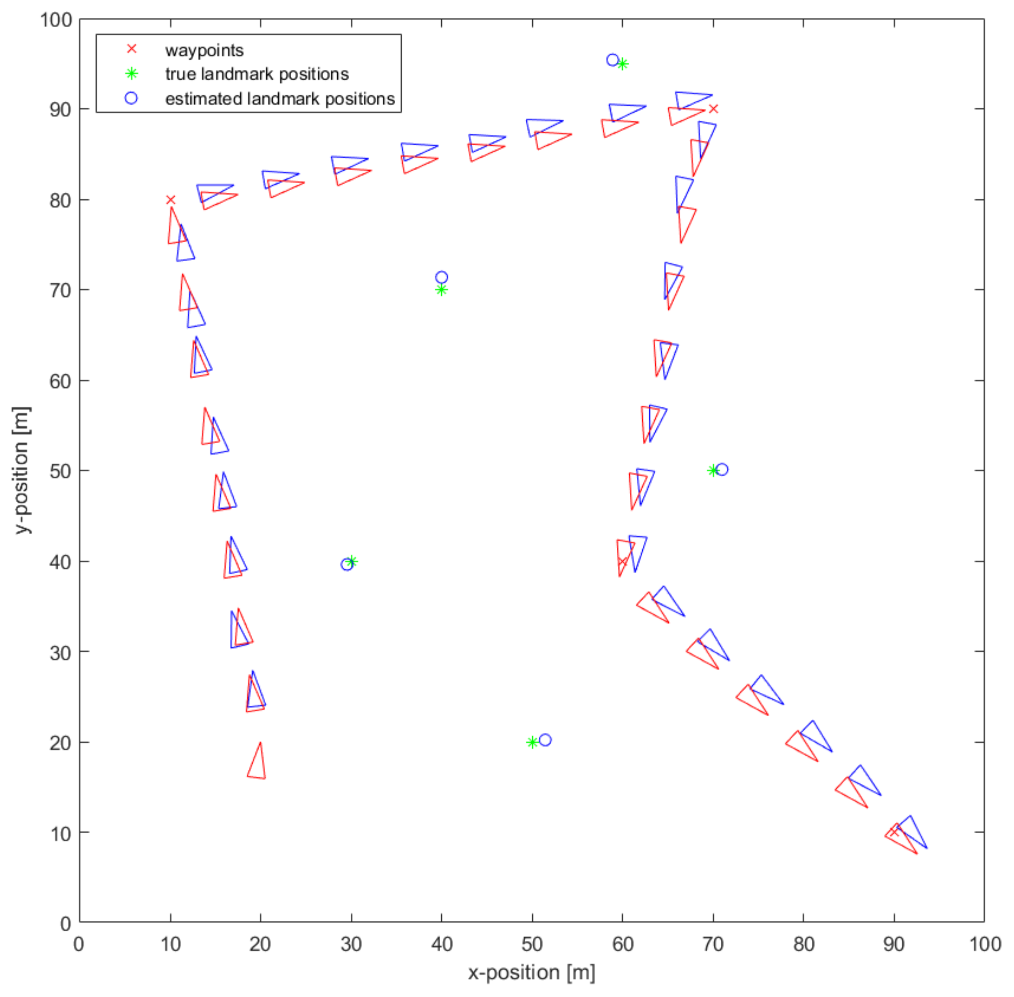


Figure 3.4: Estimated trajectory and landmark positions using FastSLAM with 10 particles. The red trajectory is the groundtruth and the blue trajectory is the estimated trajectory.

of the landmark estimation will be measured using the root mean squared (RMS) error, discussed in Section 3.5.2. The variation error will be expressed using the Kullback-Leibler Divergence, discussed in Section 3.5.3.

However, these metrics are not the only measures that expresses the efficiency of a SLAM algorithm. Other indices that should be taken into account while verifying the performance of the algorithm are:

- Processing time, discussed in Section 3.5.4
- Memory, discussed in Section 3.5.5

The metrics used to verify the simulations in the next chapters are explained in the following sections.

3.5.1. Error of the trajectory estimation

The position error is inherent to the SLAM problem. This is induced by the noise in the measurements. A useful measure to express the performance of the trajectory estimation in the SLAM algorithm is the time-averaged root mean squared error given as

$$\text{ARMS} = \sqrt{\frac{1}{T}\sum_{k=1}^{T} ||\mathbf{x}_k - \hat{\mathbf{x}}_k||^2}, \tag{3.20}$$

where T is the total number of time steps needed to complete the trajectory, $\mathbf{x}_k$ is the true robot state at time step k and $\hat{\mathbf{x}}_k$ is the estimated robot state at time step k [29]. In this metric the difference in length of robot trajectories is compensated.

3.5.2. Error of the map estimation

The position error of the map estimation is expressed using the root mean squared error of the positions of the landmarks

$$\text{RMS} = \sqrt{\sum_{m=1}^{M} ||\boldsymbol{\mu}_m - \hat{\boldsymbol{\mu}}_m||^2}, \tag{3.21}$$

where M is the total number of observed landmarks, $\boldsymbol{\mu}_m$ is the true location of landmark m and $\hat{\boldsymbol{\mu}}_m$ is the estimated location of landmark m.

3.5.3. Kullback-Leibler Divergence

While using a particle filter for SLAM, this induces a new type of error, specifically for this filter design. A finite number of samples is drawn, so the statistics of these samples differ from the statistics of the original density. For example the mean and variance of the drawn samples are slightly different than the mean and variance of the original density. The more samples are drawn, the smaller the error is. This variability due to random sampling is called the variance [34, Ch. 4.3.4].

A method to analyze the performance of a particle filter is in general done using the mean squared error between point estimators. In the case of a non-Gaussian distribution, this metric can be a meaningless number. Another metric to compare particle filters with each other is the Kullback-Leibler Divergence (KLD) [4].

Let p and q be two densities on $\mathbb{R}^d$, the KLD $D_{KL}(p, q)$ is then defined as the expected value of the log likelihood ratio between p and q:

$$D_{KL}(p, q) := \mathbb{E}_p\left[\log\frac{p(\mathbf{x})}{q(\mathbf{x})}\right] \tag{3.22}$$

If D_{KL} has a small value, the densities are close together and in case it is zero the densities are equal. The number on its own does not have a significant meaning, it should be evaluated to a benchmark.

For comparing two clouds this metric is combined with the κ-Nearest Neighbor (κNN) [37]. Using this metric, two clouds with different numbers of samples can be compared and thus an optimal number of particles can be found using this method. Assume that the simulations will be run twice with a different number of particles, the first time with the set of samples $\{X_1, \ldots, X_{N_1}\}$ are drawn and the second simulation every time step the samples $\{Y_1, \ldots, Y_{N_2}\}$, where N_1 and N_2 are the number of particles in the first and second simulations respectively. Let then $\nu_{k_i}(i)$ be the Euclidean distance from X_i to its k_i-NN in $\{Y_j\}$ and $\rho_{l_i}(i)$ the Euclidean distance between X_i to its l_i-NN in $\{X_j\}_{j \neq i}$. In [37] an asymptotically unbiased and mean-square consistent estimator $\hat{D}_{KL}(p, q)$ is presented as

$$\hat{D}_{KL}(p, q) = \frac{d}{N_1} \sum_{i=1}^{N_1} \log \frac{\nu_{k_i}(i)}{\rho_{l_i}(i)} + \frac{1}{N_1} \sum_{i=1}^{N_1} [\psi(l_i) - \psi(k_i)] + \log \frac{N_2}{N_1 - 1}, \tag{3.23}$$

where ψ is the Digamma function, the logarithmic derivative of the Gamma function.

3.5.4. Time complexity

The time complexity of an algorithm can be expressed in the big O notation ($\mathcal{O}$). This notation expression is normally a function of the input parameters and denotes the time to run an algorithm up to a constant factor.

Representing the time complexity of an algorithm in the big O notation can be difficult, in that case, the run time of an algorithm can also be taken as a measure for the time complexity. However, the run time of an algorithm is highly language and platform dependent, but it provides a measure to compare two algorithms in the same language and platform.

This metric will play the biggest role to determine whether the proposed algorithms use less computation and therefore less energy.

3.5.5. Memory

A disadvantage of using a particle filter for the SLAM problem is the multiple storage for the same data: every particle stores the whole map. Using EKF-SLAM might sound more beneficial in terms of memory load, however, in EKF-SLAM covariance matrices grow quadratically with the number landmarks. So for a large number of landmarks, the memory needed and computational load grows much faster than for FastSLAM. Every particle only stores the last known pose and all landmarks and is therefore in both memory and computation more efficient. In addition, the use of memory can be tweaked, depending on the application, by choosing a different number of particles. This method is not possible for EKF-SLAM. This metric will not be measured for the proposed algorithms, but there will be reflected on what effect the new algorithms have on the memory.

4 Constrained FastSLAM

In this chapter the first improved algorithm will be derived, based on FastSLAM. In this proposed algorithm, constrained FastSLAM, spatial constraints will be added. A derivation will be proposed, which will be validated by simulations of the constrained FastSLAM and unconstrained FastSLAM on four synthetic datasets, to compare the results in the last section.

4.1. Introduction

The idea behind this proposed algorithm is that the observed landmarks will be incorporated into the trajectory estimation as spatial constraints. In the unconstrained FastSLAM algorithm, the landmark information is only used to estimate the pose of the robot relative to the landmarks. In this proposed algorithm, the information about the landmarks will also be used to improve the sampling of unconstrained FastSLAM. The hypothesis is that the proposed algorithm requires more computations than unconstrained FastSLAM, because it processes the observed landmark information as spatial constraints. However, using the proposed algorithm might result in a more accurate estimation. If this is the case, the number of particles can be reduced to reduce the computation load without loss of accuracy compared to unconstrained FastSLAM. To verify whether this is the case, the proposed algorithm is simulated for a few different numbers of particles.

4.2. Mathematical derivation

To improve FastSLAM for autonomous navigation, the idea is to add spatial constraints to the unconstrained FastSLAM algorithm. The robot will navigate itself through a new environment and thereby it will need to avoid obstacles. The downside of using the unconstrained FastSLAM algorithm is that it might still sample particles at a point where an obstacle is already observed. So the idea of the proposed algorithm is to not only use the observation measurements for map estimation, but also for a more efficient method to sample during the unconstrained FastSLAM algorithm.

The first step is to implement constraints in the unconstrained FastSLAM algorithm. These constraints are on a subset of the states of the robot, namely the position of the robot. The constraints obeyed by the robot are represented as an area where the robot may sample its particles from, so the states need to be within an admissible set $\mathbf{C}_k$:

$$\mathbf{x}_k \in \mathbf{C}_k \tag{4.1}$$

This set $\mathbf{C}_k$ is a region where no obstacles are observed, this set changes every time step.

Implementing spatial constraints in a particle filter can be done in multiple stages of the algorithm. A general solution, proposed in Challa et al. [2], is rejection sampling. This method draws samples from an unconstrained proposal distribution as described in (3.9). If it does not belong to the constrained set, the sample will be redrawn, otherwise the sample will be accepted and continue. The advantage of this method is that it will always continue with a feasible sample. The downside, however, is that it can be computational expensive if it needs to redraw a lot of samples. Due to this possible high computational load, this method will not be used.

A second solution is given in Pirard et al. [28], where the constraints are introduced in the update step. A generalized likelihood is given as:

$$p(\mathbf{C}_k|\mathbf{x}_k) = \begin{cases} 1 & \mathbf{x}_k \in \mathbf{C}_k \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Adding these constraints in (2.6) gives the following measurement update:

$$p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k}, \mathbf{U}_{1:k}, \mathbf{x}_0, \mathbf{C}_k) = \frac{p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m})p(\mathbf{C}_k|\mathbf{x}_k)p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k-1}, \mathbf{U}_{1:k}, \mathbf{x}_0, \mathbf{C}_{k-1})}{p(\mathbf{z}_k|\mathbf{Z}_{1:k-1}, \mathbf{U}_{1:k}, \mathbf{C}_k)p(\mathbf{C}_k|\mathbf{C}_{k-1})} \tag{4.3}$$

This is the same measurement update, however, the constraints are incorporated. This posterior distribution is either zero if the states are outside the feasible set, or $p(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{1:k}, \mathbf{U}_{1:k}, \mathbf{x}_0)$ if the states are admissible. This method uses an unconstrained proposal distribution, which makes it easy to sample from. The disadvantage of this technique is that there is no guarantee that a sample is drawn which lays within the constrained area. Due to this uncertainty about a feasible solution, this method will not be used to build on.

A third proposed technique, focused on the Rao-Blackwellized particle filter, is proposed in Pirard et al. [28]. The principles of the Rao-Blackwellized particle filter, as discussed in Section 3.3 is combined with the spatial constraints. These constraints are implemented in the proposal distribution, where the state is drawn from. This method is explored in the remainder of this chapter, since it is closest to already existing unconstrained FastSLAM algorithm [28].

4.2.1. Proposal distribution

The constraints are incorporated in the proposal distribution. To make sure that the samples are drawn from an admissible set, the proposal distribution is altered. First an approximate support S_k for the robot positions need to be found. In this case the support is a function of the previous state and the odometry measurements: $\mathbf{S}_k(\mathbf{x}_k) = p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k)$. The support is equal to the confidence region, corresponding to an one-sigma ellipse of the robot position. This support is then intersected with the feasible set. This feasible set is the area in which the robot has not detected any landmarks. Since the observation of the landmarks is noisy as well, there is no guarantee where a landmark is located. To tackle this problem, the robot cannot be in the area associated with confidence region of the covariance matrix of a certain landmark. This admissible set is the set $\mathbf{C}_k$ as described in (4.1). After every observation, the confidence region of an observed landmark gets smaller, which makes the admissible set more accurate. The proposed distribution is then the intersection between the support and the admissible set $\mathbf{D}_k = \mathbf{S}_k \cap \mathbf{C}_k$. The weight of the sample is then calculated as a function of the volume of set $\mathbf{D}_k$. The proposal distribution π can be computed as Algorithm 4.1.

Algorithm 4.1 Proposal distribution π for constrained FastSLAM [28]

```
1: for $j := 1$ to $N$ do
2:     Find appropriate support $\mathbf{S}_k$
3:     Calculate $\mathbf{D}_k = \mathbf{S}_k \cap \mathbf{C}_k$
4:     Calculate the volume $V(\mathbf{D}_k)$
5:     Draw a uniform sample from $\mathbf{D}_k$
6:     Calculate importance weight $p(\mathbf{x}_k^{(j)}|\mathbf{x}_{k-1}^{(j)}, \mathbf{z}_k, \mathbf{C}_k) = 1/V(\mathbf{D}_k)$
7:     return feasible particle $j$
8: end for
```

4.3. Simulation setup

For this simulation a ground robot (in 2D) is used, so the state space, input space and covariance matrix is the same as (3.17), (3.18) and (3.19) respectively. The simulations are done both for unconstrained

Table 4.1: Parameters used for the simulation

Parameter	Description	Value
v	Velocity robot	$3m/s$
obs_range	Observation range of the sensors of the robot	$30m$
dt	Sampling time	$0.25s$
N_{eff}	Effective number of particles	$0.75N$
σ_v	Standard deviation velocity robot	$1m/s$
σ_θ	Standard deviation angle robot	$12\pi/180^\circ$
σ_r	Standard deviation distance measurements	$0.8m$
σ_ϕ	Standard deviation bearing measurements	$12\pi/180^\circ$

FastSLAM as discussed in Section 3.4 and constrained FastSLAM to be able to compare the performance. In this thesis, the focus of the results is mostly in finding opportunities to make the algorithms more energy-efficient. Therefore the simulations will be run with a varying limited number of particles $N = \{10, 100, 1000\}$ to keep the run time as low as possible while still being able to see the trend in the outcomes. This way, a lot of simulations can be run to find out the trends of the algorithm in the results. The setup will be run for 100 Monte Carlo runs, after which the results have converged to a level where the outcomes of the different algorithms and parameters are distinguishable. The parameters used for the simulations are listed in Table 4.1.

To determine the performance of the two methods, the following error metrics will be used, as described in Section 3.5:

- RMS error of landmark estimation compared to the groundtruth.
- ARMS error of the poses of the robot compared to the groundtruth.
- Mean and variance of the performance of the particle filters using KLD.

Also the following performance indicators are investigated to be able to compare the algorithms with respect to their energy efficiency:

- Computational complexity
- Memory

4.3.1. Datasets

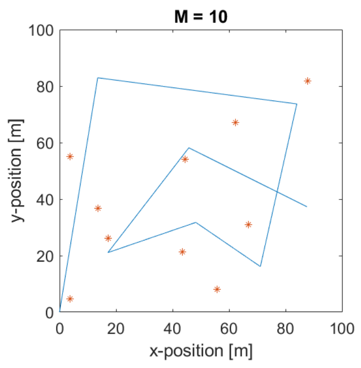
To verify in what kind of environment the algorithm performs best, the simulations are run on different datasets, visualized in Figure 4.1. The field covers in all cases an area of $100m \times 100m$. The trajectory (blue line) is the same, all starting from point $(0, 0)$, but the number of landmarks (red stars) and their positions is different. The number of landmarks and the placement of the landmarks effect the performance of unconstrained FastSLAM, because a lot of landmarks in the observation range of the robot improves the localization performance. Due to this dependence the number and configuration of the landmarks is varied.

4.4. Results

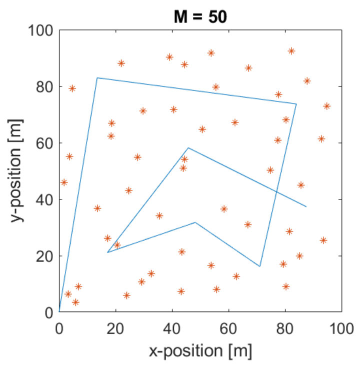
In this section the simulation results will be presented using the simulation setup described in the previous section. The accuracy and performance of the unconstrained and constrained FastSLAM are compared. Each simulation contains 100 Monte Carlo runs. The simulations are done using MATLAB R2019b.

4.4.1. Dataset 1

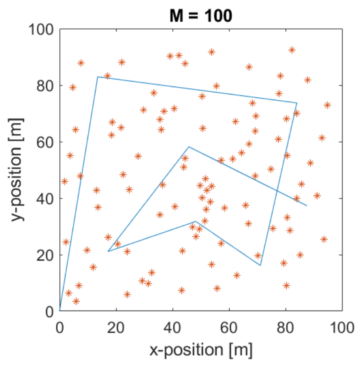
Dataset 1 contains 10 landmarks, this dataset can be found in Figure 4.1a. The hypothesis is that the algorithms will give similar results, since the effect of the constraints will be minimal. The ARMS error for the trajectory of the robot is shown in Figure 4.2 as well as the RMS error for the landmark estimation. These results show that the performance of the unconstrained FastSLAM algorithm increases



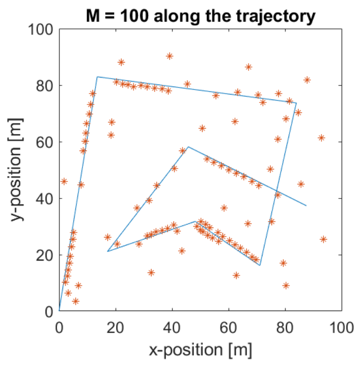
(a) Dataset with 10 landmarks



(b) Dataset with 50 landmarks



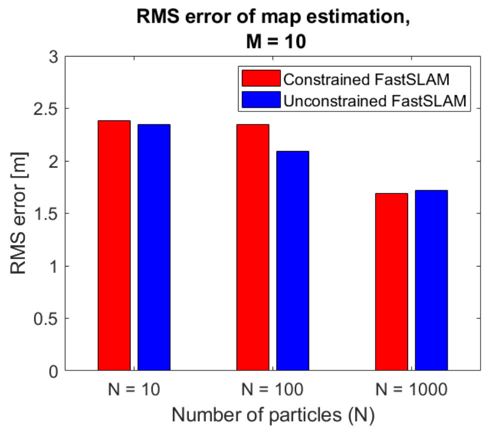
(c) Dataset with 100 landmarks



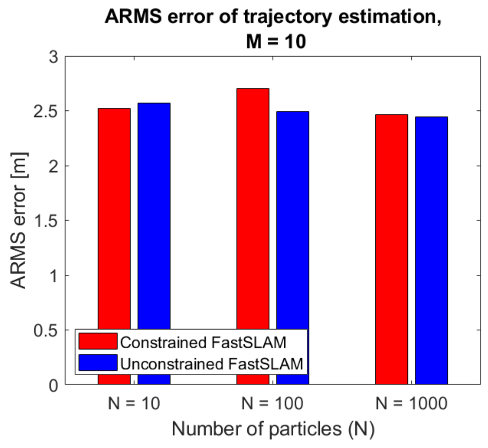
(d) Dataset with 100 landmarks in an organized way

Figure 4.1: Datasets used for the simulations. The trajectory (blue line) is the same, starting from point (0,0). The number of landmarks (red stars) differ.

monotonically with an increasing number of particles. However, for 1000 particles the performance of constrained FastSLAM is even better than for unconstrained FastSLAM looking at the map estimation. Overall these datasets give similar results for both algorithms.



(a) RMS error of the map estimation containing 10 landmarks averaged over 100 Monte Carlo runs.



(b) ARMS error of the trajectory estimation in a map containing 10 landmarks averaged over 100 Monte Carlo runs.

Figure 4.2: Comparison of the performance of unconstrained and constrained FastSLAM on the first dataset containing 10 landmarks averaged over 100 Monte Carlo runs.

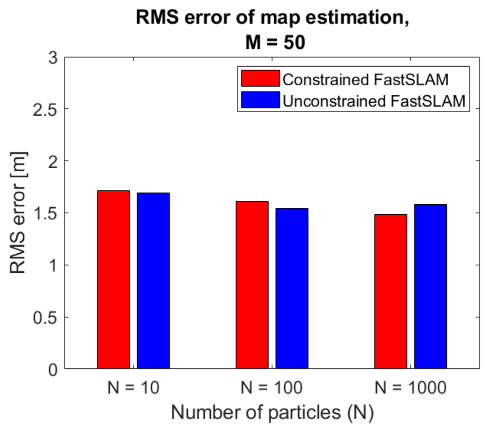
4.4.2. Dataset 2

The second dataset contains 50 landmarks, as visualized in Figure 4.1b. The hypothesis is that the simulations on this dataset will provide better results than the first dataset, because the environment

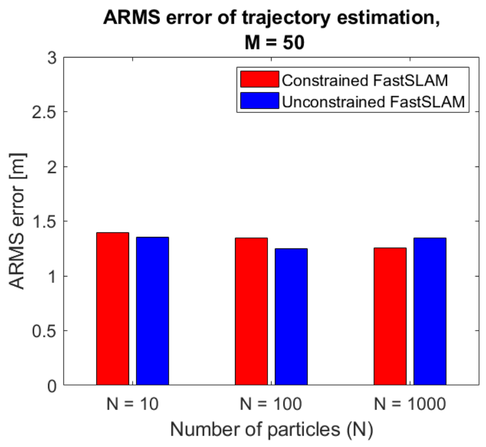
Table 4.2: Comparison performance between unconstrained FastSLAM and constrained FastSLAM for 10 landmarks

		N = 10	N = 100	N = 1000
ARMS error trajectory [m]	Unconstrained	2.57	2.50	2.45
	Constrained	2.52	2.71	2.47
RMS error map [m]	Unconstrained	2.35	2.09	1.72
	Constrained	2.38	2.35	1.69

is more dense. The errors for a simulation averaged over 100 Monte Carlo runs is shown in Figure 4.3 and Table 4.3. These results show that unconstrained FastSLAM is better for a lower number of particles, however, for 1000 particles the performance of constrained FastSLAM is clearly better both for the trajectory estimation as the map estimation. This is in line with the results of the first dataset. The results also show that overall the error is smaller than the simulations with 10 landmarks as expected.



(a) RMS error of the map estimation containing 50 landmarks averaged over 100 Monte Carlo runs.



(b) ARMS error of the trajectory estimation in a map containing 50 landmarks averaged over 100 Monte Carlo runs.

Figure 4.3: Comparison of the performance of unconstrained and constrained FastSLAM on the second dataset containing 50 landmarks averaged over 100 Monte Carlo runs.

Table 4.3: Comparison performance between unconstrained FastSLAM and constrained FastSLAM for 50 landmarks

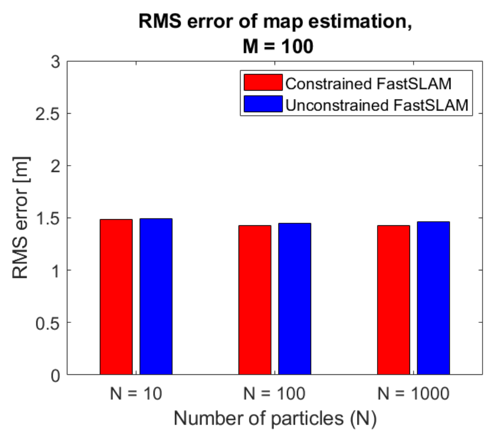
		N = 10	N = 100	N = 1000
ARMS error trajectory [m]	Unconstrained	1.35	1.25	1.34
	Constrained	1.40	1.35	1.26
RMS error map [m]	Unconstrained	1.69	1.55	1.58
	Constrained	1.71	1.61	1.48

4.4.3. Dataset 3

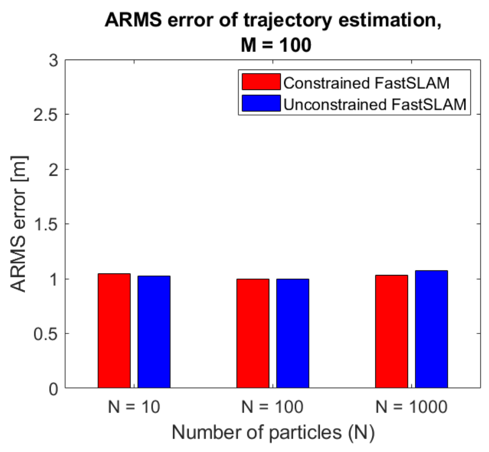
The third dataset contains 100 landmarks in an arbitrary position, as shown in Figure 4.1c. The results of this dataset can be found in Figure 4.4 and Table 4.4. The overall error is slightly smaller than the errors of the second dataset as expected. For this dataset, the constrained FastSLAM is slightly better than unconstrained FastSLAM for most cases, but the differences between the errors are closer together than in the previous two cases. The trend that constrained FastSLAM is better for 1000 particles also continues in this case.

4.4.4. Dataset 4

The last dataset also contains 100 landmarks, however, in a structured order along the trajectory, visualized in Figure 4.1d. Comparing the results in Table 4.5 with the previous dataset show that the performance of constrained FastSLAM is slightly worse for the structured dataset, especially for 1000 particles. The decrease in performance might be caused by the fact that the algorithm is sometimes



(a) RMS error of the map estimation containing 100 landmarks averaged over 100 Monte Carlo runs.



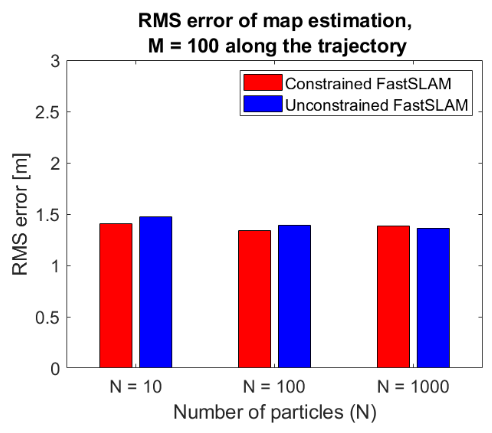
(b) ARMS error of the trajectory estimation in a map containing 100 landmarks averaged over 100 Monte Carlo runs.

Figure 4.4: Comparison of the performance of unconstrained and constrained FastSLAM on the third dataset containing 100 landmarks averaged over 100 Monte Carlo runs.

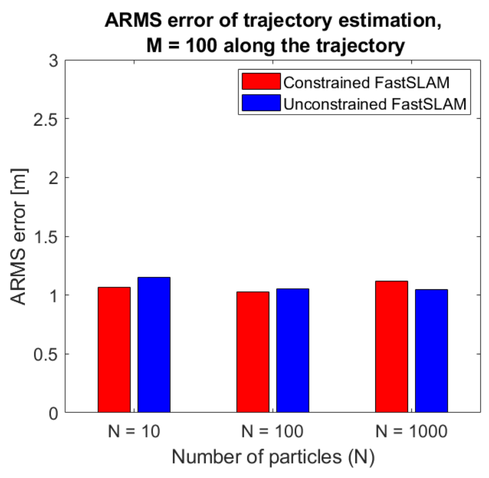
Table 4.4: Comparison performance between unconstrained FastSLAM and constrained FastSLAM for 100 landmarks

		N = 10	N = 100	N = 1000
ARMS error trajectory [m]	Unconstrained	1.02	0.99	1.08
	Constrained	1.05	1.00	1.03
RMS error map [m]	Unconstrained	1.49	1.45	1.47
	Constrained	1.49	1.42	1.43

too constrained and is therefore giving worse results. This problem of being too constraint is the result of a lot of overlapping confidence regions per time step, which makes the admissible set too small and thus unreliable. The performance of this dataset is also visualized in Figure 4.5.



(a) RMS error of the map estimation containing 100 landmarks along the trajectory averaged over 100 Monte Carlo runs.



(b) ARMS error of the trajectory estimation in a map containing 100 landmarks along the trajectory averaged over 100 Monte Carlo runs.

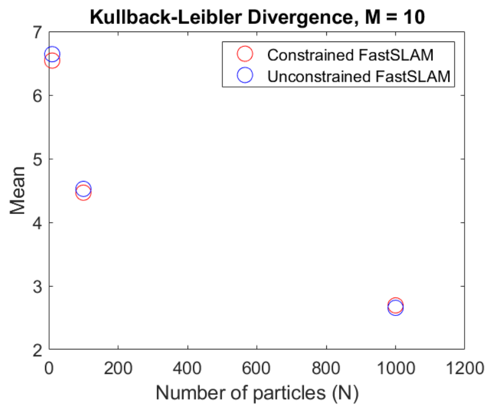
Figure 4.5: Comparison of the performance of unconstrained and constrained FastSLAM on the fourth dataset containing 100 landmarks along the trajectory averaged over 100 Monte Carlo runs.

4.4.5. Kullback-Leibler Divergence

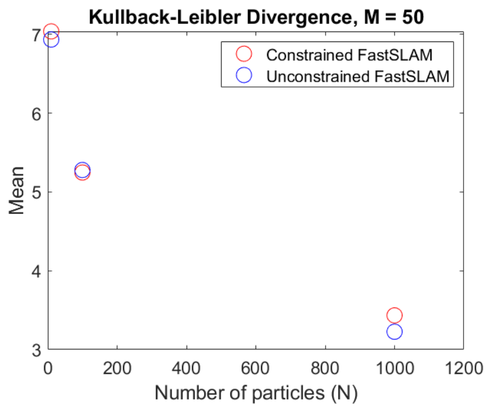
Since the results are not conclusive, another metric is used to see whether the number of particles to represent the robot is of influence on the performance. To do so, the Kullback-Leibler Divergence as

Table 4.5: Comparison performance between unconstrained FastSLAM and constrained FastSLAM for 100 landmarks in a structured way

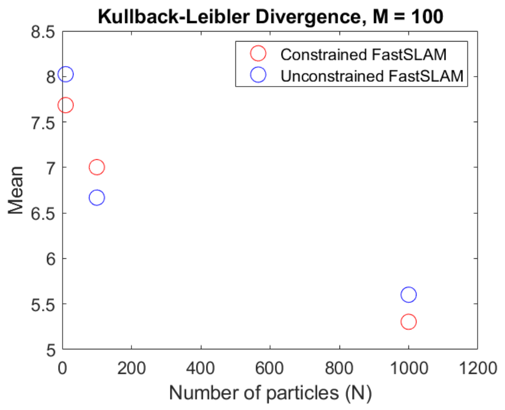
		N = 10	N = 100	N = 1000
ARMS error trajectory [m]	Unconstrained	1.15	1.05	1.05
	Constrained	1.07	1.03	1.12
RMS error map [m]	Unconstrained	1.48	1.39	1.36
	Constrained	1.41	1.34	1.38



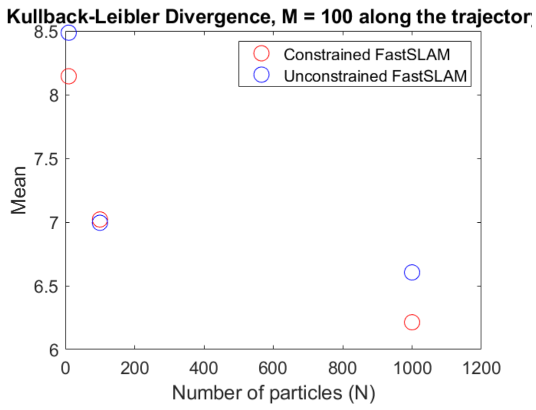
(a) Dataset with 10 landmarks



(b) Dataset with 50 landmarks



(c) Dataset with 100 landmarks



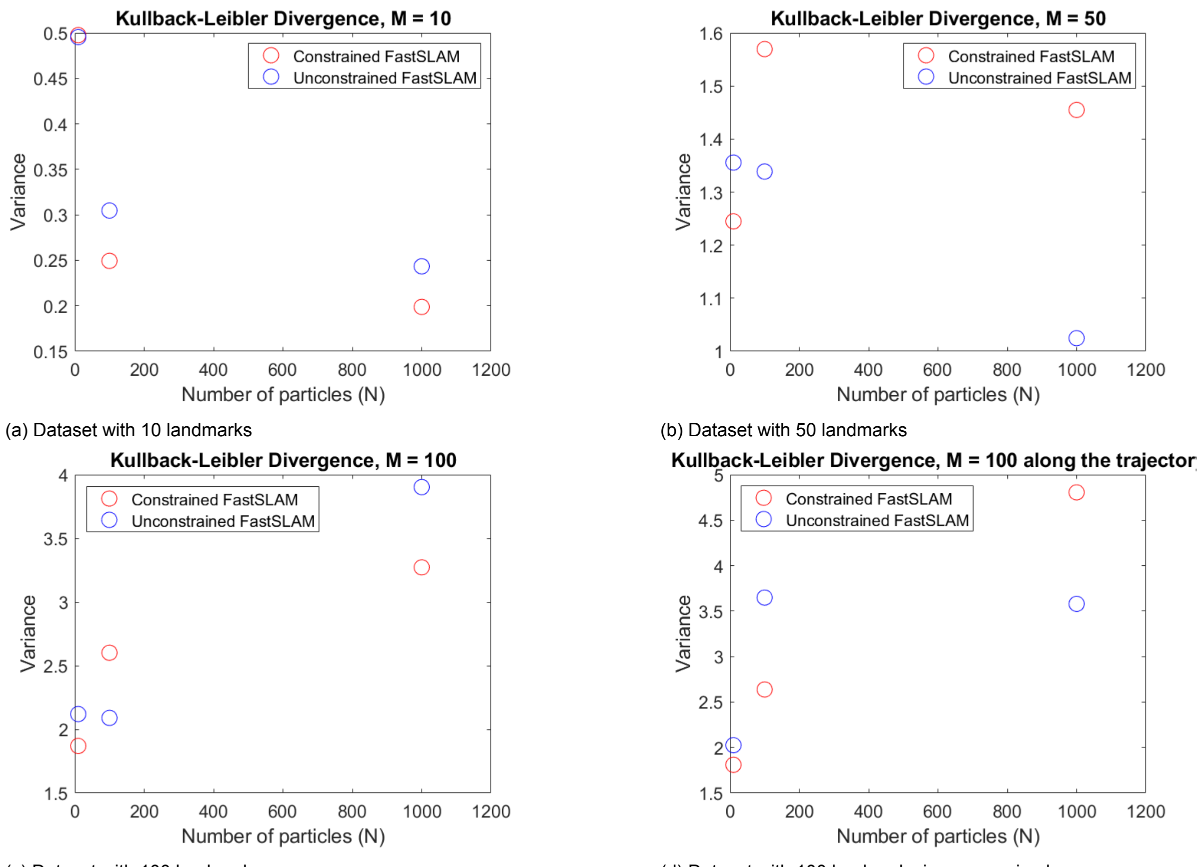
(d) Dataset with 100 landmarks in an organized way

Figure 4.6: Estimated mean of the Kullback-Leibler Divergence estimator. The KLD is an average calculated over 100 Monte Carlo runs.

discussed in Section 3.5.3 is used. A particle swarm containing 5000 particles is used as a reference swarm, which is assumed to represent the true distribution. Only the first nearest neighbour is considered, so both l_i and k_i are 1. The results of this KLD estimator are shown as mean in Figure 4.6 and as variance in Figure 4.7.

The mean value of the KLD estimator show similar results with the results of the trajectory and map estimations. In the case of 10 landmarks, the difference between the means of the KLD estimator of the two algorithms is negligible. This result follows from the fact that the influence of the spatial constraints in constrained FastSLAM is small due to little landmarks in the dataset. For 50 landmarks the gap between the two algorithms is already visible, but still small. For 100 landmarks the difference in mean becomes clearly visible. Especially for 1000 particles, the difference between the constrained and unconstrained algorithm is significant. This follows from the fact that the spatial constraints have a bigger effect on constrained FastSLAM than for a lower number of landmarks, which is clearly a significant improvement.

The variance of the KLD estimator shows different results than the mean of the KLD. With an increasing



(a) Dataset with 10 landmarks

(b) Dataset with 50 landmarks

(c) Dataset with 100 landmarks

(d) Dataset with 100 landmarks in an organized way

Figure 4.7: Estimated variance of the Kullback-Leibler Divergence estimator. The KLD is calculated over 100 Monte Carlo runs.

number of particles, the variance is expected to reduce, this would mirror the reduction of error of the random sampling. However, for the constrained FastSLAM algorithm this is not the case. This means that the variance of the calculated distribution of the constrained FastSLAM is significantly higher than the calculated distribution for unconstrained FastSLAM compared to the 'true' distribution. This would mean that the constrained FastSLAM with a high number of particles is less 'certain' about the estimated trajectory. This means that a higher number of particles is not particularly more certain than a lower number of particles.

4.4.6. Computational complexity

The complexity of the algorithm increases with the addition of constraints for unconstrained FastSLAM. The complexity of the unconstrained FastSLAM algorithm is $\mathcal{O}(NlogM)$, whereas the complexity of the constrained FastSLAM increases to $\mathcal{O}(M_o NlogM)$, where M_o is the average number of observed landmarks per time step. This average number of observed landmarks depends on the observation range of the robot, which is $30m$ for this robot. In a area that covers $100m \times 100m$, this means that the robot can approximately sense a quarter of the field. That would mean that the computational load of the robot is 2.5 upto 25 times higher depending on the dataset used. So the addition of spatial constraints to unconstrained FastSLAM results in a higher computational load than for unconstrained FastSLAM in the case of the same number of particles and landmarks.

4.4.7. Memory

This algorithm reuses information about the estimated landmark position and certainty of this estimation. This does not require a bigger memory, however, building the proposed distribution every time step requires a little more memory per particle, so that will have an influence on the memory resources needed.

4.5. Conclusion

The constrained FastSLAM algorithm has shown potential to make autonomous robots more energy-efficient. The improvement is in particular visible for dense environments, in terms of landmarks. The performance of the unconstrained and constrained FastSLAM is around the same level in terms of accuracy for a lower level of particles. For 1000 particles, the trend is clearly visible that constrained FastSLAM gives a better accuracy than unconstrained FastSLAM. The mean of the KLD estimator shows that the potential of this algorithm. The trend shows that for a larger number of landmarks and particles constrained FastSLAM comes closer to true distribution, which shows that less particles are needed to realize the same accuracy as unconstrained FastSLAM.

However, the complexity of the proposed algorithm is higher and dependent on the number of observed landmarks. The contradiction of this algorithm in terms of computations, and therefore energy efficiency, is that the new algorithm performs better with more landmarks, but also demands a higher computational load with more landmarks. Even though the performance in terms of accuracy shows a great potential in this algorithm, the increasing computational load prevents constrained FastSLAM to be more energy-efficient than unconstrained FastSLAM.

4.6. Future work

The simulations that validated this algorithm have shown that there is potential in constrained FastSLAM. This potential could be sorted out by running more simulations with a higher number of landmarks and a higher number of particles. However, the significant increase in computational complexity is a problem that still needs to be tackled before constrained FastSLAM will become more energy-efficient than unconstrained FastSLAM.

If this problem has been solved, a more accurate version of constrained FastSLAM could be developed. In this thesis, the assumption has been used that the confidence region of the landmarks is a non-admissible region for the sampling of the particles. However, this is overconstrained and might make the algorithm less accurate in case of a lot of landmarks. This challenge can be tackled by adding chance-constrained optimization. This will add more complexity to the algorithm, but will probably also result in a more accurate estimation.

5 Parallelized particle filter

Another method of reducing the processing power is to reduce the computations. To do so, the computation of the measurements will be processed by a parallelized particle filter (PPF-SLAM). The idea is that after the observation a particle will only update one landmark in its map at a time and not all the observed landmarks at once. Every particle will pick a random landmark to update. After this update the information is communicated with the other particles to pass through the processed information. The particles are still maintaining its own map and trajectory. By implementing this method, the communication complexity between the particles needs to be taken into account, because there will always be a trade-off between the communication overhead, computational power and accuracy.

This type of algorithm has been used in literature before, so those methods will be listed. Afterwards the robot model will be revisited. Then the mathematical derivation will be shown in Section 3. This algorithm will be validated by simulations, using the same dataset as in Chapter 4 and the results of those simulations will be discussed in this chapter as well. After which a conclusion will be drawn and some ideas for future work will be given.

5.1. State of the art

Several papers have already implemented a form of distributed/decentralized/parallelized FastSLAM; most of those methods are actually a parallelized particle filter, since it is all for a single agent system, where the update step to spread out the computational load is parallelized. One method of splitting this computation is execute the localization task and mapping task concurrently [18, 36]. These methods show improvement in accuracy, but with a higher computational load as result.

Other proposed methods split the observation measurements into feature measurements, so the measurement vector is represented as $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_m)^\top$. Each of these observations can then be computed using a local filter [27, 40, 41]. This method shows major improvement, especially if done in a practical set-up. This will also be used in the new proposed algorithm.

In the mentioned papers, the computation time decreases at the expense of the accuracy and communication overhead. In this chapter a new method will be proposed, based on a graph Laplacian Distributed Particle Filter, as used in Rabbat et al. [29]. The goal is to design an algorithm that reduces the computational load, thus reducing the needed energy, while keeping the communication overhead between the particles as low as possible.

5.2. Robot model

The goal is to explore a feature based map $\mathbf{m}$ using a single robot. The robot motion model is the same as (3.1), so it will be denoted by:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{v}_k, \tag{5.1}$$

where $\mathbf{x}_k$ is the state of the robot at time k, $\mathbf{u}_k$ is the input vector and $\mathbf{v}_k$ is the process noise. The measurements follow the model:

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{e}_k, \tag{5.2}$$

where $\mathbf{z}_k$ is the observation of the landmarks, $h(\cdot)$ is the sensor model as defined in (3.7) and $\mathbf{e}_k$ is the measurement noise. The first step to reduce the computation load is to split the observation model into landmark observations. So for every time step the observation model can be written as:

$$\mathbf{z}_k = \begin{pmatrix} \mathbf{z}_{1,k} \\ \mathbf{z}_{2,k} \\ \vdots \\ \mathbf{z}_{m,k} \end{pmatrix} = \begin{pmatrix} h(\mathbf{x}_k, \mathbf{m}_1) + \mathbf{e}_k \\ h(\mathbf{x}_k, \mathbf{m}_2) + \mathbf{e}_k \\ \vdots \\ h(\mathbf{x}_k, \mathbf{m}_m) + \mathbf{e}_k \end{pmatrix} \tag{5.3}$$

Every particle only uses one randomly picked feature observation used to estimate the trajectory and the landmark position. This information is then shared over a graph with the other particles, which will be explained in the following section.

5.3. Mathematical derivation

Now that the robot model has been adjusted, the particle filter itself also needs to be adjusted to make sure that the information between the particles can be shared efficiently. A small communication overhead leads to a small extra computational load, which means that the filter will be more energy-efficient than the particle filter described in Section 3.3.

The following derivation of the parallelized particle filter will be used. Algorithm 3.1 functions as a starting point for this algorithm. The weight update as explained in Algorithm 3.1 is written as follows:

$$w_k^{(j)} \propto w_{k-1}^{(j)} \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(j)})p(\mathbf{x}_k^{(j)}|\mathbf{x}_{k-1}^{(j)}, \mathbf{u}_k)}{\pi(\mathbf{x}_k^{(j)}|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k)}, \tag{5.4}$$

where $w_k^{(j)}$ is the weight of particle j at time k, $p(\mathbf{z}_k|\mathbf{x}_k^{(j)})$ is the distribution of the measurements given the position of the robot. $p(\mathbf{x}_k^{(j)}|\mathbf{x}_{k-1}^{(j)}, \mathbf{u}_k)$ is the distribution of the estimated position of particle j at time k given the estimated position of the previous time step and the odometry measurement $\mathbf{u}_k$ [29]. $\pi(\mathbf{x}_k^{(j)}|\mathbf{X}_{0:k-1}^{(j)}, \mathbf{Z}_{1:k}, \mathbf{u}_k)$ is the proposal distribution of the estimated position of particle j, which can make use of the information of the trajectory $\mathbf{X}_{0:k-1}$, observations $\mathbf{Z}_{1:k}$ so far and the odometry measurement $\mathbf{u}_k$.

To be able to implement a parallelized particle filter, a bootstrap particle filter is used [9, Ch. 4]. In this filter the proposal distribution $\pi(\mathbf{x}_k^{(j)}|\mathbf{X}_{0:k-1}, \mathbf{Z}_{1:k}, \mathbf{u}_k)$ is taken to be $p(\mathbf{x}_k^{(j)}|\mathbf{x}_{k-1}^{(j)}, \mathbf{u}_k)$, that simplifies the un-normalized weight update to

$$w_k^{(j)} = w_{k-1}^{(j)} p(\mathbf{z}_k|\mathbf{x}_k^{(j)}) \tag{5.5}$$

Let $\gamma_k^{(j)}$ be the log-likelihood of the measurements, so $\gamma_k^{(j)} = \log p(\mathbf{z}_k|\mathbf{x}_k^{(j)})$. The normalized weight update is then expressed as

$$w_k^{(j)} = \frac{w_{k-1}^{(j)} \exp\{\gamma_k^{(j)}\}}{\sum_{i=1}^{N} w_{k-1}^{(i)} \exp\{\gamma_k^{(i)}\}} \tag{5.6}$$

where the denominator is the normalization factor. By stacking these log-likelihoods a vector γ_k is obtained.

5.3.1. Graph theory

Particles with similar state estimation have typically similar weights. This assumption is used to built an adjacency matrix for the particles. For N particles, the adjacency matrix $\mathbf{A}$ denotes the symmetrized κ-nearest neighbor graph. In this case κ is a fixed number, where $\kappa \ll N$, that denotes whether an edge is placed in the adjacency matrix between particles. If particle j has particle i as κ nearest neighbor an edge is placed, between the particles, which means that $A_{ij} = A_{ji} = 1$. Note that for some particles the number of edges is bigger than κ, because it will not necessarily mean if particle j is the κ nearest neighbor of particle i that automatically particle i is also the κ nearest neighbor of particle j. But it is assumed that if information can flow from particle i to particle j, the other way around is also possible.

The degree matrix $\mathbf{D}$ denotes the number of neighbors, on the diagonal it contains the number of neighbors of particle i, otherwise it is zero. So the diagonal entries can be calculated by $D_{ii} = \sum_{j=1}^{N} A_{ij}$. The Laplacian matrix $\mathbf{L}$ is then denoted as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. This matrix has an eigendecomposition, because it is symmetric and real, so it can be written in the form $\mathbf{L} = \mathbf{F}\boldsymbol{\Lambda}\mathbf{F}^\top$, where $\mathbf{F}$ is an $N \times N$ matrix containing the eigenvectors and $\boldsymbol{\Lambda}$ is an $N \times N$ diagonal matrix of eigenvalues. The eigenvalues are ordered in ascending order, so

$$\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_N$$

5.3.2. Laplacian decomposition

The columns of $\mathbf{F}$ have some interesting properties to understand the graph on a signal processing basis [29]. A column $\mathbf{f}_i$, denoting the ith column of $\mathbf{F}$, can be interpreted as a Fourier basis for signals supported on the graph. The information contained in vector γ is shared between neighboring nodes, so the scalars are so called signals on the graph $\mathbf{A}$. However, sending this information over the whole graph might nullify the benefits of a parallelized particle filter due to the high communication overhead. First, γ_k can be approximated by

$$\hat{\gamma}_k = \sum_{j=1}^{c} \mathbf{f}_j^\top \gamma_k \mathbf{f}_j, \tag{5.7}$$

where $c \leq N$ and the eigenvalues $\{\mathbf{f}_1, \dots, \mathbf{f}_c\}$ correspond to the c smallest eigenvalues. This approximation makes sense since most of the energy of the graph is associated with the eigenvectors corresponding with the smallest eigenvalues. Besides using an approximation, the dimension of the shared information is also reduced, by transforming γ using the Fourier basis matrix $\mathbf{F}$: $\boldsymbol{\alpha}_k = \mathbf{F}^\top \gamma_k$, where $\boldsymbol{\alpha}_k$ is referred to as the vector containing Laplacian transform coefficients [29]. In Algorithm 5.1 the SLAM algorithm is explained using a parallelized particle filter.

Algorithm 5.1 PPF-SLAM (based on [29])

```
1: $\mathcal{X}_0 = \{w_0, \mathbf{x}_0, \hat{\boldsymbol{\mu}}, \boldsymbol{\Sigma}\}$
2: while true do
3:     $\mathbf{z}_{ik} = h(\mathbf{x}_k, \mathbf{m}_i)$
4:     if $\mathbf{z}_{ik}^{(j)}$ is never observed before then
5:         add_new_landmark$\left(\mathbf{z}_{ik}^{(j)}, \mathbf{x}_k^{(j)}, \mathbf{R}\right)$
6:     else
7:         update_landmark$\left(\boldsymbol{\mu}_{i,k-1}^{(j)}, \boldsymbol{\Sigma}_{i,k-1}^{(j)}, \mathbf{x}_k^{(j)}, \mathbf{z}_{ik}^{(j)}, \mathbf{R}\right)$
8:     end if
9:     for $j = 1$ to $N$ do
10:        Sample $\hat{\mathbf{x}}_k^{(j)} \sim p\left(\mathbf{x}_{k-1}^{(j)}, \mathbf{u}_k\right)$
11:    end for
12:    Compute $\mathbf{A}$ as the $\kappa$ nearest neighbor graph of the particles
13:    Calculate the Laplacian matrix $\mathbf{L}$ and do the eigendecomposition $\mathbf{L} = \mathbf{F}\boldsymbol{\Lambda}\mathbf{F}^\top$
14:    Calculate Laplacian transform coefficients $\boldsymbol{\alpha}_k = \mathbf{F}_c^\top \gamma_k$
15:    $\hat{\boldsymbol{\alpha}}_k$ = mean($\boldsymbol{\alpha}_k$)
16:    Calculate $\hat{\gamma}_k = \mathbf{F}_c \hat{\boldsymbol{\alpha}}_k$
17:    for $j = 1$ to $N$ do
18:        $w_k^{(j)} = \frac{w_{k-1}^{(j)} \exp\{\gamma_k^{(j)}\}}{\sum_{i=1}^{N} w_{k-1}^{(i)} \exp\{\gamma_k^{(i)}\}}$
19:    end for
20:    $\mathcal{X}_k$ = Resample($\mathcal{X}_{k-1}$)
21:    $k = k + 1$
22: end while
```

In theory, this algorithm should reduce the computation time and thus the computational power, which makes it more energy-efficient. However, the number of particles does not have a linear effect on the

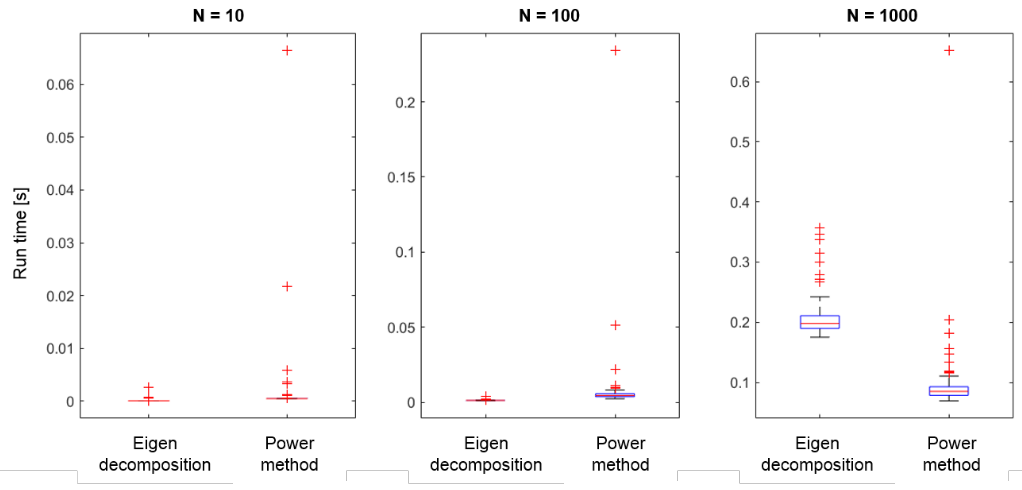


Figure 5.1: Comparison of the eigendecomposition and power method for 100 iterations of the PPF-SLAM algorithm for $c = 7$.

Table 5.1: Comparison between FastSLAM and PPF-SLAM

	FastSLAM	PPF-SLAM
Proposal distribution	$p(\mathbf{x}_k^{(j)} \vert \mathbf{x}_{k-1}^{(j)}, \mathbf{u}_k, \mathbf{z}_{k-1})$	$p(\mathbf{x}_k^{(j)} \vert \mathbf{x}_{k-1}^{(j)}, \mathbf{u}_k)$
Weight update	$w^{(j)} = \vert 2\pi\mathbf{R}\vert^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{z}_k - \hat{\mathbf{z}}^{(j)})^\top \mathbf{R}^{-1}(\mathbf{z}_k - \hat{\mathbf{z}}^{(j)})\right]$	$w_k^{(j)} = w_{k-1}^{(j)} p(\mathbf{z}_k \vert \mathbf{x}_k^{(j)})$
Complexity	Dominated by number of measurements	Dominated by number of particles

computation time of this algorithm. Computing the κ nearest neighbor graph and the eigendecomposition take up a significant part of the computation time as the number of particles increases.

The computational time of the eigendecomposition is up to $\mathcal{O}(N^3)$ depending on the form of the matrix and expected accuracy [6]. The sparsity of the matrix can be exploited to compute the first c eigenvectors of $\mathbf{F}$ using a power method [32]. Such a power method is for example the Lanczos algorithm. This algorithm finds the c most useful eigenvalues and eigenvectors of a Hermetian matrix. The most useful is for example the first c eigenvectors with the highest or lowest eigenvalues, or the first c eigenvalues with the smallest or biggest absolute value. By using this method, the computational time is drastically lowered, especially for a high number of particles. To show the difference between the eigendecomposition and the Lanczos method, the run time is plotted in Figure 5.1 for 100 iterations of the PPF-SLAM algorithm for $c = 7$. For a small number of particles, the difference is not significant, the power method is less consistent than the eigendecomposition, but especially for a higher number of particles, the difference in run time is clearly visible. For this reason the power method will be used for the simulations instead of the eigendecomposition.

5.3.3. Comparison

The differences between FastSLAM and PPF-SLAM are summarized in Table 5.1. As shown in the table, the main difference in the computation of the two algorithms lay in the proposal distribution and the weight update of the different particle filters. This then results in a different kind of computational complexity, one dominated by the number of measurements, the other dominated by the number of particles.

5.4. Simulations

The simulations are done on the same datasets (shown in Fig. 4.1) as explained in Chapter 4 and with the same set of particles $N = \{10, 100, 1000\}$. The performance of a parallelized particle filter is compared to FastSLAM. The parameters of these simulations are shown in Table 5.2.

Table 5.2: Parameters used for the simulation

Parameter	Description	Value
v	Velocity robot	$1m/s$
obs_range	Observation range of the sensors of the robot	$30m$
dt	Sampling time	$0.25s$
c		10
κ	Number of neighbors	7
N_{eff}	Effective number of particles	$0.75N$
σ_v	Standard deviation velocity robot	$0.1m/s$
σ_θ	Standard deviation angle robot	$12\pi/180^\circ$
σ_r	Standard deviation distance measurements	$0.8m$
σ_ϕ	Standard deviation bearing measurements	$12\pi/180^\circ$

To determine the performance of the two methods, the following error metrics will be used, as described in Section 3.5:

- RMS error of landmark estimation compared to the groundtruth.
- ARMS error of the poses of the robot compared to the groundtruth.

Also the following performance indicators are investigated to be able to compare the algorithms with respect to their energy efficiency:

- Run time of the algorithms
- Memory

5.5. Results

In this section the simulation results will be presented using the simulation setup described in the previous section. Besides the estimation accuracy of the algorithms, also the computation time is taken into account. This reflects the effect on the energy efficiency of the algorithm. Each simulation contains 100 Monte Carlo runs. The simulations are done using MATLAB R2019b.

5.5.1. Dataset 1

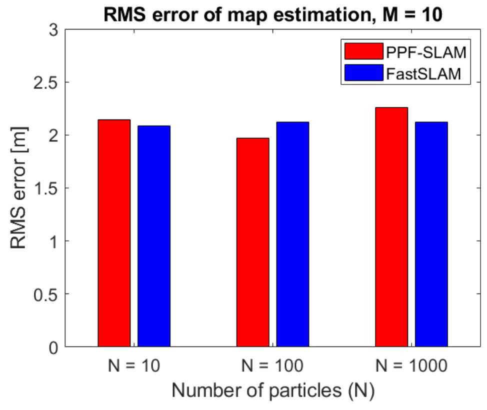
The first dataset contains 10 landmarks. The hypotheses is that splitting the observation model into different landmark observations does not have major impact in the simulations, since there are not a lot of landmarks that can be seen at once. This will thus not influence the accuracy and the computation time. However, computing the adjacency matrix every time step takes probably more time than the time that is gained with computing the rest parallelized.

The ARMS error of the trajectory estimation and the RMS error for the map estimation for FastSLAM and PPF-SLAM is shown in Figure 5.2 for different number of particles averaged over 100 Monte Carlo runs. From these results it is clear that FastSLAM gives a more consistent result. The accuracy of PPF-SLAM follows no clear trend with thus amount of landmarks.

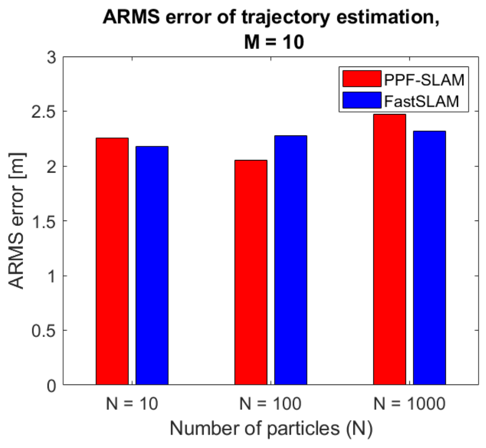
The other metric of interest is the run time of the two algorithms for different numbers of particles. This result is shown in Figure 5.3. From these results it is clear that the PPF-SLAM is for all three cases significantly slower than the FastSLAM algorithm. This is due to the computation time of the adjacency matrix every time step including computing the eigenvalue decomposition of the Laplacian.

5.5.2. Dataset 2

The second dataset contains 50 landmarks. The hypothesis is that the time complexity of the PPF-SLAM simulations is significantly lower than the time complexity of FastSLAM, since the reduction in processing time of the distributed landmark observations is more significant compared to the first dataset.



(a) RMS error of the map estimation containing 10 landmarks averaged over 100 Monte Carlo runs.



(b) ARMS error of the trajectory estimation in a map containing 10 landmarks averaged over 100 Monte Carlo runs.

Figure 5.2: Comparison of the performance of PPF-SLAM and FastSLAM on the first dataset containing 10 landmarks averaged over 100 Monte Carlo runs.

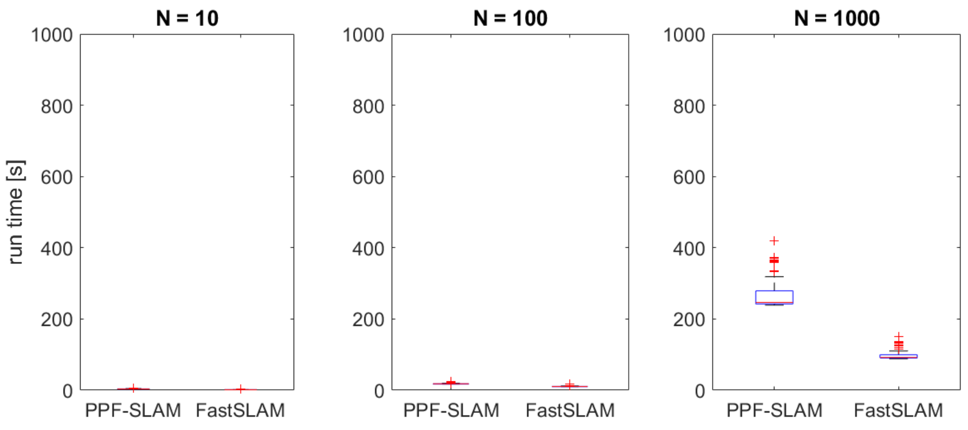


Figure 5.3: Comparison between the run time of FastSLAM and PPF-SLAM for the first dataset containing 10 landmarks over 100 Monte Carlo runs.

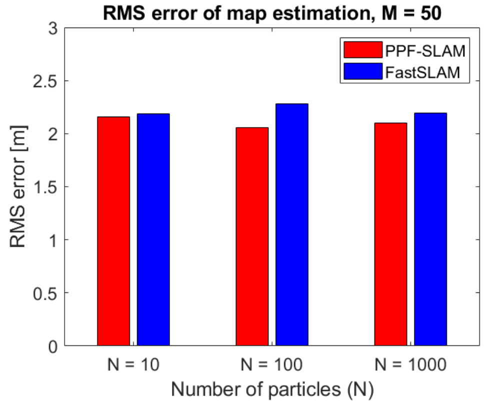
The results of the second dataset are shown in Figure 5.4. The accuracy of the map estimation is in all three cases a little bit better for PPF-SLAM, but this improvement is not visible yet in the error of the trajectory estimation.

The time complexity is visualized in Figure 5.5. As expected, the run time of PPF-SLAM for 10 and 100 is shorter than for FastSLAM. Interesting, however, is that the run time for 1000 particles is for both algorithms almost equal. So also in the case of 50 landmarks, the gain in computation time is not visible yet, whereas the accuracy stays more or less equal for both algorithms.

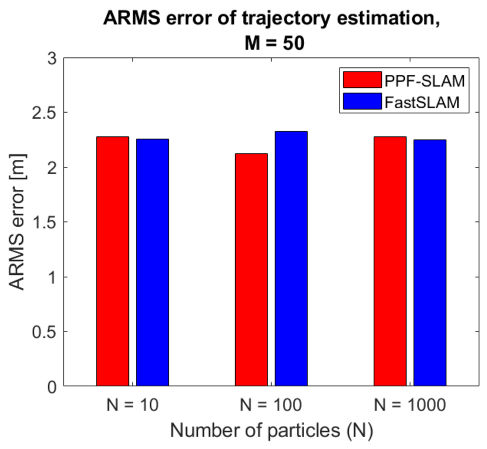
5.5.3. Dataset 3

The third dataset contains 100 landmarks randomly distributed over the map. The hypothesis is that the run time for this dataset will be lower for PPF-SLAM than for FastSLAM, because of the parallel computed landmark observations.

The performance of the map estimation and trajectory estimation are shown in Figure 5.6. For this dataset the two algorithms have the similar performance in terms of accuracy, it even is slightly in favor of the PPF-SLAM. Especially for the trajectory estimation with 1000 particles the difference is significantly visible.



(a) RMS error of the map estimation containing 50 landmarks averaged over 100 Monte Carlo runs.



(b) ARMS error of the trajectory estimation in a map containing 50 landmarks averaged over 100 Monte Carlo runs.

Figure 5.4: Comparison of the performance of PPF-SLAM and FastSLAM on the second dataset containing 50 landmarks averaged over 100 Monte Carlo runs.

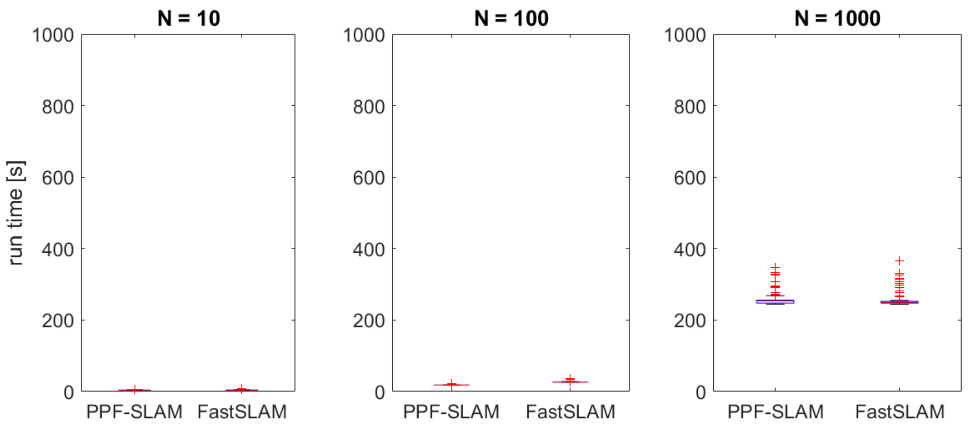


Figure 5.5: Comparison between the run time of FastSLAM and PPF-SLAM for the second dataset containing 50 landmarks over 100 Monte Carlo runs.

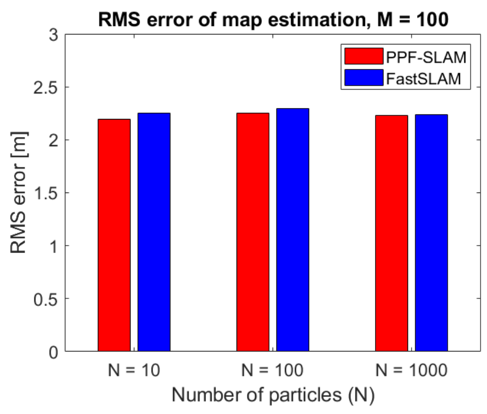
Another improvement of PPF-SLAM is the run time, as shown in Figure 5.7. In all three cases the run time of PPF-SLAM is lower than the run time of FastSLAM. So with improvement of both the accuracy and run time of PPF-SLAM, the increase of landmarks and particles looks like a trend that will lead to a more energy-efficient algorithm.

5.5.4. Dataset 4

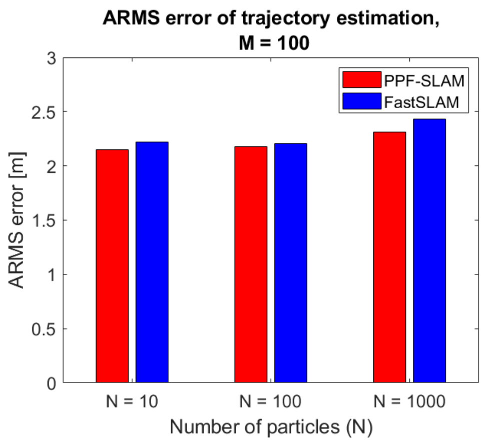
The fourth dataset contains 100 landmarks as well, however, in this dataset the landmarks are placed along the trajectory. This would probably not influence the run time compared to dataset 3, but might influence the performance.

The performance of the two algorithms is shown in Figure 5.8. This figure shows that the performance of PPF-SLAM has not significantly improved compared to the third dataset. So this particular type of dataset has no advantage for PPF-SLAM over an arbitrary distributed map.

The run time of the two algorithms for this dataset is shown in Figure 5.9. As expected, the run time is similar to the run time for the third dataset. So also in this case PPF-SLAM is more beneficial over FastSLAM, but this particular structured dataset has no influence on the performance compared to arbitrarily distributed landmarks.



(a) RMS error of the map estimation containing 100 landmarks averaged over 100 Monte Carlo runs.



(b) ARMS error of the trajectory estimation in a map containing 100 landmarks averaged over 100 Monte Carlo runs.

Figure 5.6: Comparison of the performance of PPF-SLAM and FastSLAM on the third dataset containing 100 landmarks averaged over 100 Monte Carlo runs.

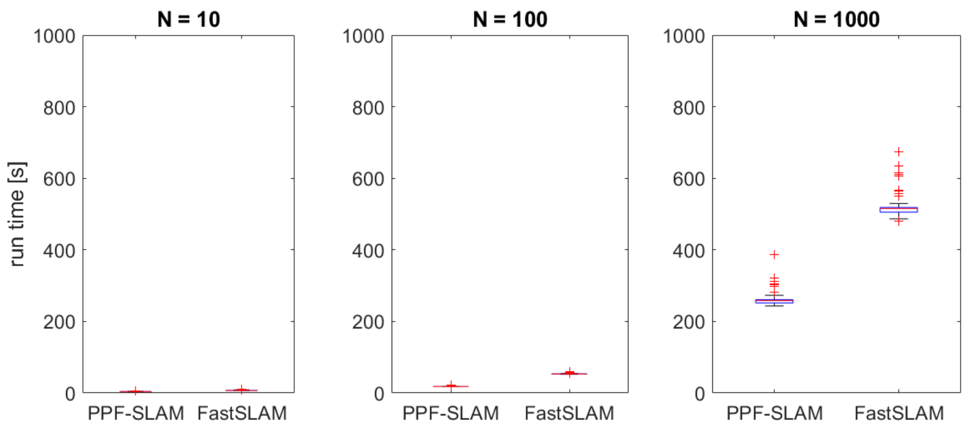
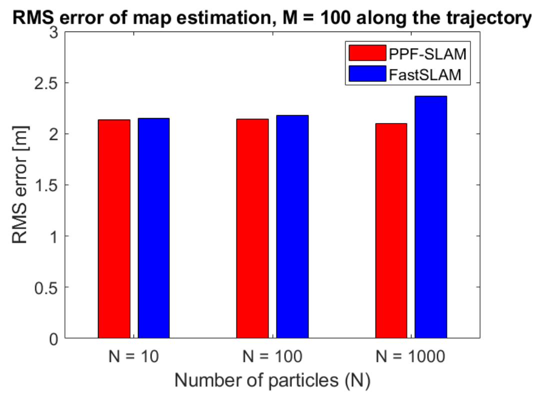
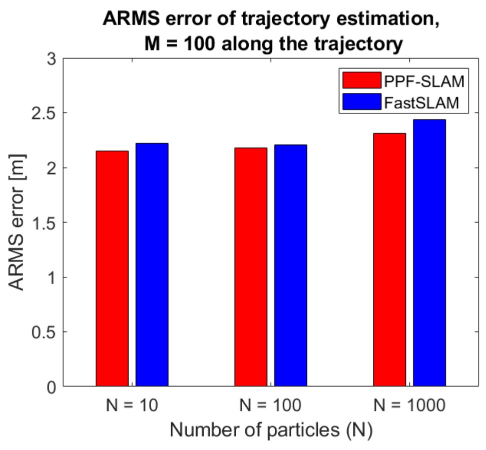


Figure 5.7: Comparison between the run time of FastSLAM and PPF-SLAM for the third dataset containing 100 landmarks over 100 Monte Carlo runs.



(a) RMS error of the map estimation containing 100 along the trajectory landmarks averaged over 100 Monte Carlo runs.



(b) ARMS error of the trajectory estimation in a map containing organized 100 landmarks averaged over 100 Monte Carlo runs.

Figure 5.8: Comparison of the performance of PPF-SLAM and FastSLAM on the fourth dataset containing 100 landmarks along the trajectory averaged over 100 Monte Carlo runs.

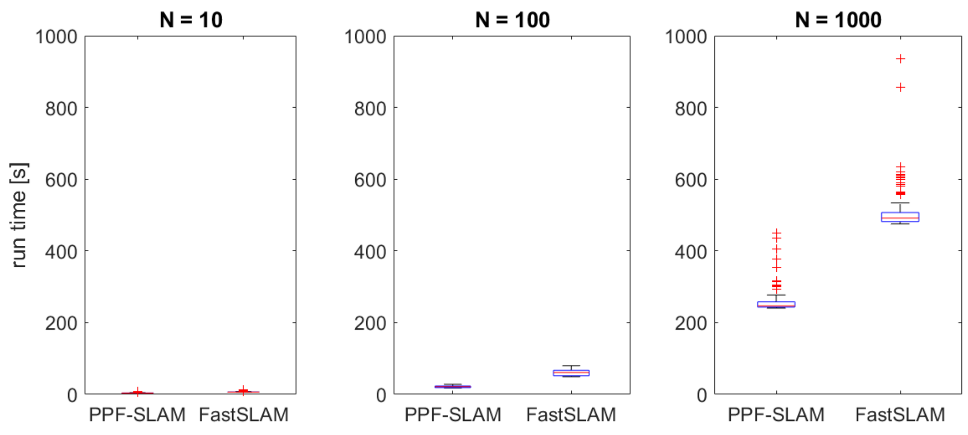


Figure 5.9: Comparison between the run time of FastSLAM and PPF-SLAM for the fourth dataset containing 100 landmarks along the trajectory over 100 Monte Carlo runs.

5.5.5. Time complexity

As seen in the results of the algorithm, the time complexity of PPF-SLAM is dominated by other parameters than in the case of FastSLAM. The time complexity of FastSLAM compared to PPF-SLAM is increasing faster with an increasing number of landmarks, whereas the time complexity of PPF-SLAM is more dominated by the number of particles. The bottleneck for PPF-SLAM is mostly the computation of the adjacency matrix and its eigendecomposition, so this scales faster with a larger number of particles than the time complexity of FastSLAM.

5.5.6. Memory

To run this algorithm, a bigger memory is required for the robot, since it needs to compute the adjecency matrix and do the eigendecomposition. These operations generate information that is not used in FastSLAM, whereas the information for FastSLAM is also required for PPF-SLAM and therefore requires more memory than FastSLAM.

5.6. Conclusion

The accuracy of PPF-SLAM, measured as the RMS error of the map estimation and ARMS error of the trajectory estimation, is in most cases similar to the accuracy of FastSLAM. In some cases, there is a slightly better result for PPF-SLAM, especially for a higher number of particles.

On the other hand, in terms of time complexity of the two algorithms there is a significant difference between the two algorithms. Only for a dataset with just 10 landmarks, FastSLAM is faster, but in all other cases PPF-SLAM computes the map and trajectory estimation significantly faster. So without loss of accuracy, even with a little in gain in accuracy sometimes, but with a huge gain in computation time, this algorithm has shown big potential to become an energy-efficient solution to the SLAM problem.

5.7. Future work

The results shown in this chapter are only theoretical results that can be gained by the proposed algorithm. These results are still language and platform independent. Choosing a suitable platform and language might be even more beneficial for the performance of this algorithm. Both the FastSLAM and PPF-SLAM algorithms can perform tasks in parallel for the different particles. A part of the PPF-SLAM that can be parallel computed is the building of the κ nearest neighbor graph every time step [5, 7]. However, this will not directly effect the energy consumption, since the number of memory accesses and number of instruction will stay constant, but it will reduce the run time.

This method would be beneficial for a multi-agent system, since the information that is shared between the agents is kept as small as possible. In theory, only the Laplacian transform coefficients α need to be communicated between the agents. This can be done using a gossip protocol [29]. A gossip protocol will keep the communication overhead between the multiple agents as low as possible, such that the extra power consumption for communication will outweigh the lower energy consumption and memory load per robot for exploring the whole environment. However, this method computes the adjacency matrix every time step and to do so, it is assumed that every agents know the map estimation and weights of all particles of all other agents, this is a problem that still needs to be overcome.

6 Conclusion and future work

This chapter concludes the two proposed algorithms compared to the existing FastSLAM algorithm. The focus of these results will be on the computational complexity, as measure of the energy efficiency of the algorithms. Also some recommendations for future work building onto these algorithms are made in this chapter.

6.1. Conclusion

The goal of this research was to improve a particle filter based SLAM algorithm to reduce the energy needed to run the algorithm. This question is answered through two subquestions:

- How can the available information about the landmarks be used more efficient?
- How can the computational complexity and run time be reduced without compromising the accuracy of the algorithm.

The work in this thesis is platform and language independent and is only focused on the overall performance indicators, such as run time, memory and estimation accuracy.

The baseline of this thesis is a FastSLAM algorithm for a single robot simulated on a landmark-based environment. These options are chosen because they require in general less resources than other well-known SLAM paradigms. The FastSLAM algorithm is in this thesis compared with two new algorithms; constrained FastSLAM and parallelized particle filter-SLAM.

6.1.1. Constrained FastSLAM

The first algorithm, constrained FastSLAM, uses the available information of the landmark positions to formulate a constrained problem for the trajectory estimation. These spatial constraints lead to a more accurate region to sample from. The simulations of constrained FastSLAM compared to unconstrained FastSLAM show that a higher number of particles generate a more accurate estimation of the trajectory and environment. Also the KLD estimator shows an improvement in accuracy for a high number of particles and a high number of landmarks.

Regarding the overall performance of the proposed algorithm, the constrained FastSLAM algorithm provides similar results as unconstrained FastSLAM considering the accuracy of the algorithm. However the time complexity of constrained FastSLAM is higher than in the case of unconstrained FastSLAM. The simulations show that there is a trend that more particles will give a more accurate estimation. But even though there is no major difference in accuracy, the constrained FastSLAM algorithm is not considered more energy efficient than FastSLAM, since the time complexity increases.

6.1.2. Parallelized Particle Filter SLAM

The second algorithm is the parallelized particle filter. This method splits the observation model into multiple landmark observations. These measurements are then used in parallel to compute an estimation of the map and trajectory. The weights are obtained by sharing this information in a graph-based method. Even though not every particle has access to all information every time step, the accuracy of the trajectory and map estimation is not influenced by it, due to an efficient communication system where the information is shared between the particles.

This proposed algorithm has reduces the computation time of the algorithm significantly. This results in less computation time and thus less energy consumption. The performance has slightly improved in terms of accuracy for this proposed algorithm. Especially with an increasing number of particles, the accuracy of PPF-SLAM starts to outperform FastSLAM. So without loss of estimation accuracy, the computational time and thus the energy efficiency has been decreased considerably. So PPF-SLAM is considered to be more energy-efficient than FastSLAM without the loss of accuracy.

6.1.3. Energy efficiency

In the absence of a good metric to measure the power consumption of an algorithm, the two proposed algorithms in this thesis have been tested on their computation times as a measure for the energy efficiency of the algorithms. Except for this metric, the algorithms have also been tested on their accuracy, to see whether the reduction in computation time will not influence the accuracy of the algorithms.

The constrained FastSLAM has a higher complexity than FastSLAM, without major improvement in the accuracy of the algorithm. Thus this algorithm does not lead to a lower energy consumption.

On the other hand, PPF-SLAM has a lower time complexity without compromising on the accuracy of the estimation.

6.2. Future work

The proposed algorithms have been tested under on four synthesized datasets under certain assumptions, such as a single robot system with limited number of landmarks and particles and the data association and loop closure has been out of the scope of this thesis. So there are possible improvement alternatives for future research to increase the performance of the algorithm.

6.2.1. Constrained FastSLAM

The bottleneck to make constrained FastSLAM more energy-efficient is the computational complexity of the algorithm. This increases faster than unconstrained FastSLAM with an increasing number of landmarks. When this challenge is tackled, the algorithm can be expended by adding chance-constrained optimization where the uncertainty of the landmark position is taken into account in the spatial constraint optimization. This will result in a more accurate algorithm.

6.2.2. PPF-SLAM

PPF-SLAM has shown big improvement in run time compared to FastSLAM. This results in a more energy-efficient algorithm. This algorithm has now been simulated in theory, but can perform even better if it is implemented on the right platform, because a lot of the computations, such as the eigendecomposition and computing the adjacency matrix can be done in parallel.

6.2.3. Multi-agent system

Both algorithms have now been tested on a single agent system, but could be expanded to a multi-agent system. This would extent the exploration area without changing the energy consumption of a single robot too much. Especially the PPF-SLAM algorithm is suitable for a multi-agent system, because the information that needs to be distributed over the agents is kept small. This advantage can be used to keep the communication overhead using a gossip protocol as low as possible. However, this method assumes that all agents have knowledge over the trajectory and map estimation of the other agents, which is a problem that should be overcome, to be able to gain from the advantages.

6.2.4. Active autonomous exploration

The algorithms have been tested on a dataset with a given trajectory for the robot. This passive exploration has provided a measure to compare the different algorithms, however, both algorithms will be even more beneficial if they are incorporated in an active exploration algorithm.

6.2.5. Implementing on a platform

Whether these algorithms will provide the performance that is reached in theory, could be tested in practice by implementing it on a test robot. This also has the advantage that, as discussed before, some computations can be done in parallel to reduce the computation time, however, this does not necessarily reduce the energy consumption, but makes the overall computation faster.

Bibliography

[1] 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: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[2] S. Challa and N. Bergman. Target tracking incorporating flight envelope information. In *Proceedings of the Third International Conference on Information Fusion*, volume 2, pages THC2/22–THC2/27, 2000.

[3] A. Chatzigeorgiou and G. Stephanides. Energy metric for software systems. In *Software Quality Journal*, volume 10, page 355–371, 2002.

[4] R. Chou, Y. Boers, M. Podt, and M. Geist. Performance evaluation for particle filters. In *14th International Conference on Information Fusion*, pages 1–7, 2011.

[5] M. Connor and P. Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):599–608, 2010.

[6] J. Demmel, I. Dumitriu, and O. Holtz. Fast linear algebra is stable. *Numerische Mathematik*, 108 (1):59–91, Oct 2007. ISSN 0945-3245.

[7] C. Deng and W. Zhao. Fast k-means based on knn graph, 2017.

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

[9] A. Doucet and A. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12, jan 2009.

[10] A. Dubey. Stereo vision - facing the challenges and seeing the opportunities for adas applications, 2020.

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

[12] S Ergasheva, I Khomyakov, A Kruglov, and G Succil. Metrics of energy consumption in software systems: a systematic literature review. *IOP Conference Series: Earth and Environmental Science*, 431, feb 2020.

[13] B. Ferris, D. Fox, and N. Lawrence. Wifi-slam using gaussian process latent variable models. volume 7, pages 2480–2485, jan 2007.

[14] C. Fox, M. Evans, M. Pearson, and T. Prescott. Tactile slam with a biomimetic whiskered robot. In *2012 IEEE International Conference on Robotics and Automation*, pages 4925–4930, 2012.

[15] J. Giacalone, L. Bourgeois, and A. Ancora. Challenges in aggregation of heterogeneous sensors for autonomous driving systems. In *2019 IEEE Sensors Applications Symposium (SAS)*, pages 1–5, 2019.

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

[17] M. Henein, J. Zhang, R. Mahony, and V. Ila. Dynamic slam: The need for speed. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2123–2129, 2020.

[18] C. Kim, R. Sakthivel, and W. Chung. Unscented fastslam: A robust algorithm for the simultaneous localization and mapping problem. pages 2439–2445, apr 2007.

[19] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[20] J. Levinson and S. Thrun. Robust vehicle localization in urban environments using probabilistic maps. In *2010 IEEE International Conference on Robotics and Automation*, pages 4372–4378, 2010.

[21] D Maithripala, D. H. S. Maithripala, and S Jayasuriya. A geometric approach to dynamically feasible, real-time formation control. *Journal of Dynamic Systems Measurement and Control*, 133, mar 2011.

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

[23] M. Montemerlo, S. Thrun, and W. Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 695–701, 2002.

[24] J. Nieto, J. Guivant, E. Nebot, and S. Thrun. Real time data association for fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 1, pages 412–418, 2003.

[25] G. Pandey. An information theoretic framework for camera and lidar sensor data fusion and its applications in autonomous navigation of vehicles. jan 2014.

[26] L. Paull, S. Saeedi, M. Seto, and H. Li. Auv navigation and localization: A review. *IEEE Journal of Oceanic Engineering*, 39(1):131–149, 2014.

[27] F. Pei, X. Wu, and H. Yan. Distributed slam system using particle swarm optimized particle filter for mobile robot navigation. In *2016 IEEE International Conference on Mechatronics and Automation*, pages 994–999, 2016.

[28] V. Pirard and E. Sviestins. A robust and efficient particle filter for target tracking with spatial constraints. In *Proceedings of the 16th International Conference on Information Fusion*, pages 30–37, 2013.

[29] M. Rabbat, M. Coates, and S. Blouin. Graph laplacian distributed particle filtering. In *2016 24th European Signal Processing Conference (EUSIPCO)*, pages 1493–1497, 2016.

[30] D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics Automation Magazine*, 18(4):80–92, 2011.

[31] R. Siegwart and D. Scaramuzza I.R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT press, 2004.

[32] G. Stewart. A krylov-schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23, aug 2000.

[33] S. Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[34] S. Thrun and D. Fox W. Burgard. *Probabilistic Robotics*. MIT press, 2005.

[35] C. Wang, C. Thorpe, and S. Thrun. Online simultaneous localization and mapping with detection and tracking of moving objects: theory and results from a ground vehicle in crowded urban areas. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 1, pages 842–849, 2003.

[36] J. Wang and Y. Takahashi. Particle filter based landmark mapping for slam of mobile robot based on rfid system. In *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 870–875, 2016.

[37] Q. Wang, S. R. Kulkarni, and S. Verdu. Divergence estimation for multidimensional densities via k-nearest-neighbor distances. *IEEE Transactions on Information Theory*, 55(5):2392–2405, 2009.

[38] X. Wang and P. Li. Improved data association method in binocular vision-slam. In *2010 International Conference on Intelligent Computation Technology and Automation*, volume 2, pages 502–505, 2010.

[39] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid, and J. Tardós. A comparison of loop closing techniques in monocular slam. *Robotics and Autonomous Systems*, 57(12):1188–1197, 2009. ISSN 0921-8890.

[40] D. Won, S. Chun, S. Sung, Y. Lee, J. Cho, J. Joo, and J. Park. Ins/vslam system using distributed particle filter. *International Journal of Control Automation and Systems*, 8:1232–1240, dec 2010.

[41] X. Zhao and S. Zhang. Distributed strong tracking unscented particle filter for simultaneous localization and mapping. In *Proceedings of the 33rd Chinese Control Conference*, pages 978–983, 2014.