Line-Adaptive Monte Carlo Localization

Improving self-localization of a mobile robot in barns

L.J. Bontan

TUDelft

LELY

Line-Adaptive Monte Carlo Localization

Improving self-localization of a mobile robot in barns

by

L.J. Bontan

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday October 2, 2023 at 3:00 PM.

Student number:	5423805	
Project duration:	November 21, 2022 – October 2, 2023	
Thesis committee:	Dr. J.F.P. Kooij,	TU Delft, supervisor
	Dr. M. Kok,	TU Delft
	Ir. M. den Toom,	Lely Technologies N.V.

Abstract

Robots are increasingly deployed in various locations to automate tasks, including in barns. However, in barns cows can obstruct the sensors such as LiDAR or camera, leading to a lack of environmental information. As a result, the robot's localization system only relies on odometry at those moments, introducing additional uncertainty to the robot's pose. When the visibility of the environment is restored, the robot may mistakenly believe it is in a location that does not correspond to its actual position.

The first contribution of this master thesis is a novel method for improving the self-localization in barns by implementing a line detection algorithm which is called Line Adaptive Monte Carlo Localization (LAMCL). The novelty is that only line segments are used to detect a localization error instead of corners between different line segments. It also retains the robot's current pose to filter out fault-detected localization errors. In addition, the new approach is applied in a dynamic environment. The proposed method combines the Split-and-Merge line detection algorithm with AMCL. The detected line segments are compared with the walls in the environment to identify localization errors. When an error is detected, the robot's pose is adjusted by placing a section of the particles at the location of the error. In this way, the robot can find its true location again.

The second contribution is a new dataset. This new dataset, called DataCow, consists of four recorded routes in a barn with GT on a handful of spots to evaluate the self-localization. DataCow includes the pseudo-2D LiDAR scans and the odometry of a robot driving through a barn. This dataset is used to evaluate the new self-localization method LAMCL. Through the experiments, it has been discovered that this new method improves the system's recovery ability, but the accuracy and precision are compromised. The influence of the hyperparameters of the new method is also tested.

Keywords: self-localization, AMCL, line detection, mobile robot, agriculture robotics

Preface

This master thesis is the final project to complete my Master Robotics at The Technical University in Delft which took place from November 2022 to August 2023. I would like to thank some people who helped me fulfill this project. First, I would like to thank the people at Lely Technologies who made it possible to do my thesis there. Especially product developer Matthijs den Toom for the guidance and support. It was nice to be able to talk to someone about the technical content of the project. Secondly, I would like to thank my supervisor from TU Delft, Julian Kooij. It was good to be able to talk with him about the process of doing research with experiments to support the findings. Thirdly, I would like to thank my parents for their support during my time as a student. Finalizing my master's thesis means the end of my time as a student. The past years gave me the opportunity to broaden my knowledge and I would like to thank everyone who helped me accomplish this.

L.J. Bontan
Maassluis, August 2023

Nomenclature

$AMCL$	Adaptive Monte Carlo Localization
GT	Ground Truth
ICP	Iterative Closest Point
$IEPF$	Iterative end-point fit
KF	Kalman Filter
$LAMCL$	Line Adaptive Monte Carlo Localization
$LiDAR$	Light Detection and Ranging
MAE	Mean absolute error
MCL	Monte Carlo Localization
$RBPF$	Rao-Blackwellized particle filter
$RMSE$	Root Mean Squared Error

Contents

List of Figures

List of Tables

1 Introduction

1.1. Background

The world's population is expected to increase by nearly 2 billion persons in the next 30 years according to the United Nations [57] and conservative estimates show that demand for food will increase 70% over the next 30 years [30]. To meet this demand, more food must be produced in the future. Lely is a robotics company that focuses on farming innovations. The automation of the first link of the food chain is the beginning to meet the increasing demand for food. One of Lely's robots drives around autonomously in a barn to keep the barn clean with pre-programmed routes. However, to operate properly the robots need to localize with reference to a map. This is challenging in barns because cows are able to move around freely and might obscure the view of the surrounding area as a group. The environment is also characterized by long corridors that look mostly the same which makes it difficult to determine the location from a glance. More specifically, in this thesis project a barn of approximately 600m^2 with 120 cows is considered. The robot itself is 1.2m wide, 1.5m depth, and 2m high and is programmed to drive to certain waypoints with the two wheels it has, which makes it a holonomic system. For localization, the robot currently uses a stereo camera which produces a 3D point cloud. An algorithm converts the point cloud to a planar LiDAR scan. A 2D map is the input of the Adaptive Monte Carlo Localization (AMCL) algorithm together with the planar LiDAR scan and odometry of the robot. Odometry is the movement of the robot seen from within the robot itself. This is done with a combination of IMU data and the displacement of the wheels. To understand why AMCL struggles with the barn environment, the next section will give a more detailed explanation of its components, and why they can fail in dynamic barn environments. For generalization, Lely's robot is referred to as a mobile robot in a barn in the rest of this thesis project.

Adaptive Monte Carlo Localisation

AMCL (Adaptive Monte Carlo Localisation)[1] is an algorithm used by robots to find their own location in a known environment. Figure 1.1 shows the system with its inputs, internal AMCL process, and output. AMCL is a modified version of MCL which consists of three steps: the motion update, sensor update, and resampling. In the beginning, the first step is different from the motion update. Instead of using a motion update, a large number of particles are spread across the map. These particles represent the poses where the robot might be. Besides the change in the first step in the initiation, the sensor update and resampling are the same.

1. The **motion update** is the first step. All particles are moved with the motion measured by odometry and an extra Gaussian noise. These particles are called the prior of the algorithm.
2. The **sensor update** uses the sensor measurements to iterate over all the particles and checks the probability of the robot being in each particle's place. Each particle is given a weight to represent how well the particle matches the sensor measurements with the given map. Since the particles are now compared with the likelihood of being in a particular pose the particles are now called the posteriors. When there are too few measurement points available the uncertainty in the location grows. When this occurs the particle cloud will spread.

3. With **resampling**, new particles are generated based on the old ones to represent the highest-scoring particles after which another iteration starts with the motion update.

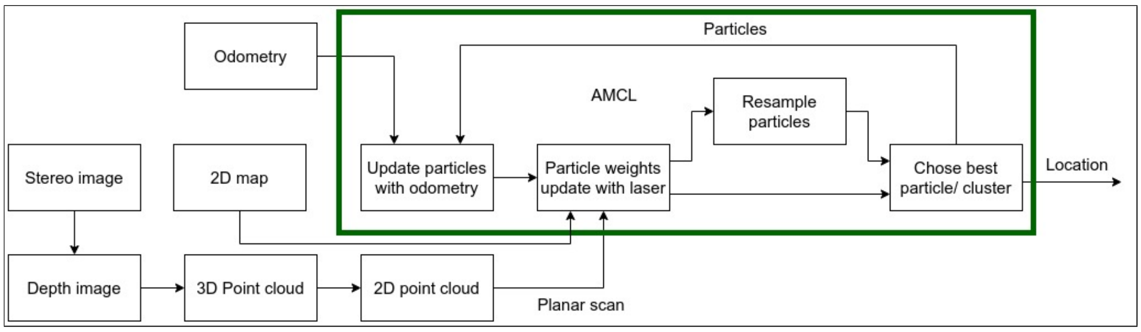


Figure 1.1: Pipeline of the localization system. The green box is included in the AMCL algorithm

An important feature of this algorithm is that the robot can autonomously locate itself. This can be done even when the localization has gone wrong a couple of meters after the robot has been in operation for several hours. Finding its own location again is useful if a robot has to drive autonomously all the time. In fact, a localization error does not require a person to assist in order to get the robot operational again.

The mobile robot drives around in a barn with cows. This is an indoor and dynamic environment. A barn with cows has the characteristic that there are many long straight stretches with the same kind of walls of about 0.2m or 0.7m high. There are also fences to stop the cows which also have a bar at 0.7m. So the cows walk freely in the barn, making the environment dynamic. Many cows can stand together in certain places, for example near the milking robot. In these places, the robot loses its external sensor information. Only with the internal sensors, there is uncertainty in the robot's movement. The longer the interruption of the external sensors, the greater the uncertainty of self-localization becomes. With too much uncertainty, the robot may be unable to find back where it is and has to do a so-called recovery.

1.2. Problem statement

The robot only knows it is at a different position from the one received from AMCL when it runs into a wall or takes a turn too tightly. This is not desirable, because a recovery from an incorrect location estimate takes time. To identify the main causes of these temporal location errors, a day's worth of data from the robot navigating independently through the barn with cows was examined. During this analysis, the causes, as well as the consequences of a localization error were investigated.

The two main problems are shown in figure 1.3 and 1.4. Figure 1.3a shows that the measurements are in a straight line and represent the wall. The measurements are rotated relative to the wall. However since the robot does not have particles that are in the right spot, AMCL does not correct this. In figure 1.3b the particle cloud is split into two parts. This may have occurred because AMCL has had to make a decision before and, for example, mistook a cow for a wall. Given the localization, the other cluster would be a better choice from an external point of view. However, the algorithm does not know the difference in the rotation if the total error is the same as shown in figure 1.2. In the situation of figure 1.3b the rotated chosen cluster has a lower error than the cluster parallel to the wall. The right cluster is too dense and is too much forwards to correct to the right spot. This is also seen in figure 1.4 where the wall and corners are visible in the measurements for a human eye, but the robot has no particles in the right place to correct this.

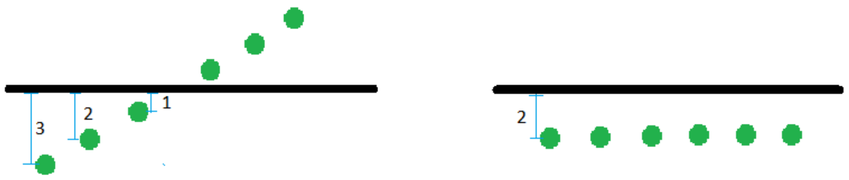
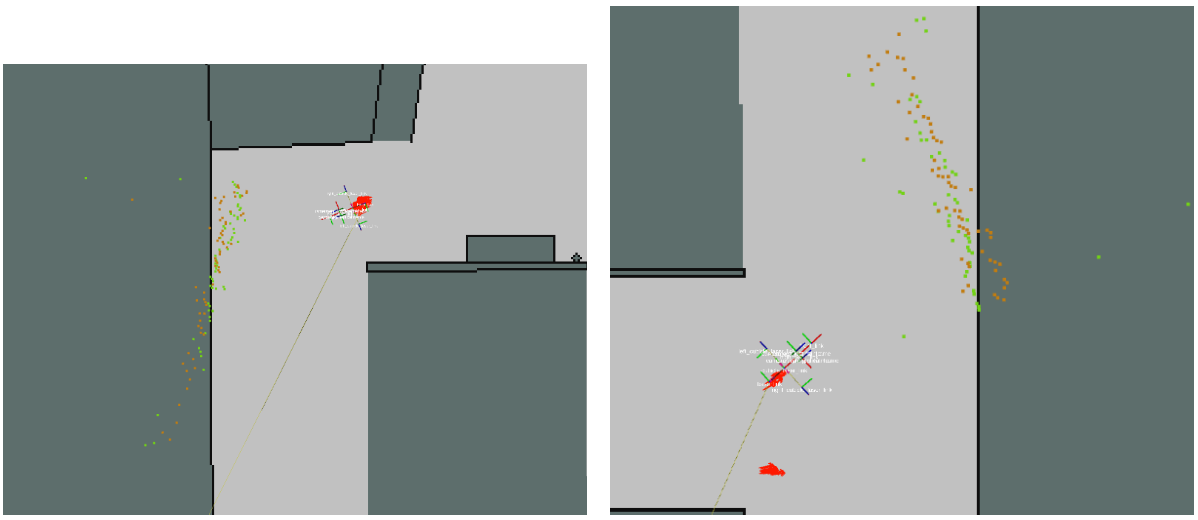
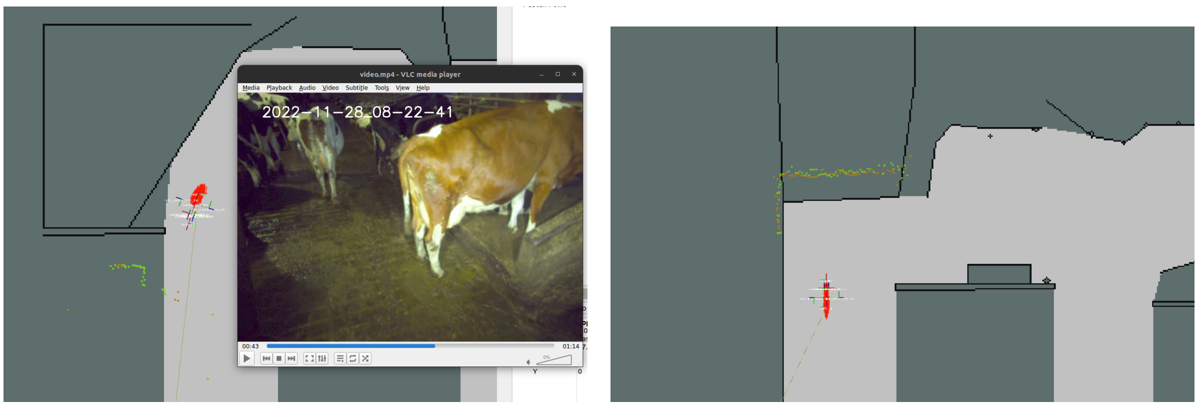


Figure 1.2: Black line represents the wall with the green dots measurements of the wall. Both situations have a total error of 12 although the actual location of the robot is completely different



(a) Measurements, as seen from particle cloud, are rotated (b) Particle cloud broke up, but the wrong cluster is chosen

Figure 1.3: Top view of the map with the robot, particles, and measurement points. The particles are represented in red with the robots' transform frames on top. the green and orange dots are planar lidar measurement points at respectively a height of 0.2m and 0.7m. The map is a combination of obstacles represented at a height of 0,2m and 0,7m


video.mp4 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
2022-11-28_08-22-41
00:43
01:14

(a) Corner measurements do not overlap with the map (b) Wall measurements do not overlap with the map

Figure 1.4: Location could not be restored because particles are too dense

The challenge is that this robot only has a stereo camera and a map of its environment to determine its position. So when cows walk in front of the robot, no fixed barn structures can be seen. AMCL is specialized in cases when there is uncertainty in the measurements. The particle cloud will expand when the uncertainty of the localization grows. However, the algorithm converges too fast to a specific spot in some cases where enough measurements are available again. This results in situations where there are no particles in the real location of the robot later in the route. This causes situations where the measurements can not be in line with the map of the environment and AMCL does not restore the localization.

In the related work chapter, the possibilities of re-initializing particles at the location of the robot will be investigated. Additionally, other ways to improve localization are being explored. To improve localization, it is possible to look at how the measurement points can be used to modify the particle cloud when it is not spread enough to correct the error. Therefore, a part of the research will focus on how the information about the measurement points can be extracted from the measurement points and used to reduce localization offsets by adjusting the particles' location while resampling.

1.3. Research questions

The robot in the barn uses pseudo planar LiDAR, which means the measurements are a 2D LiDAR scan converted from a stereo camera point cloud. In order to make the final result work with this limited input and process power, the focus will only be on planar LiDAR to narrow down the scope of this master thesis.

The main research question is:

Does using extracted lines from a planar LiDAR improve the accuracy, precision, and robustness of self-localization with AMCL for a mobile robot in a barn?

The research question is separated into multiple sub-questions to answer the main research question.

1. *Can the localization with AMCL be improved by correcting for offsets by resampling particles through additional line extraction and matching between those line segments and map?*
2. *How can the accuracy, precision, and robustness of self-localization be tested and evaluated in a real-world barn environment?*
3. *How sensitive is the new self-localization method to hyperparameter choices?*

1.4. Document structure

Chapter 2 has an overview of methods that have been done to improve self-localization. This also covers different line algorithms that could be used and methods to evaluate the performance of a localization system. This is also where the research gap in the literature becomes clear and what this master thesis will add. Chapter 3 explains the three parts of the contribution with why they are needed, what they do, how they do it, and a description of the parameters they use. In Chapter 4, the new dataset and experiments are described, explained, and their results are shown. Finally, Chapter 5 discusses the results and draws conclusions closing with recommendations on what could be done to improve it even further.

2 Related work

In this section, three topics are covered to see what related research has been done. The first topic is improving localization in MCL. This is examined because the robot uses AMCL which is an improvement to MCL. These findings can be used to see what other improvements have been made. Secondly, different line algorithms are examined because the goal is to improve localization with extracted lines. The lines are extracted from pseudo-LiDAR, so the different factors that can cause errors are also examined. Thirdly, we will look at how the performance of the self-localization can be tested and evaluated to eventually compare the new method with the current method.

2.1. Localization error improvement

Self-localization is widely used for robots that need to drive autonomously in an environment [28, 50, 40]. The robot in the barn uses AMCL which is a modified method compared to MCL. Likewise, there are other methods of self-localization that have an adjustment with the goal of improving localization. The current algorithm works with particles, hence variations of self-localization methods related to the update step of the particles will be considered first. Thereafter, other possible methods will also be discussed.

2.1.1. Change of localization method

Adaptive Monte Carlo localization

AMCL [21] is a variation on MCL. Instead of using variations in the order of the algorithm, AMCL changes the way the particle update state works. MCL has a fixed number of particles throughout the whole localization which is dependent on the computer power. So, when there is a high certainty the robot is in a specific spot, the particles are close together. In situations with low certainty, the particle cloud is less dense, because of its spread. AMCL tries to solve this problem by dynamically changing the number of particles in the algorithm. When the uncertainty becomes larger, the number of particles increases as well. This means that the density of the particle cloud is quite constant when the uncertainty in the localization rises. Fox [21] shows that this adapting sample size needs 12 times fewer particles on average to achieve the same error as the MCL approach.

Biased Monte Carlo localization

Biased Monte Carlo difference from MCL in the way it samples particles [56]. Where MCL places new samples at possible locations, Biased-MCL takes into account the extra probability of the robots' dimensions. Given the fact that a robot can not locate itself into the wall, those particles will be filtered out right when the new samples are thrown. This could even end up taking into account the properties of the environment. Such as if the robot is charging, a quite specific pose should be provided to do so. Other non-charging poses are then excluded from the probability of particles.

Unscented Monte Carlo localization

Another variation of MCL that affects the particles is Unscented Monte Carlo localization [18]. The idea of unscented sampling is that samples are drawn with sigma points instead of based on the mean of the samples. Figure 2.1 gives the idea behind this method. Fei J. et al. [19] discovered that adding this method to MCL outperforms regular MCL and Rao-Blackwellized (EPF-MCL). It has an average position error of 0.10m whereas MCL has an average position error of 0.15m both with 15 particles. Even with fewer particles, Unscented-MCL shows better results than MCL.

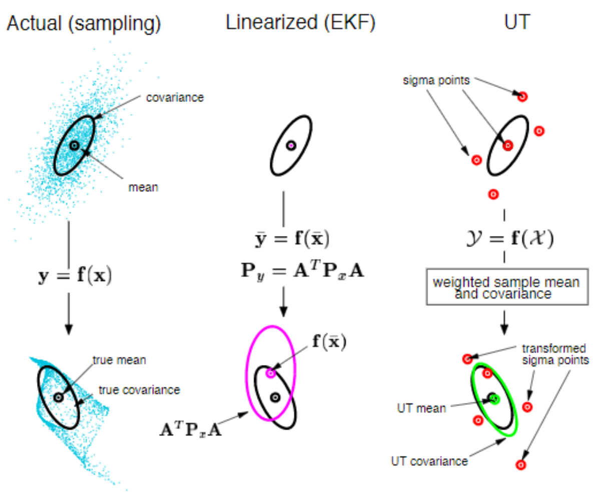

Actual (sampling)
Linearized (EKF)
UT
covariance
mean
sigma points
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathcal{Y} = \mathbf{f}(\mathcal{X})$
weighted sample mean and covariance
true mean
true covariance
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
transformed sigma points
UT mean
UT covariance

Figure 2.1: Unscented transform (UT), the idea behind unscented sampling, figure 1 of [61]

Rao-Blackwellized particle filter

By adding a Rao-Blackwellized (RB) particle filter [26, 44, 27] to MCL, a new type of MCL is created which has a change in the way the particles update. With an RB particle filter, the pose is split into a non-linear position and linear orientation. The basics of an RBPF are compared with EKF and the regular particle filter in figure 2.2. With the computational efficiency of the particle filter and the update steps of an Extended KF the RB particle filter extends the regular MCL and EKF in performance

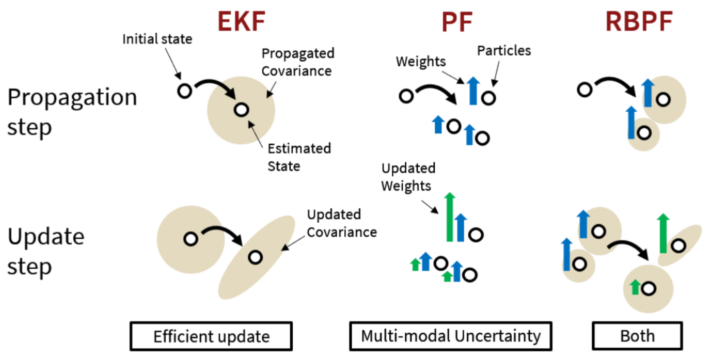


Figure 2.2: Rao-Blackwellized filter compared to Extended KF and a particle filter. Figure 1 in [26]

Normal distributions transform Monte Carlo localization

Normal distributions transform Monte Carlo localization (NDT-MCL) [46] is a variation on MCL with a likelihood field model that changes the way the map is represented. Instead of a grid with a defined grid size, the obstacles are represented as a normal distribution. This continuous representation of the map instead of discrete gives the sensor model a more precise error, which results in more continuous errors for the particles. Here regular MCL has an error of 0.054m, NDT-MCL has an error of 0.014m.

Dual-timescale normal distributions transform Monte Carlo localization

Dual-timescale normal distributions transform Monte Carlo localization [58] is an advanced version of NDT-MCL. It adds a new feature to the map which improves the beam measurements. It keeps track of a dynamically changing second map beside the regular map. This map is updated when there are measurements in a specific spot for some time. This is especially useful in warehouses where there are multiple robots driving through a semi-dynamic environment due to the moving boxes.

2.1.2. Other studies

Liu [33] introduces a new method for accurately locating 3D LiDAR-equipped robots without relying on GNSS. It uses AMCL and multiple sensors, including 3D LiDAR, IMU, and a wheel speed odometer. The method combines data from the wheel speed odometer and IMU to achieve better accuracy using the Extended Kalman Filter. This fused sensor data helps predict the robot's initial position in the AMCL algorithm. Then, the AMCL output at different time points is used in the PL-ICP algorithm to align the 3D laser point cloud. The PL-ICP algorithm also calculates a 3D laser odometer, which corrects the AMCL's initial prediction. This method enhances the performance of 3D LiDAR-based robot localization without the need for GNSS.

Tang [54] published a new algorithm with the addition of a grid. This algorithm shares similarities with MCL, but it differs in how it selects the sampling area and sets up the prediction grid. The key distinction lies in how it constructs anchor boxes and sample boxes while utilizing node mobility to minimize the sampling area. This area is then divided into smaller cells using the prediction grid. The grid is updated based on information gathered from the seed node, and as long as this seed node data is available, the original MCL algorithm is applied. The main change is that each sample now receives the weight of the grid cell it corresponds to. In simpler terms, the algorithm uses a modified version of MCL, adjusting the sampling area and grid setup to enhance localization accuracy based on data obtained from the seed node.

A more basic approach by sampling particles in an area is augmented MCL [56]. Random particles are placed in the particle cloud such that the possibility of particles' absence at the true pose of the robot is reduced. The random particles are sampled from either a uniform distribution of the pose space or the posterior distribution of the measurement [12].

Other studies that improve localization are for example Miguel [35] and Obregon [38] where they use additional GNSS information or Liu [32] which uses neural networks to merge vision and the 2D laser data. Ge [24] uses a text-based MCL method to extract additional information from the environment and use it to improve localization. Where Portugal [42] uses a combination of sensor data fusion and scan matching techniques to make the system perform more smoothly.

2.1.3. Features usage in self-localization

The features found in the measurement points can be used to improve the self-localization of the robot. Kang *et al.* [28] uses the extracted lines in the LiDAR measurements to improve the localization. By matching the detected lines with the map the error in the self-localization can be corrected. Due to the error in the measurements themselves, it is not possible to exclude the complete localization error.

Another way in which features from LiDAR data can be used is for mapping an unknown environment. By converting the data points to lines, the layout of the environment is determined and the walls are located. This map can eventually be used for localization or navigation, for example. In the planar LiDAR data, in addition to walls, corners can also be used for features. Amin *et al.*[3] have a state-of-the-art method of determining corners independently of the line extraction, making their method faster than the standard corner detection in IEPF. When data does need to be clustered it is impossible to find out in advance how many lines there are exactly. Yang *et al.*[63] comes up with a method that solves this problem so that it is not necessary to know in advance how many different lines there are. However, this method is especially suitable when the data points are not delivered as a planar LiDAR scan, but as a 2D point cloud. Another approach is done by Ravankar *et al.*[45]. They use a hopping method through the points combined with Hough transform-based algorithm to determine the lines.

2.1.4. To sum up

There are several variations of Monte Carlo Localization which show an improvement in the localization of a mobile robot. In section 2.1.1, the variations discussed are adaptations of MCL based on a change in how to deal with the particles. Similarly, MCL has already been improved by combining additional sensors or handling input data in a smarter way. Furthermore, Kang *et al.* [28] showed that localization can be improved by using features such as walls and corners in the measurement points.

2.2. Line algorithms

The first sub-question is about line detection algorithms that can be used to extract line structures from planar LiDAR measurements. Before explaining the different methods it is important to understand what LiDAR is and what influences the structure of the planar LiDAR measurement.

Light Detection And Ranging (LiDAR) is an active sensor that uses the time between outgoing and incoming laser signals to calculate how far the near obstacle in a given direction is [15]. These points can be converted to 2D measurement points in the map that can be used as input for self-localization. In the Introduction is mentioned that, when an error in self-localization occurs it could be that the particle cloud converges too quickly to a location that is correct for the algorithm. In those situations, there are no particles in the real location of the robot anymore and the measurement points can no longer be viewed from locations other than those of the particles. Still, it can be seen in figures 1.3 and 1.4 that a good measurement is clearly visible to the human eye. However, this human-observed information is not available to the localization algorithm. In situations where there is a structure in the measurement points visible to the human eye, the algorithm could use this information to find the right pose if the algorithm could also extract this information.

2.2.1. Type measurement errors

In order to find structure in measurement points, it is important to know how the measurement points may differ from the actual situation measured. The measurement points may contain measurement errors that can be divided into four different categories: small measurement noise, errors due to unexpected objects, errors due to failures to detect objects, and random unexplained noise [56].

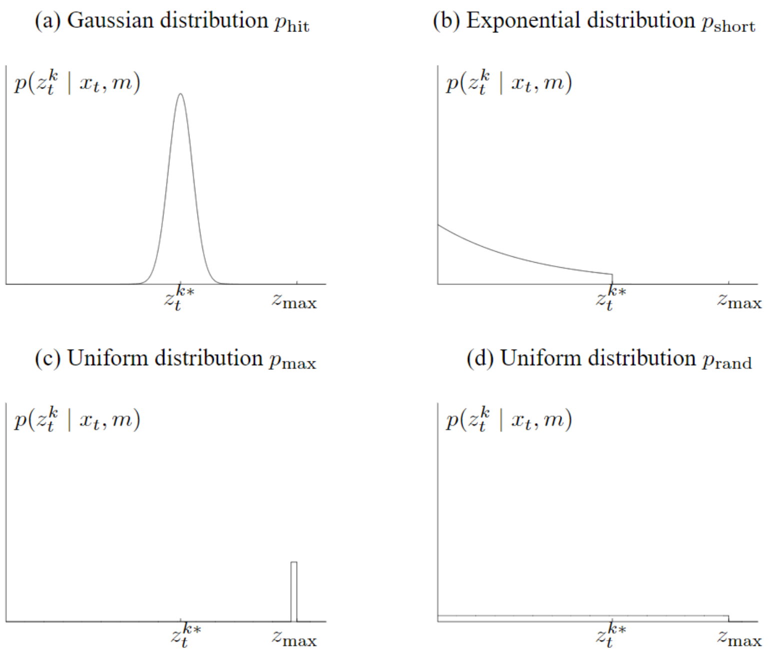


Figure 2.3: Types of different distribution likelihood errors. In each diagram, the horizontal axis corresponds to the measurement z_t^k, the vertical to the likelihood. z_t^{k*} represents the true range of the object. Figure 6.2 in [56]

1. **Small measurement noise** is shown in figure 2.3a. The probability of measuring the object at the true range of the object has a Gaussian distribution named p_{hit}. This is mainly due to the accuracy of the sensor. This p_{hit} due to the noise in measurement is important while finding structure in the measurement points. Then the Gaussian distribution of the p_{hit} has a higher variance a structure could be found in more noisy measurements. The mobile robot in the barn converts stereo camera images into LiDAR measurements, which adds extra uncertainty to the measurements. Measurements further away from the robot have a higher variance in the Gaussian distribution of the p_{hit} due to the pixel size in the cameras.

2. **Errors due to unexpected objects** are shown in figure 2.3b. Especially in a dynamic environment, an object could block the view of the known object given the map. Unknown dynamic obstacles shave the property that they cause the range to be shorter than z_t^{k*}, surely not longer. This likelihood probability is called p_{shot}.

3. **Errors due to failures to detect objects** are shown in figure 2.3c. Here, the obstacle at z_t^{k*} is not detected and the max range of the sensor is used to fill the gap of the not reflecting beam. This likelihood of missing the obstacle and adding the max range likelihood is called the uniform distribution p_{max}. With LiDAR this could be caused by the fact that the beam could sense a black, light-absorbing object or when measuring objects in bright light. The mobile robot in the barn uses a stereo camera and uses similarities in the two images to gain the depth image which is used to create the point cloud and a planar LiDAR scan. A failure in this system could be that similarity is not found, so no depth can be calculated in a specific region which results in no measured point of the obstacle.

4. **Random unexplained noise** is shown in figure 2.3d. There is a possibility p_{rand} over the whole range of the beam that a random error occurs that gives measurement z_t^{k*} another range. In the case of a mobile robot with a camera, this could be a light reflection on the ground.

By combining the four distributions of the different errors the combined likelihood of measuring z_t^k at z_t^{k*} is shown in figure 2.4.

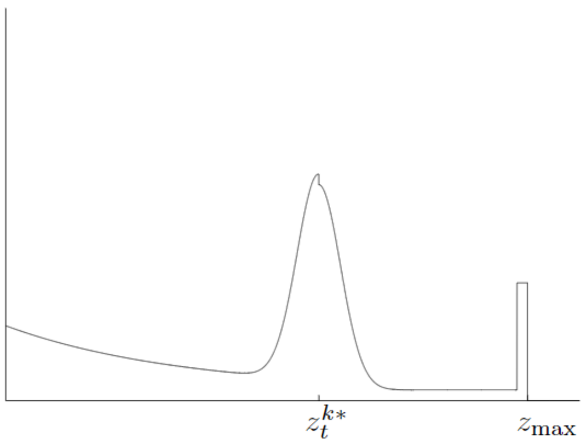


Figure 2.4: The combined likelihood of measuring z_t^{k*} along the range of the beam. Figure 6.3 in [56]

In conclusion, there are different types of noise/errors that can occur in the range of a beam. By comparing the LiDAR beams with the stereo camera-generated ones, the p_{hit} distribution could be more spread. And since there is a comparison between the two images of the stereo camera, which could fail, the p_{max} can also occur more than in LiDAR-generated beams.

2.2.2. Structure in measurements points LiDAR

The measurement points of the LiDAR consist of points in the 2D map. The first thing to do is distinguish between real measurement points and noise. This can be done using, for example, beam skipping [56]. If a certain beam for all particle filters has an error greater than the preset beam skip threshold, but the rest of the beams are within the margin, this is an indication of the quality of the beam. For example, this beam may be a dynamic obstacle that is not known on the map. By filtering out these anomalous beams, faulty measurements have less influence on determining the robot's pose. Borges *et al.* [8, 9] evaluate the noise present at the measurement points depending on the distance from the object. Shown in Figure 2.5a. Once the bias error at the different distances has been measured by testing, the new measurements can be corrected for this. Borges *et al.* filter out a bias error of up to 0.15m depending on the distance to the measured object. The residual noise due to the absence of high accuracy is shown in figure 2.5b.

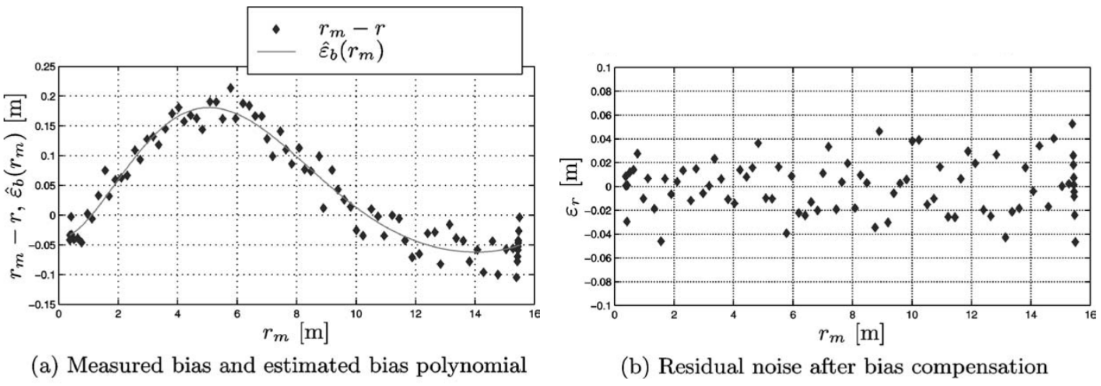


Figure 2.5: (a) measurements of an object plotted with the real distance on the horizontal axis and the differences between the measurement and the true distance on the vertical axis. (b) measurements after filtering the bias. Figure 2 in [9]

The structures found in the measurements point are lines and angles [8, 10, 9, 6, 31, 36, 37]. The lines, visible in the measurement points of the planar LiDAR, are the walls near the sensor. There are several state-of-the-art methods [31] to extract this line structure from the measurement points.

Successive Egde Following
Successive Egde Following(SEF) [49] is the most basic algorithm. All it does is connect all points in one line to create a line. However, this is not desirable for feature extraction of walls because it includes noise and dynamic objects. Besides, there are no obvious lines that could be detected as walls. It is a simple and efficient method, but sensitive to noise.

Line Tracking
Line Tracking (LT) [59, 8, 9] begins with the first two points and draws a line between them, the third point is added when the distance from the line to the third point is within threshold T_n. This continues until the distance is larger than T_{max}, then a new line is started. See figure 2.6a.

Iterative end-point fit
Iterative end-point fit (IEPF) [16] takes a subset of the measurements and defines it into two new different subsets. As long as $T_n > T_{max}$ the algorithm will split the two subsets again. When $T_n \leq T_{max}$ is satisfied the algorithm is done. See figure 2.6b. It is robust against noise and can overcome gaps in the line segments. Meanwhile, it is sensitive to outliers and computationally more expensive compared to other methods.

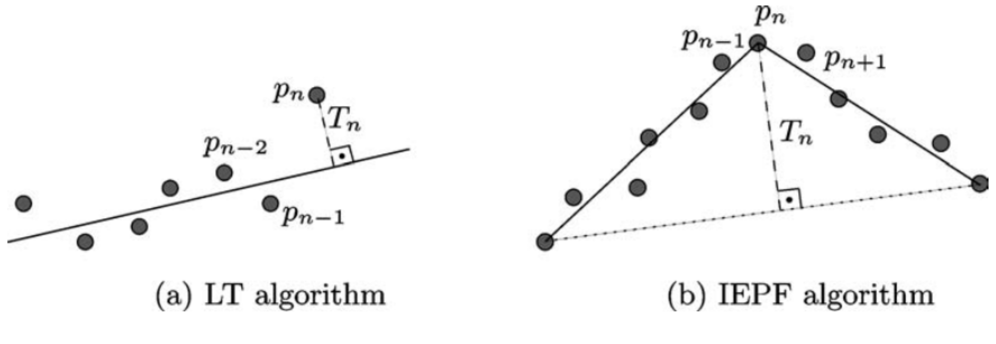


Figure 2.6: LT and IEPF algorithm for line extraction. Figure 6 in [9]

Split-And-Merge
Split-And-Merge (SAM) [41, 8, 9, 23] has the same principle as IEPF by splitting a line until $T_n \leq T_{max}$. It has an additional step of filtering out outliers and combining split lines based on similarities and the requested number of groups. It can handle complex structures since it can deal with the presence of noise and gaps. All the complexity is handled by quite some parameter that needs to be chosen.

Random Sample Consensus

Random Sample Consensus (RANSAC) [36, 37] is a robust algorithm for fitting a model in data with outliers. The features in the planar LiDAR data are the walls. These are represented by straight lines in the measurement points with a certain variation. These lines can be considered first-order linear models. The noise of the sensor are outliers in the data. RANSAC can filter these out by fitting a linear line in the data to distinguish the noise from the measurement points representing a wall. It is robust against outliers and noise but needs parameter tuning to get the best result.

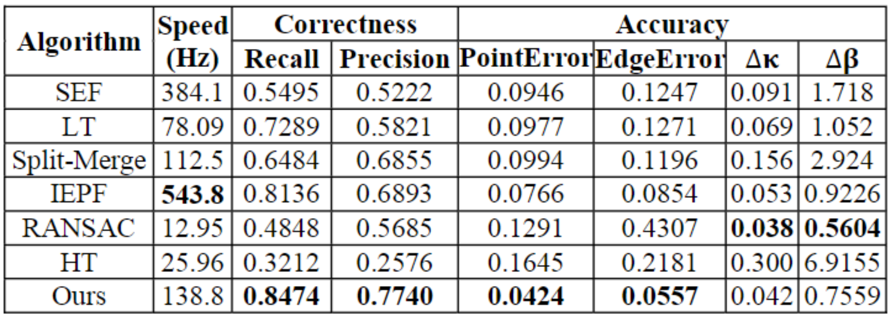
Hough Transform

Hough Transform (HT) [39] is about rotating a line through every measurement point. The right angle between the line and the origin has a distance and angle regarding the origin. This distance against the angle is plotted for every point. In the end, the most fitted line through the points is highlighted in the graph and can be used to draw a line through the points. It needs quite some computation power since it draws a rotating line through every point to find the best-fitting line.

Over-segmentation, undirected graph, and line extraction

Over-segmentation, undirected graph, and line extraction [31] is an improvement on IEPF with a combination of an undirected graph. In IEPF the threshold is automatically adjusted according to the number of points in the group. This results in a better and quicker process. The undirected graph is to find out the relations between the different groups. This helps to merge lines with similarities.

Compared to the other state-of-the-art methods the undirected graph method has the highest correction and accuracy with differences up to 200%. Adding an undirected graph increases the computation time, but it can still run in real time. In figure 2.7 a comparison is made between the different line extract methods. It is given that HT and RANSAC based on the speed, correctness, point- and edge error are not the methods to choose since they score lower than the other methods. The ‘ours’ method (undirected graph method) outperforms the other methods in correctness and accuracy error. Split-merge and IEPF are behind the ‘our’ performance and form the high middle mode in the performance of line detection algorithms.



Algorithm	Speed (Hz)	Correctness		Accuracy			
		Recall	Precision	PointError	EdgeError	Δκ	Δβ
SEF	384.1	0.5495	0.5222	0.0946	0.1247	0.091	1.718
LT	78.09	0.7289	0.5821	0.0977	0.1271	0.069	1.052
Split-Merge	112.5	0.6484	0.6855	0.0994	0.1196	0.156	2.924
IEPF	**543.8**	0.8136	0.6893	0.0766	0.0854	0.053	0.9226
RANSAC	12.95	0.4848	0.5685	0.1291	0.4307	**0.038**	**0.5604**
HT	25.96	0.3212	0.2576	0.1645	0.2181	0.300	6.9155
Ours	138.8	**0.8474**	**0.7740**	**0.0424**	**0.0557**	0.042	0.7559

Figure 2.7: ‘Ours’ is the last named method above with the undirected graph. The comparison shows that it has the highest correctness and accuracy. These higher values are a trade-off against speed. Table II in [31]

2.2.3. To sum up

The first sub-question is about lines in planar LiDAR measurements which could contribute to more robustness. In conclusion, while searching for lines in the planar LiDAR measurement points, it is important to take into account the noise that may occur in the measurement points. The bias noise can be tested for filtering out in the future. Determining the maximum variance in the remaining noise can be used to determine whether a measurement represents a wall or noise. This is especially important with the stereo camera because the variance in the measurement points of a wall increases with distance. Several line segment extraction methods have been examined and compared. Split-Merge, IEPF, and the variation on it from Li *et al.* with the graph are the best options according to the correctness, accuracy, and computation time.

2.3. Performance evaluation

The second research question pertains to testing and the evaluation of self-localization. There are different metrics to test the performance.

2.3.1. Metrics

There are different metrics of an algorithm that can be measured to determine its performance:

1. **Accuracy** is about how well the result is compared with the true value. Accuracy is hard to test in self-localization because you do not know the true location of the robot. Therefore a ground truth is needed which provides an absolute measure of accuracy [5]. By comparing the algorithm's estimated positions with the ground truth, it can be measured how much the algorithm's estimates differ from the actual locations. This allows us to make fair comparisons between different algorithms and understand their strengths and weaknesses. Without ground truth, evaluating the algorithm would rely on subjective judgments or indirect measurements, which makes it hard to know how well the algorithm actually performs. The quantitative evaluation can be done with error metrics like mean error or root mean square error, to measure how closely the estimated positions match the true locations [60]. This precise evaluation helps to understand how reliable the algorithm is and whether it consistently provides accurate estimates in different situations and environments. Additionally, ground truth helps assess the robustness of self-localization algorithms. By comparing the algorithm's performance in challenging conditions, such as when the environment is partially blocked or when there's noise in the sensor data. Ground truth data gives the ability to analyze algorithm failures more effectively and develop strategies to make the algorithm more adaptable and reliable.

   Without ground truth, alternative evaluation methods like using surrogate measurements or relying exclusively on sensor data can be used, but they have their limitations [53]. Surrogate measurements, such as using an external tracking system or trusting the accuracy of other sensors, can introduce additional errors or inaccuracies that affect the evaluation results. A method for external tracking is the use of beacons [17] where they measure the Euclidean distance to the robot. By 3 or more different beacons the 2D position of the robot can be measured. When the evaluation only depends on sensor data for evaluation it lacks a benchmark for comparison and makes it hard to know how well the algorithm is actually performing. Another method is with a landmark. The relative position of the robot can be compared with the true pose in this way. With for example a line marked on the ground of the exact route which the robot has to follow, the precision can be measured. Ground truth errors could occur due to the marking process or map distortions [25]. Wang *et al.* [62] use two known points in the environment where the robot has to drive from one to the other. The error regarding the second point can be measured.

2. **Precision** is about how constant the result is when the process is repeated. The precision in self-localization can be compared with the variation in paths when a specific trajectory is driven a couple of times. In [47] the robot drives a specific route a couple of times. After every route, an image is made from the same point of view. If the robot overlaps perfectly in the combined images it has high precision. Figure 2.8 shows the difference in the precision of the two algorithms. Note that this does not give information about the accuracy. In the comparison of [60] they use the root-mean-squared error (formula (2.1)) and the max error to evaluate the algorithms.

   In [47, 60, 27, 18, 26] the robot drives a specific route a couple of times as well, but the evaluation of the precision is done by plotting all the routes on another to visually present the variation in the different routes.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \left\| y(i) - \hat{y}(i) \right\|^2}{N}} \tag{2.1}$$

$y(i)$: Ground truth (GT) pose of the robot
$\hat{y}(i)$: Calculated pose of the robot
N : Number of poses

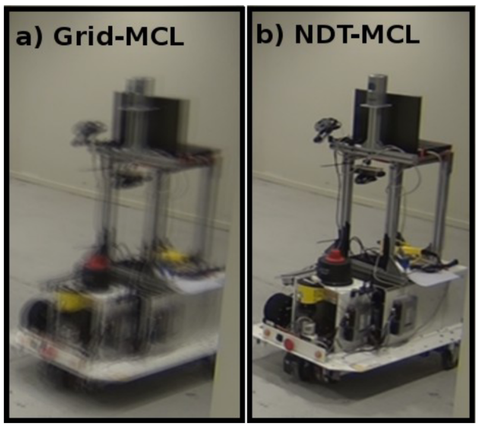


Figure 2.8: Positioning precision visualized: Fifteen images recorded from a stationary camera overlaid in positioning accuracy test. Figure 1 in [47]

3. **Latency** [37, 6] is about how fast the input data such as odometry and sensor measurements are processed in one iteration. The amount of iterations per second is also referenced as speed in Hz. It could also be how fast the algorithm converges to the right pose of the robot pose from an unknown start location if the algorithm is able to do so. The time to find the right beginning location is connected to the processing power on the system. The location can be found faster with more particles, but this requires more processing power.

4. **Qualitative analysis** can also be used to evaluate the performance of the robot. By counting the wrong positions of the robot by observing the robot during its route. Such observation is also done in the analysis in chapter 1.2 to find the most common error. Features in the environment seen by the camera can be used here. When the robot is not in the right place, the measurement points do not overlap the map properly, this error can be calculated. In this way, accuracy can also be analyzed.

To conclude, there are different ways to evaluate the performance of a self-localization algorithm. Performance in driving can be analyzed by accuracy or precision. A ground truth is necessary when evaluating the accuracy of a mobile robot. This can be done by marking known positions in the real world and letting the robot drive through them. Another simple method is analyzing the time needed for the calculations. In this way, it can be verified whether the algorithm can be used in real time.

2.3.2. Usage of a dataset

Datasets allow comparisons between different self-localization methods. By evaluating multiple methods using the same dataset, the performance can objectively be assessed and evaluated which ones work best in different situations. This helps understand the strengths and weaknesses of each approach [13]. Datasets could reflect real-world situations as well, including various lighting conditions, sensor noise, and other factors that affect the self-localization accuracy. By evaluating a system on these datasets, it can be seen how well it handles different real-world conditions. This is important for making sure the algorithm works well in practical applications [20]. Using datasets also promotes transparency and reproducibility. When the dataset is available to others, it enhances the replication of experiments and validation of results. No datasets are available for self-localization in a barn. However, there are datasets used for self-localization in self-driving cars and other studies.

RAWSEEDS

The RAWSEEDS dataset [60, 13, 20] includes in- and outdoor datasets created to evaluate the different methods for self-localization with mobile robots. It is also used for simultaneous localization and mapping (SLAM) [29] which adds an extra step to localization by mapping the unknown environment with the images of the stereo camera. The dataset contains data from LiDARs, odometry, cameras, and ground truth. The ground truth (GT) is in self-localization the most important data because it is necessary to evaluate the accuracy of the system. The precision and accuracy of the sensor, which is used to get the GT, affected the error in the GT. Therefore, evaluating using the RAWSEEDS dataset will include some inaccuracy in the error of the result. The disadvantage is that the dataset was recorded in 2009, which at the time of writing was 14 years ago. This is reflected in the quality of the images from the 0.3MP camera. In comparison, the camera of the mobile robot is 1.2MP which shows less noise.

Crowed
The environment of the mobile robot is seen as crowded. JackRabbot Dataset and Benchmark (JRDB) [34] and Robocentric Indoor Crowd Analysis (RICA) [48] are datasets with data from places with many people, but are mainly focused on detecting people and do not have a map available as input for the self-localization.

COLD
Another dataset called CoSy Localization Database (COLD) [43] is mainly focused on classifying in which space the robot is located. The authors are in the process of releasing more comprehensive sub-datasets including localization data based on the camera and LiDAR [4]. This new data set is unavailable at the time of writing.

(a) Screenshot of RAWSEEDS dataset

(b) Screenshot of JRDB with pose estimation

(c) Screenshot of RICA dataset

(d) 4 screenshots of COLD dataset

Figure 2.9: Screenshots of the different datasets

Dataset	Sensors	Environment	Number of samples	Extra info
RAWSEED (2009) [60, 13, 20]	- Binocular (B/W) 0.3MP - Mono-camera 0.3MP - LiDAR - IMU	- Indoor, hallways - Dynamic	- 15 fps camera's	-Low-resolution camera
JRDB (2019) [34]	- RGB-D stereo 0.25MP - LiDAR - IMU	- Indoor and outdoor - Static and dynamic	- 58K depth, 15 fps - 51k LiDAR, 15 fps	- Low-resolution camera - Focus on human detection
RICA (2020) [48]	- RGB-D 0.25MP - LiDAR - IMU	- Indoor - Dynamic	- 40k depth, 9 fps - 51k LiDAR, 12 fps	- Low-resolution camera - Focus on human detection
COLD (2009) [43]	- Stereo RGB 0.3MP - LiDAR - IMU	- Indoor - Static	- 16k camera, 5 fps	- Low-resolution camera - Focus on room recognition

Table 2.1: The different datasets most suitable to a crowed dynamic environment

In conclusion, datasets are vital for evaluating self-localization systems. They enable comparisons, capture real-world conditions, provide measures for performance analysis, and add transparency to the comparison due to equal input. There is no dataset that represents the barn use case. Therefore a new dataset will be created in a barn with ground truth. In this way, the different algorithms can be evaluated under the same conditions.

2.3.3. To sum up
There are different metrics to evaluate the performance of a self-localization algorithm, such as accuracy, precision, latency, and qualitative analysis. These help to compare the different self-localization methods. A dataset can be used for a fair comparison between the methods by giving each the same input, but there is no dataset that represents a barn use case. The accuracy of the self-localization method can be compared to the output of the algorithm with the ground truth. This ground truth can be measured by external sensors such as beacons, GPS, and LiDAR in the environment [17, 53]. Another method is internal data like cameras in combination with SLAM, landmarks, or onboard LiDAR with more precision than the sensor for the localization of the robot [5, 60].

2.4. Thesis contributions

The representative paper by Kang [28] shows that the accuracy of self-localization can be improved by extracting lines and corner points from the LiDAR scan and matching them with the map. In the study, the robot drives in a static environment with a nearly 360-degree LiDAR. This differs from the robot in a barn driving in a dynamic environment using pseudo-LiDAR with only a field of view in front of the robot. If due to the smaller field of view there will be much less data available per scan than in Kang's study so corners will not be visible very often. Instead, Kang uses these to find the x- and y-error in the localization. In this thesis project, the 2D error will be extracted from the line segments themselves to find the necessary localization error when no corners are available. When Kang has found an error in the localization, the position is adjusted by the average of all errors of the corners. However, this is only possible if you are confident that the lines found also represent the walls. When the robot drives around in a static environment this will be the case, but not in the dynamic environment of the barn with cows. It is chosen to adjust a part of the particle positions with the error obtained from the line algorithm, because it is not certain, in a dynamic environment, that the obtained error is correct.

The newly proposed method to improve the self-localization, called LAMCL (Line Adaptive Monte Carlo localization), will consist of 3 parts:

1. Detecting the line segments in the pseudo-LiDAR scan. This will be done by the standard split and merge algorithm. Research has proved that this works well and it is easy to implement with the given examples on the internet. However, the parameters will be tweaked because they are made for normal LiDAR which has less noise than the pseudo-LiDAR used by the robot. The implementation of M. Gallant [22] will be used, but adapted so it can be used in ROS2.

2. The second part is finding which wall best corresponds to a found line segment as well as determining the difference between that wall and line segment in terms of distance and rotation. A new method will be created for this because, in the representative paper, this is done with the corners of the measurement points, which are not available from the line detection algorithm. This will be done based on the distance and rotation of a line segment from the wall.

3. The third part is about improving the location based on the given error according to the comparison of the line segments with the walls. The representative paper does so by moving the position of the robot based on the error without considering the probability of the error. In LAMCL a fraction of the particles will be resampled based on the localization error instead of moved. The effect of this fraction will also be tested.

To test the LAMCL, a new dataset called DataCow will be created. This dataset will consist of four different routes in a barn with cows with the corresponding ground truth recorded at 7 to 9 different points per route. The dataset allows for comparison between different settings of LAMCL. Chapter 2.2.1 states that there will be noise in the measurement points. Therefore, the additional noise in the pseudo-LiDAR measurement points will be examined in order to compensate for this in the experiments.

3 Methods

In chapter 1.2, it has been explained that the main issue is the absence of particles at the location of the robot. The availability of particles in the actual position of the robot is crucial to determine the robot's pose. Therefore, particles need to be placed around the robot's actual pose if absent. One possibility, as suggested in the related work section 2.1.2, is to add random particles at each iteration with a standard deviation based on the current posterior pose of the robot. This increases the likelihood of having a particle at the robot's pose where previously there was none. Another approach is to identify the localization error and add particles at that location specifically. Kang [28] has demonstrated the feasibility of this through the detection of lines and corners in the LiDAR scan.

The proposed localization method is thus a combination of AMCL with a line detection algorithm and is called LAMCL (Line Adaptive Monte Carlo Localization). It consists of three parts that each do their own task, but depend on each other to make the whole system work. The three different parts and their relationship to each other are shown in figure 3.1 in red. The rest of chapter 3 will discuss what the different parts do and how they work.

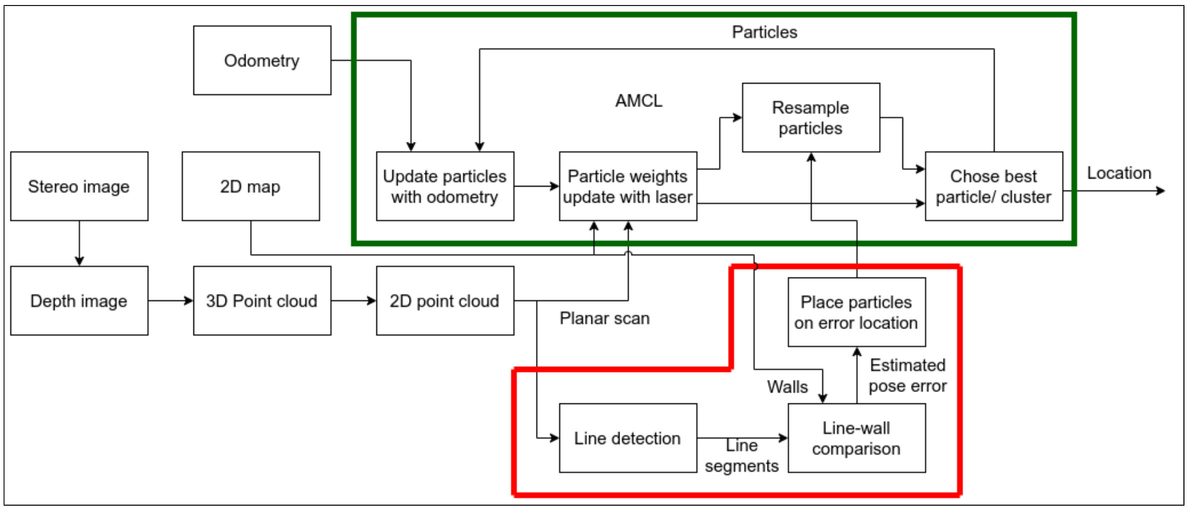


Figure 3.1: Pipeline of LAMCL with AMCL in green and the added functionality in red. AMCL has the odometry, map, and planar scan as input. The same map and planar scan are used for the newly added line detection and line-wall comparison.

3.1. Line detection

To place new particles based on the error in localization, this error needs to be measured. This can be done by comparing the robot's view of the world to that of the map. In the map, all walls are defined as line segments. By examining whether line segments can be found in the LiDAR scan, they can be compared with the walls on the map. When a line segment does not correspond in position on the map with a wall, it indicates a probable localization error. Therefore, it is desirable to detect an identified wall in the LiDAR scan as a single line segment in order to limit additional comparisons between numerous line segments and walls. A standard Split-and-Merge algorithm [7] is chosen for detecting line segments in the LiDAR scan. This algorithm starts with one line segment which is split into several line segments depending on certain conditions. Finally, matching line segments are merged and a better fit is made to the measurements to which the line segments belong. Being able to detect multiple line segments is important because multiple walls are commonly spotted in a LiDAR scan. In the comparison with the map, these must all be compared separately for the best results. However, not too many line segments should be created to minimize the comparison made later between the line segments and the walls.

3.1.1. The algorithm

For the application of the line detection algorithm, the implementation by M. Gallant [22] was used and adapted to ROS2. Algorithm 1 provides the pseudocode for this algorithm. The input is the LiDAR scan with all the data points from the environment. First, these points are filtered in L2-L11 based on range and whether they are outliers. The EndPointFit-line is used for this purpose, which is a line segment from the starting point of the measurement points to the last point, shown in figure 3.2 in the upper right corner in black. After filtering the measurements, an EndPointFit-line segment is used and split, in L13 of the algorithm. When the perpendicular distance of a measurement to the line segment is too large according to the threshold, the line segment is split at that point. The line segment is also split if the distance between two consecutive measurement points is larger than the set threshold. Once the line segments no longer meet the split criteria, the line segments which are too small are filtered out in L17 of the algorithm, based on their length and the number of points associated with each line segment. The remaining line segments are fitted to their corresponding points. Finally, the line segments that are sufficiently aligned are merged based on the Chi-squared method [14]. This method measures how much two lines differ from each other. If the difference is small enough, they are merged.

Algorithm 1 Line detection algorithm Split and Merge

```
1:  Inputs:
        2D LiDAR scan
2:  for measurement ∈ 2D LiDAR scan do
3:      if range measurement < $min\ range$ or > $max\ range$ then
4:          Filter out measurement.
5:      end if
6:      if distance to neighbors > $outlier\ dist$ then
7:          if distance to EndPointFit-line segment > $min\ split\ dist$ then
8:              Filter out measurement.
9:          end if
10:     end if
11: end for
12: while max distance to EndPointFit-line segment > $min\ split\ dist$ or max distance between mea-
    surements > $outlier\ dist$ do
13:     Split line segment into two line segments at point where split criterion is met.
14: end while
15: for line ∈ line segments do
16:     if line segment length < $min\ line\ length$ or number of points of line segment < $min\ line\ points$
    then
17:         Delete line segment.
18:     else
19:         Fit line segment with least square error using $least\ sq\ radius\ thresh$ and
    $least\ sq\ angle\ thresh$.
20:     end if
21: end for
22: if Chi-squared of two line segments < 3 then
23:     Merge line segments
24: end if
```

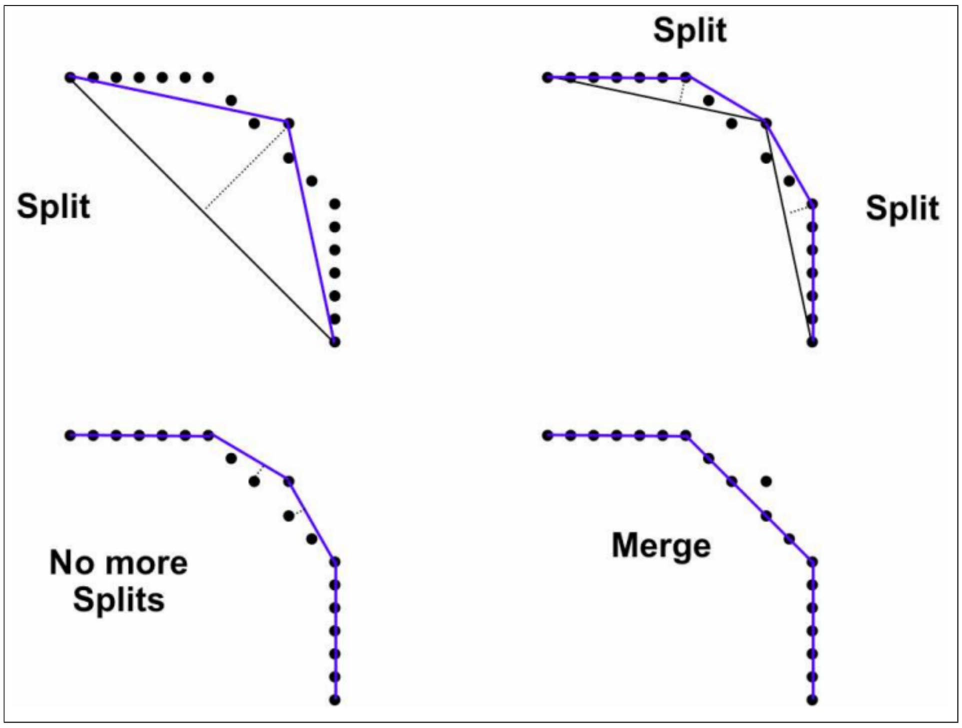


Figure 3.2: Split and Merge example with perfect data [55]. The black dots are the data point. The blue line segments are the split black EndPointFit-line

3.1.2. Parameter description

The line algorithm has several parameters that determine when a line segment is detected in the 2D LiDAR scan. The parameters ensure that it also utilizes when there is more noise than normal, for example with pseudo-LiDAR. For this purpose, there is the $range\_std\_dev$ that determines how much standard deviation is allowed in the measurement points by which it can still be seen as a line segment. The rest of the parameters are explained in table 3.1.

Parameter name	**Description**
Line algorithm	
bearing std dev	The standard deviation of bearing uncertainty in the laser scans [rad]
range std dev	The standard deviation of range uncertainty in the laser scans [m]
least sq angle thresh	Change in angle [rad] threshold to stop iterating least squares (*least sq radius thresh* must also be met)
least sq radius thresh	Change in radius [m] threshold to stop iterating least squares (*least sq angle thresh* must also be met)
max line gap	The maximum distance between two points in the same line [m]
min line length	Lines shorter than this are not published [m]
min range	Points closer than this are ignored [m]
max range	Points farther than this are ignored [m]
min split dist	When performing ”split” step of split and merge, a split between two points results when the two points are at least this far apart [m]
outlier dist	Points who are at least this distance from all their neighbours are considered outliers [m]
min line points	Lines with fewer points than this are not published

Table 3.1: Parameters description line algorithm

3.2. Line-wall comparison

The line-wall comparison algorithm is the second part of what has been added to AMCL. The inputs are the wall lines in the map and the detected line segments from the line algorithm. The output is the average localization error according to the lines relative to the walls. Comparing the lines with the walls is essential because from just the line segments no information can be extracted about whether there is a localization error. No external algorithms were used to create the line-wall algorithm and it is created from scratch.

3.2.1. The algorithm

The line-wall comparison algorithm can be seen in Algorithm 2. During initialization, the map of the environment is requested. It contains all the walls used during the comparison with the line segments. For each LiDAR scan, the algorithm considers the error according to the line segments obtained from the line algorithm. These line segments are created from the robot's perspective and therefore must first be converted to the coordinate system of the walls. A LiDAR scan can contain several line segments, therefore for each line separately the comparison is made with the walls what the possible localization error is. When a line segment is compared to a wall, it is first calculated whether the projection of the line falls within the walls, see figure 3.3.

Algorithm 2 Line-wall comparison algorithm

```
1: Initialize:
       Get map
2: Inputs:
       Line segments extracted from pseudo-LiDAR scan
3: if $frame$ of line segments is not $map\_frame$ then
4:     Transform line segments to map frame
5: end if
6: for line segment ∈ line segments do
7:     for wall ∈ map do
8:         Calculate line-wall info
9:         Check |line segment - wall| < |line segment - current best wall|
10:        Check ∠(line segment, wall) < ∠(line segment, current best wall)
11:        wall-best wall difference = |line segment - wall| - |line segment - current best wall|
12:        Δ∠ = ∠(line segment, current best wall) - ∠(line segment, wall)
13:        Check if (L9 is $false$) and ( L12 > $\frac{angle\ end - angle\ start}{distance\ end - distance\ start}$ * | L11|) and (| L11| < $distance\ end$)
14:        Check if (L10 is $false$) and ( L11 > $\frac{distance\ end - distance\ start}{angle\ end - angle\ start}$ * | L12|) and (| L12| < $angle\ end$)
15:        if (9 and 10) or (10 and 13) or (9 and 14) is $true$ then
16:            Wall is new best wall
17:        end if
18:    end for
19:    Calculate additional displacement due to rotation of line segment
20:    Add line error to separated x, y and theta history queue
21: end for
22: if average line segment error of x, y, or theta > $max\_allowed\_difference$ then
23:    Publish average line segment error
24: end if
```

Figure 3.3 shows two different line situations (the 3rd is when both projections are outside the wall, but it is the same as when one projection is outside the wall). For line 1, both projections are inside the wall. In this situation, the angle relative to the wall is calculated and the center of the line is used for the distance from the line to the wall. For line number 2, one of the projections lies outside the wall. In this situation, the angle relative to the wall is calculated the same. However, the distance is not calculated from the center, but the end of the line furthest from the wall is used for the distance after rotating with the angle. This ensures that the line is completely inside the wall with rotation and displacement. From the displacement of the line segment, the Euclidean distance is used to check if the current wall being compared is better than the currently best wall along with the angle. When the distance and angle are both smaller it is clear that the current wall is better.

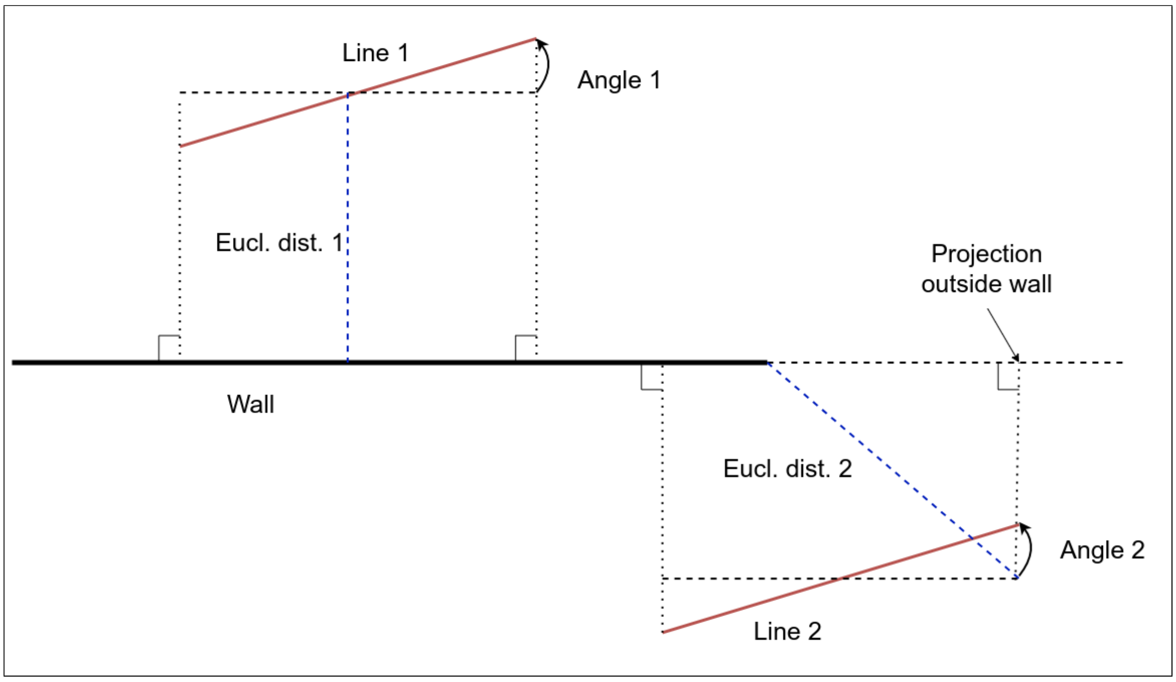


Figure 3.3: Two of the situations of the projection of a line segment with a wall. In big black the wall, red is the line segments, the dotted line is the projection, the dashed black line is the compared orientation of the wall, and the dashed blue is the distance of the line segment to the wall.

It can also happen that the distance gets better, only the angle gets worse (bigger), or vice versa. In these cases, there must be a minimal improvement in one when the other gets worse. This relationship is included in the enough-improvement function, plotted in figure 3.4. This is checked in Algorithm 2 L13 and L14.

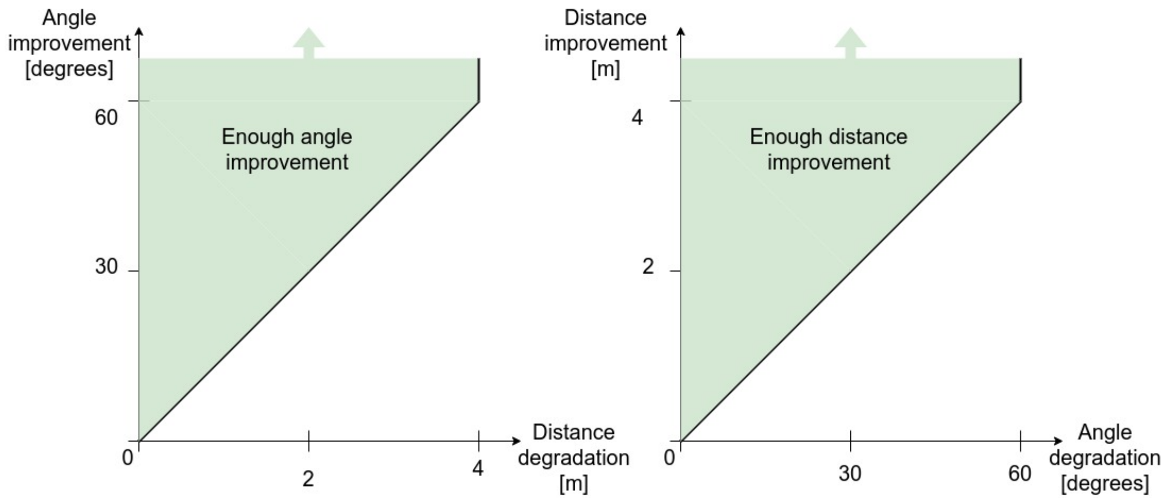
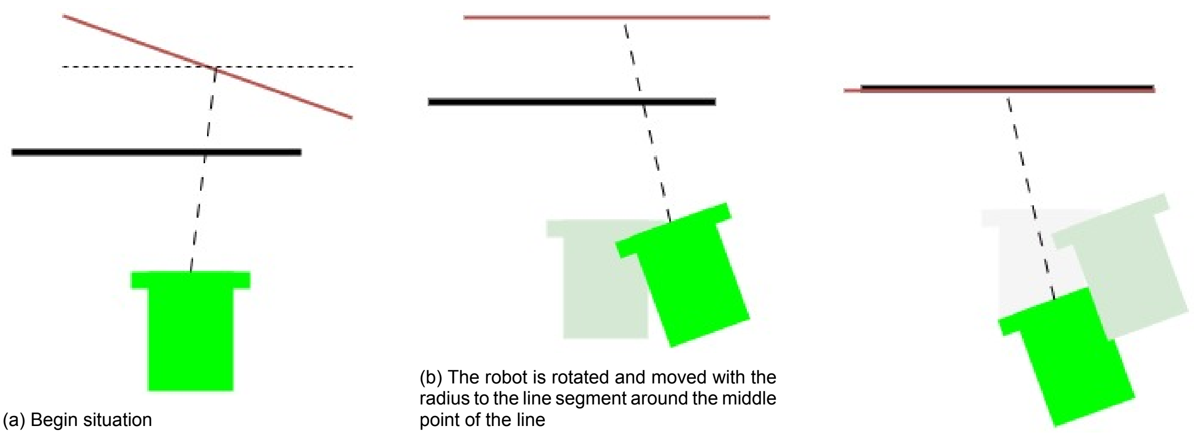


Figure 3.4: The functions used to check if there is enough improvement in algorithm 2 line 13 and 14. For example, if the distance gets worse by 2 meters, there needs to be a minimum improvement of 30 degrees in the angle between the line segment and the wall.

When the best matching wall is chosen for a line segment, the displacement, and rotation of the line segment relative to the wall are known. However, the robot is at a distance from that line segment, and while rotating the line segment relative to the wall, there is an additional displacement that the robot must undergo to match the line segment with the wall. This is L19 of algorithm 2 and is visualized in figure 3.5.


(a) Begin situation

(b) The robot is rotated and moved with the radius to the line segment around the middle point of the line

(c) The robot is moved with the displacement of the line segment to the compared wall

Figure 3.5: Calculation process of the localization error. The black line represents the wall, the red line is the detected line, and the green the robot. Note: The wall itself is not moved in the figures

The calculated displacement per line segment is then added to the history queue on L20 of algorithm 2. When the average displacement of x, y, or heading exceeds the set threshold, the average estimated pose error is published.

3.2.2. Parameters description

For the line-wall comparison algorithm, there are several parameters which are explained in table 3.2. As mentioned before you can see that the x, y, and angle all have their own threshold. The start and end values of the distance and angle of the plots in figure 3.4 can be adjusted with the distance and angle start and end values.

Parameter name	**Discription**
Line-wall compare	
wall max history	The maximum amount of lines used to calculate the mean position error
max allowed angle difference	Only poses with more rotational error then this are published for correction [degrees]
max allowed pose difference x	Only poses with more error in x-axis then this are published for correction [m]
max allowed pose difference y	Only poses with more error in y-axis then this are published for correction [m]
distance start	Start distance that defines where the betterwall-function begins [m]
distance end	End distance that defines where the betterwall-function ends [m]
angle start	Start angle that defines where the betterwall-function starts [degrees]
angle end	End angle that defines where the betterwall-function ends [degrees]

Table 3.2: Parameters description line-wall comparison

3.3. Localization modification

Now, the possible localization error of the robot is known. As indicated in earlier sections, the correct pose can be restored by regaining particles at the robot's pose. It has been chosen to use the localization error to place particles at the location of the error as a replacement for part of the current particle cloud. This will allow AMCL to view the LiDAR scans from these poses and evaluate whether those new particles are a better representation of the robot pose.

3.3.1. The algorithm

The particle place algorithm is shown in Algorithm 3. The input is the localization error from the line-wall compare algorithm. Only when enough time has passed between the last modification and the current time, this localization error is used. When sufficient time has passed, a new particle cloud is filled with samples from the error location with a standard distribution. The new particle cloud is filled until the preset percentage of the total particle cloud is filled. This percentage is the parameter $fraction\ resample\ from\ line\ error$. The weight of each added new particle is set to 1 and divided by the preset division value. The rest of the new particle cloud is resampled by the standard AMCL method by resampling from the current particle cloud.

Algorithm 3 Localization modification algorithm

```
1: Inputs:
       Estimated localization error
2: if (current time - time last modification) > $min\ time\ between\ lamcl\ resamples$ then
3:     Get pose robot
4:     while new particle cloud size < (particle cloud size * $fraction\ resample\ from\ line\ error$) do
5:         Sample new particle from normal distribution with $stddev$... around (pose robot - localization
       error)
6:         New particle weight = $1/division\ weight\ added\ particles$
7:         Add new particle to the new particle cloud
8:     end while
9:     Tell AMCL to resample with partly filled new particle cloud
10: end if
```

3.3.2. Parameters description

Table 3.3 shows all the key parameters related to the placement of particles around the error location. The minimum time between two placements of particles at the error location is chosen to give AMCL time to consider the newly placed particles when evaluating the robot's location. The standard deviation of the standard distribution of the particles is because there is an inaccuracy in identifying the lines in the pseudo-LiDAR. It is not certain that the error is also exactly the localization error. The weights of the particles from the line algorithm are not the same as the sampled ones from the previous particle cloud. This is related to the fact that it is not sure that these particles are correct. With the same weights, the position could jump to a false location due to a dynamic obstacle being mistaken as a wall. With lowering the weights, the AMCL has more time to check that this is not the case. The most important addition is that not all new particles are sampled from the localization error. The percentage ensures that not all odometry information from the past is forgotten. So, the majority of the new particles are sampled from the current position of the robot.

Parameter name	Description
Place particles on error pose	
min time between lamcl resamples	Minimum time between resamples using the estimated localization error [sec]
std dev pose x	Standard deviation of the placed particles in the x-axis
std dev pose y	Standard deviation of the placed particles in the y-axis
std dev orientation	Standard deviation of rotation of the placed particles
fraction resample from line error	Percentage of particle cloud which comes from line error
division weight added particles	Weight of added particles from line error will be divided by this

Table 3.3: Parameters description localization modification

4 Experiments

The research questions revolve around the accuracy, precision, and robustness of the algorithm. To test this, an experiment was conducted for each aspect. For accuracy, the same experiment as described in Wang's paper [62] is used, where the robot drives from location A to B, and the pose error at location B is measured. For precision, the robot drives the same route multiple times, and by examining the overlap of these routes, insights can be gained regarding the difference in precision between LAMCL and AMCL. Robustness is assessed by observing if and how quickly the self-localization could recover from a localization error. This is achieved by allowing the robot to drive towards a wall from a fixed distance but indicating a deviation in distance. Finally, a test is conducted to evaluate how well the algorithm performed in a barn environment with cows. Ground Truth data is recorded in this test at specific points to measure the accuracy. In most experiments, the middle of the robot's front is assumed as the robot's location since it represents the point that should align with the route when the robot drives autonomously. Only in the experiment about the precision the middle point of the robot was used. For the precision, robustness, and barn tests, rosbags were recorded and played back to test LAMCL with different parameter sets. Further details can be found in chapter 4.1 and 4.2. By conducting the experiments, it is possible to say something about the performance of the system and compare the different configurations afterward.

4.1. Barn dataset: DataCow

In chapter 2.3.2, the importance of a dataset with ground truth is explained when evaluating self-localization. There are several publicly available datasets such as RAWREEDS [60, 13, 20], JackRabbot Dataset and Benchmark (JRDB) [34], Robocentric Indoor Crowd Analysis (RICA) [48] and CoSy Localization Database (COLD) [43]. The particular barn environment, setup, and sensory equipment exhibit no similar properties as the ones used in these standardized datasets. Therefore a new dataset was created to fulfill the properties of the barn robots' situation: DataCow.

4.1.1. Setup data recording

DataCow is recorded in a barn with cows. The layout of the barn can be seen in appendix A. When the robot drives a route, the localization algorithm affects how this route is taken. There are external methods to record the GT of the robot, such as GPS, a motion capture system [51] and a total station [52]. However, these are not suitable for the barn environment since it is indoor and the space is too large to measure with an external sensor. Therefore, it was decided to manually guide the robot along a route. In addition, this allows the robot to be temporarily paused at certain landmarks in the barn to manually measure the pose for the ground truth measured to the nearest 0.005m. These landmarks mainly consist of corners of the walls or measured distances from a wall corner. In this way, they can be later drawn on the map to compare them with the calculated pose of self-localization methods with varying parameters. See chapter 4.2 for the different parameters.

4.1.2. Routes

DataCow contains four different routes in the barn and has the input for the self-localization. These are the LiDAR scan and the Translation between Frames (TF). The TF gives the relation between the origin of the map and the robot in 3 DOFs (x, y, theta). The TF contains the relationship between map-odom and odom-base_link. See figure 4.1. Odom to base_link is determined by the robot's displacement based on wheel movements and the IMU (Inertial Measurement Unit) in the robot. The map to odom relationship is initially fixed, but during motion, the localization algorithm can adjust this relationship when the particles, according to self-localization, are not in the correct positions. This is the most important output of the localization algorithm. In DataCow, only the initial position and the odom-base_link relationship are stored in the TF. The map-odom relationship is eventually added by a localization algorithm when tested.

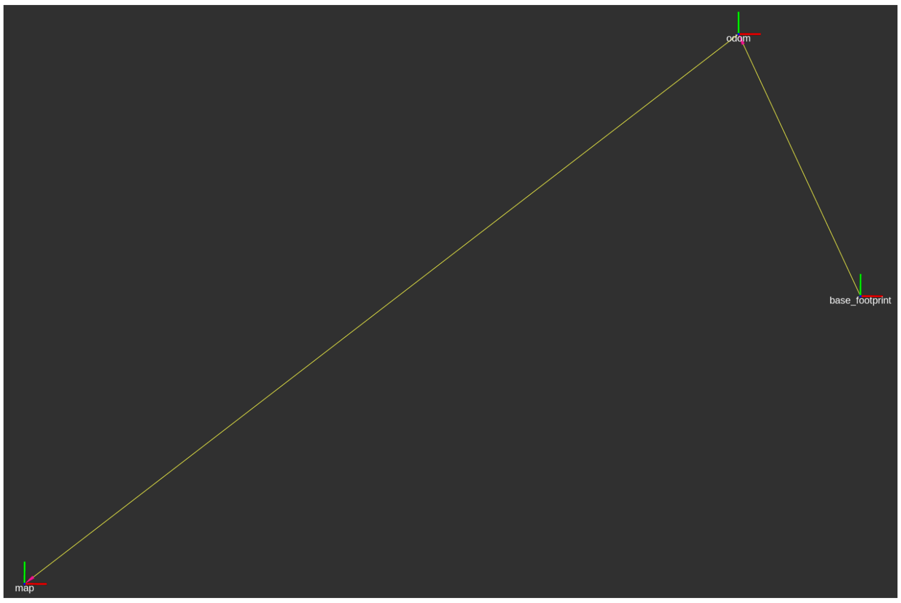
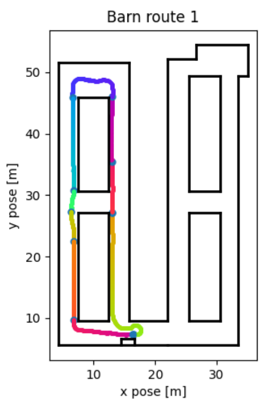
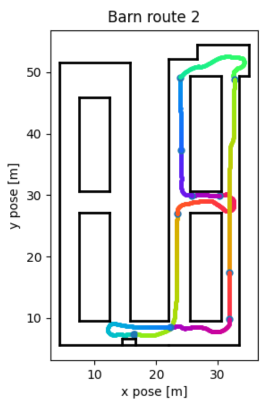


Figure 4.1: TF tree of the robot with map, odom, and baselink

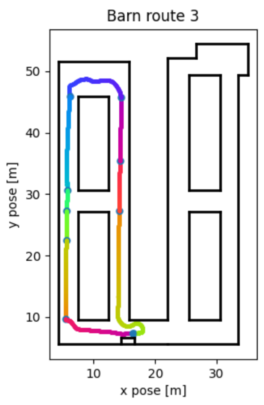
The DataCow dataset consists of 4 different routes, with two routes on each side of the barn. Route 4, in particular, exhibits a high level of dynamism, as it captures the cows returning from the pasture and then standing at the feeding racks to eat. Figure 4.2 displays the routes. These routes are visualized with the default 0 parameter set given in chapter 4.2. All the routes driven in the experiments are plotted with changing colors over time to visualize the progression when the robot transitions or jumps between locations. The blue dots indicating the locations where the robot stopped according to the self-localization.



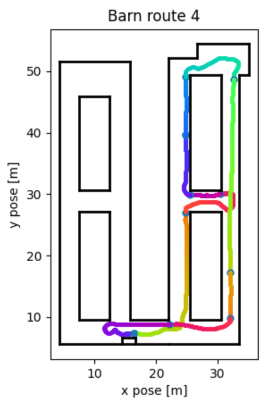
(a) Barn route 1



(b) Barn route 2



(c) Barn route 3



(d) Barn route 4

Figure 4.2: Routes in the barn. The dots represent the position where the robot stood still. The colors of the routes are for indication in order to better follow the trajectory when the localization pose is staggered.

4.2. Parameters

As discussed in Chapter 3, there are three different components in LAMCL, each of which has its own parameters. Chapter 3 provides overviews of the different parameters and their meanings. Because of the duration of 3 hours per parameter set, it was not possible to test all possible combinations. Therefore, it was chosen to work with default values derived from a qualitative analysis of the 3 subsystems. Quantitative was not possible given the absence of a test set and GT for the 3 components of LAMCL separately and the available time which made it not possible to create one. From the default values it was checked how the parameters affect the whole system by adjusting them one by one. When there is no parameter value in the column of *"value (parameter set)"* it means that the default value has been used for all parameter sets. The complete overview of the parameter sets can be found in appendix B.

4.2.1. Line detection

As described in chapter 3.1, the standard algorithm Split-and-Merge was chosen to detect line segments in the LiDAR scan. This was originally used to detect line segments in a real LiDAR scan rather than the pseudo-LiDAR scan which the robot in the barn has. The difference between LiDAR and pseudo-LiDAR is that more noise is present in pseudo-LiDAR, especially as the distance from the detected object increases. So for the line detection to work, a qualitative analysis was done on how it works. The original values [22] that work for a normal LiDAR were also tested as parameter set 10. However, no line segments were detected in the pseudo-LiDAR measurements. When testing and calculating the default deviation a $range\ std\ dev$ of 1 was chosen as default with $least\ sq\ angle\ thresh$ and $least\ sq\ radius\ thresh$ of 0.1. At greater distances, the measurement points in the scan are more spaced apart. To detect lines at large distances as well, $the\ min\ split\ dist$, $max\ line\ gap$ and

$outlier\ dist$ were also changed. The final default values became parameter set 0. The variations of the line algorithm parameter values are shown in table 4.1.

Parameter name	Default value	Value (parameter set)
Line algorithm		
bearing std dev	0,001	
range std dev	1	0.02 (10)
least sq angle thresh	0,1	0.0001 (10,11)
least sq radius thresh	0,1	0.0001 (10,11)
max line gap	1	0.4 (10)
min line length	0,5	
min range	0,1	0.4 (10)
max range	6,4	
min split dist	0,5	0.05 (10), 0.3, (11)
outlier dist	0,6	0.05 (10)
min line points	6	9 (10)

Table 4.1: Line algorithm parameter values with experiments. See appendix B for an overview of the parameters.

4.2.2. Line-wall comparison

As discussed in Section 3.2, the line-wall comparison algorithm is self-written with the necessary parameters. The localization error is calculated over an average of a number of lines including those in the past. The number of lines included in the average depends on the $wall\ max\ history$. The default value chosen for this is 80 lines. This is based on the fact that there are about 2 to 4 line segments per LiDAR scan and the algorithm can run at 10 fps while driving. Thus, about 2 to 4 seconds of past view are taken. To check the influence of the magnitude of the past, 5, 10, 20, and 150 line segments in the history are also evaluated during testing.

The values that the localization error must satisfy before adding additional particles are chosen at 0.1 meter position difference or/and a rotation of 10 degrees. When there is no pose error it is not necessary to add extra particles every time because the position where the robot might be becomes more uncertain due to the standard distribution (section 4.2.3). The end distance and angle are set such that there must be at least a 60 degree rotation improvement when a wall is 4 meters away from the current best wall when comparing the line with all walls. Likewise, there must be at least a 4 meter improvement when the rotation of the wall is 60 degrees worse compared to the current best walls during the comparison with all walls. The line-wall comparison default values can be seen in table 4.2.

Parameter name	Default value	Value (parameter set)
Line-wall compare		
wall max history	80	20 (40), 10 (41), 150 (42), 5 (43)
max allowed angle difference	10	
max allowed pose difference x	0,1	
max allowed pose difference y	0,1	
distance start	0	
distance end	4	
angle start	0	
angle end	60	

Table 4.2: Line-wall compare algorithm parameter values experiments. See appendix B for an overview of the parameters.

4.2.3. Localization modification

The localization modification indicates how to respond if there is a localization error according to the comparison of the line segments with the walls. This error is measured at 10 fps, however, it is not necessary to modify the particles at every measured error. Therefore, there is the $min\ time\ between\ lamcl\ resamples$ that indicates how much time should pass before it resamples again based on the line-wall localization error. This is set to 2 seconds by default. This gives the algorithm time to include the newly placed particles in a number of cycles to calculate the possibility of the robot pose. The localization error is an average calculated over the past period depending on the $wall\ max\ history$ as discussed in the previous section. A longer period is not desirable considering there are all localization errors of around 0m in the history in the beginning. So it takes a while for an error of, let's say, 0.6m to bring the average up to 0.6m. However, an error localization is already issued at 0.1m which means that when a longer $min\ time\ between\ lamcl\ resamples$ it will first issue a localization error of 0.1m and only much later issue the correct one of 0.6m. As a result, the algorithm will not function optimally.

The default deviation of the added particles is by default set to 0.08. This means that there is a <2% probability that particles will be placed with a Gaussian distribution with a radius >0.16m around the corrected error location according to the line algorithm. This is based on the default distribution used when the robot is at the charger and repositioning itself there. In addition, this also helps with the uncertainty there is in the detected line segments given the variation in the measurement points.

When there is a localization error following the line algorithm, particles are placed at that location. This still means that the same number of particles remain. A fraction of the number of particles in the particle cloud is placed at the error location during resampling. This percentage is defined by $fraction\ resample\ from\ line\ error$ which defaults to 0.1. This means that 10% of the total particle cloud during resampling does not come from the resample but is pre-filled by the error location from the line algorithm if there is an error. The default 0.1 was chosen because the adjusted particles are an addition and basic AMCL should be able to localize the pose properly. With a larger value, the influence of the localization correction according to the line algorithm increases. To test this, different values are being used in the experiments to evaluate the influence.

The detected line segments are seen as lines that represent walls in the measurement points. However, this could also be a cow. In that case, LAMCL will give a localization error, even though the localization of the robot could be good because a cow cannot be in the same position as a wall. This means that there will be particles at the location where the robot is not. To minimize the consequences of this error, the weights of the particles are divided by a default value of 8. This value is chosen based on a qualitative analysis. The effect is evaluated by also taking a value of 1, 4, and 16 in the experiments. An overview of the localization modification default parameters can be seen in table 4.3. The overview of all the parameters can be seen in appendix B.

Parameter name	**Default value**	**Value (parameter set)**
Place particles on error pose		
min time between lamcl resamples	2	
std dev pose x	0,08	0.006 (50), 0.0006 (51), 0.2 (52)
std dev pose y	0,08	0.006 (50), 0.0006 (51), 0.2 (52)
std dev orientation	0,03	0.002 (50), 0.0002 (51), 0.2 (52)
fraction resample from line error	0,1	0.98 (20), 0.5 (21), 0.75 (22), 0.25 (23)
division weight added particles	8	1 (30), 4 (31), 16 (32)

Table 4.3: Localization modification parameter values experiments. See appendix B for an overview of the parameters.

4.3. Recovery ability

The research question includes the improvement of the robustness of the self-localization. Robustness is defined according to the Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, as *"The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions"*. So, this section examines if and how quickly the self-localization can perform a recovery when the robot is in the wrong position as in [2]. In this experiment only driving to a wall is used. AMCL already has the ability to recover when the robot gets stuck or collides with a wall by expanding its particle cloud to regain the correct position. However, in this experiment, the focus is on whether the robot can perform the recovery before colliding with the wall and how quickly the recovery takes place once the robot starts moving.

4.3.1. Experiment setup

The robot was placed 6.2 meters in front of the wall after which its location was adjusted in the software so that the measurements of the farthest wall matched the map. See figure 4.3. Then the current pose according to the software was saved for start reference. This reference was used to have a GT of the distance from the wall drove forwards with a constant speed of 0.2 m/s. This is the black line in the figure 4.5 and 4.6. The position according to self-localization is stored 10 times per second. Each experiment was done 3 times to eliminate accidental anomalous one-time behavior, with all parameters and all distances with the line algorithm. This change of distances is 0, 0.25, 0.5, 1.0, -0.5, and -1.0 meters. The experiments without the line algorithm are done only 3 times with all distances and without the change of parameters since the parameters only influence the line algorithm.

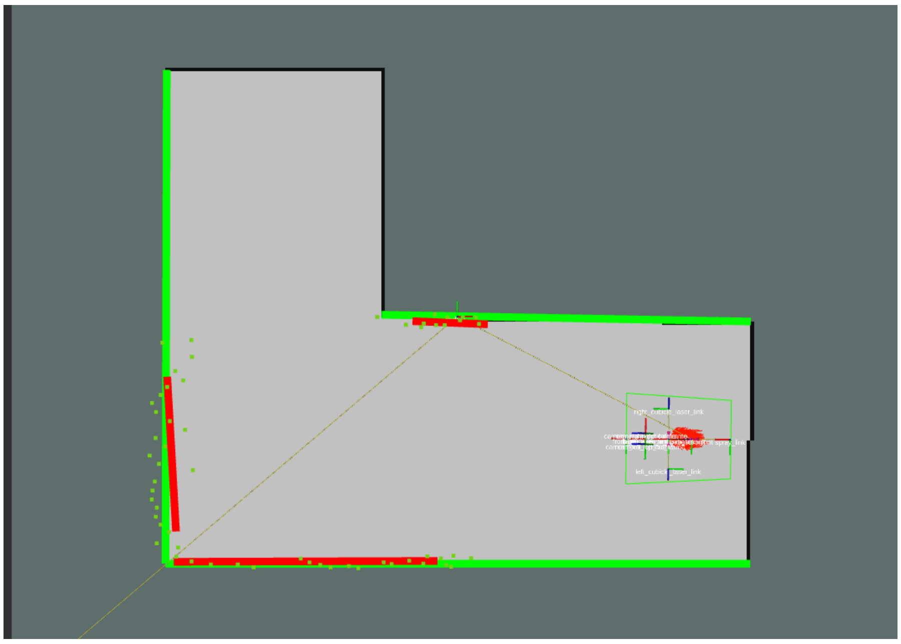


Figure 4.3: Top-view of the start situation of recovery experiment. The green box on the right is the robot. The green dots represent the measurements. The red lines are the found line segments in the measurements with their compared walls in green.

4.3.2. Measurement bias error compensation

Chapter 2.2.2 described how the different errors affect the reliability of the measurements. To see how this relates to the pseudo-LiDAR scans of the stereo camera an extra experiment was done. The robot was placed at 1, 2, 3, 4, and 5 meters from the wall after which the distance to the wall according to the LiDAR scan was measured. This trend is shown in Figure 4.4 with the true distance on the x-axis and the difference from the measurement on the y-axis. The blue points are the measured distances and the green points are the averages per distance. The red line is the second-degree polynomial of the averages. When this polynomial is extrapolated to 5.6 meters, the distance at which the robot starts the robustness test, it can be seen that there is a bias error of -0.37m in the measurement.

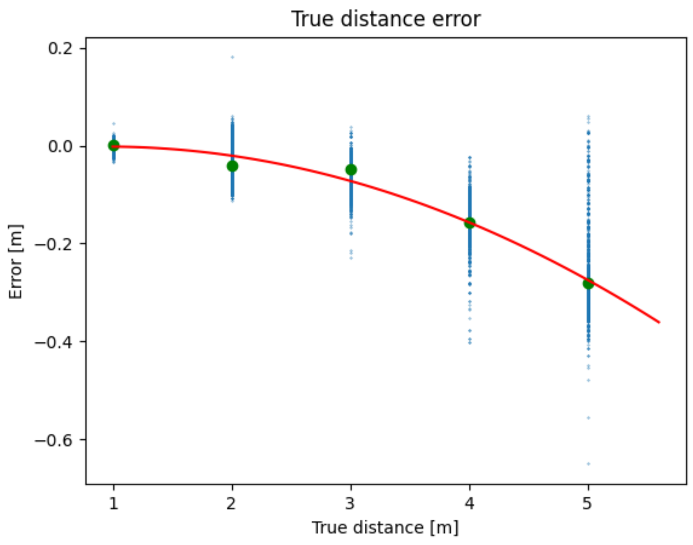
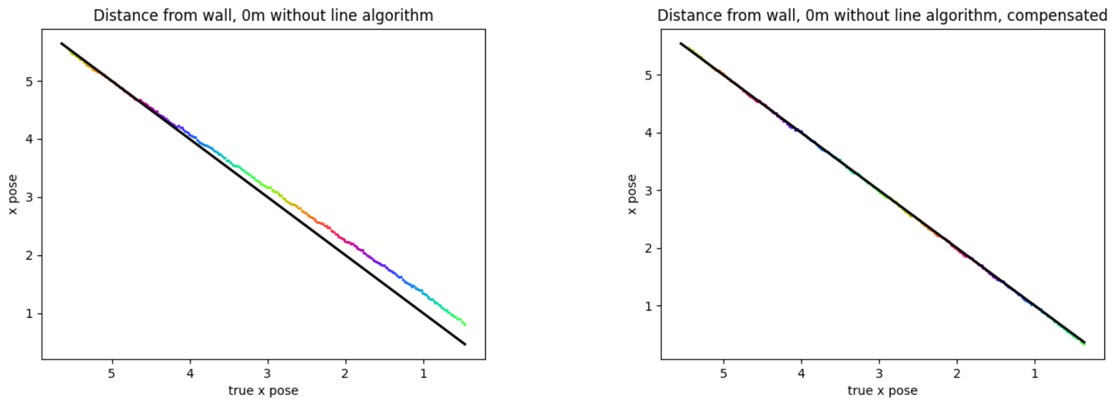


Figure 4.4: GT test of the pseudo-LiDAR measurements. In blue are the pseudo-LiDAR measurements, in green are the averages, and in red are the polynomial of the averages.



(a) Distance to the wall without compensation

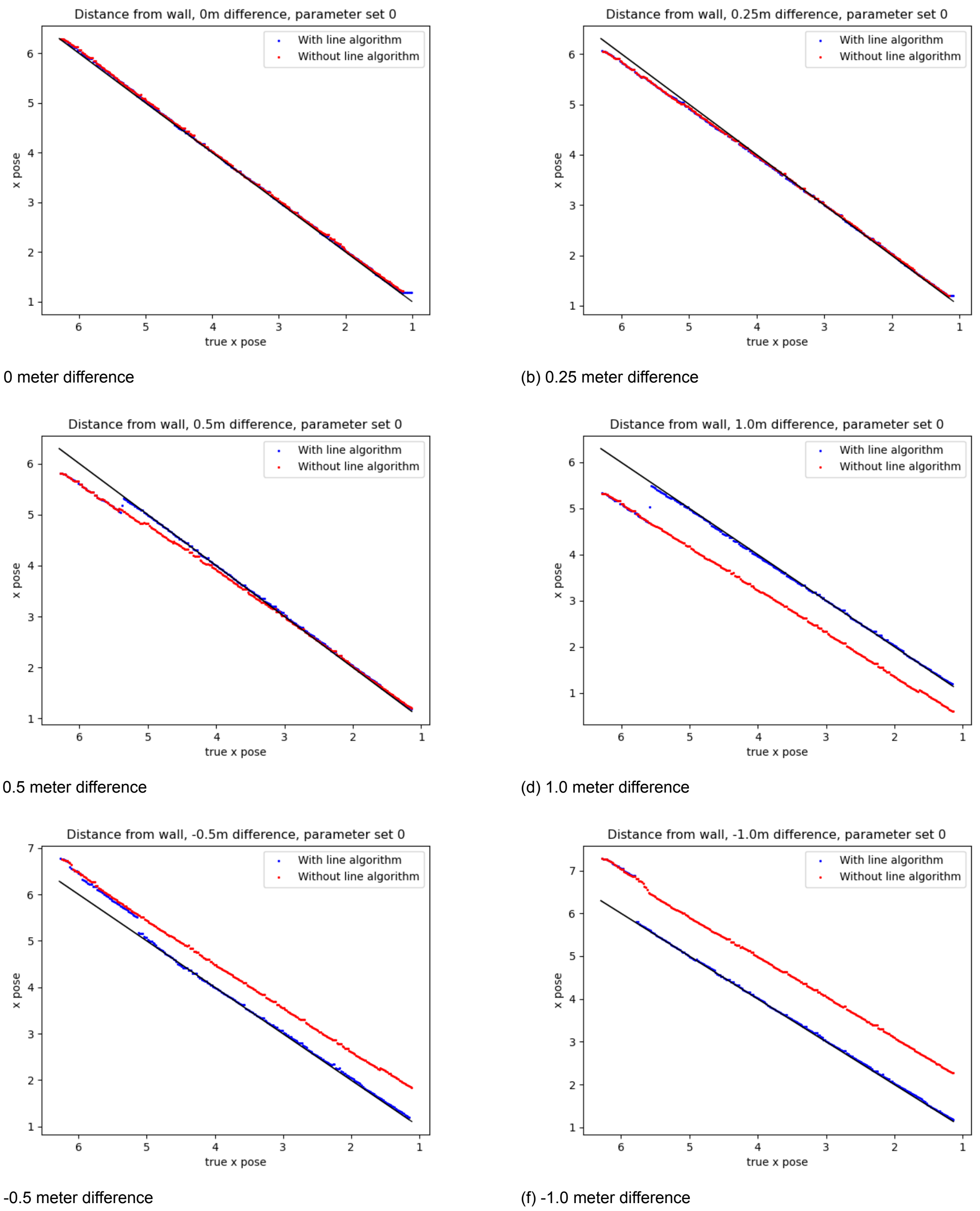
(b) Distance to the wall with compensation

Figure 4.5: Difference when compensated for bias error. In black the true 'trajectory' of the robot

Figure 4.5 shows the difference between distances from the wall. In 4.5a, AMCL has set the robot at the correct distance in front of the wall. However because the LiDAR scan was set parallel to the wall at the start and the measurements, at that distance, 5.6 meters, it has a bias of -0.37m which can be seen at the end. At the end of the test, 1 meter before the wall, there is no bias according to the GT LiDAR measurements test in figure 4.4. Therefore, a difference in position can be seen at the end of driving of 0.38m in figure 4.5a. When the distance is compensated with the polynomial of the GT LiDAR measurements test, the distance of the robot goes identical to the real distance to the wall, shown in figure 4.5b. This compensation was used for the rest of the recovery test in all experiments.

4.3.3. Results

Figure 4.6 shows the difference between with or without the addition of extra particles from the line algorithm. If the error is small enough, 0.25 or 0.5 meters, AMCL can solve the localization error itself in some cases (see Table C.1). The particles that are closest to the correct position get the highest weights in AMCL. When resampling, more particles are placed around this position so that the particles slowly converge to the correct position. When the distance becomes larger, this is no longer corrected and the robot remains at the wrong location. The line algorithm shows that during testing with the default parameter set 0, the robot always converged to the correct position. Even with an error of 1.5 and -1.5 meters, LAMCL managed to perform a recovery.



(a) 0 meter difference

(b) 0.25 meter difference

(c) 0.5 meter difference

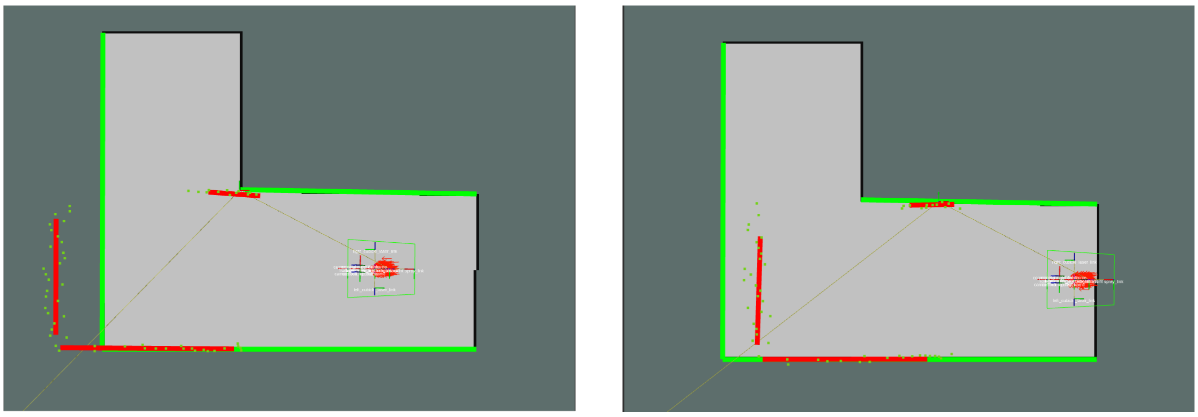
(d) 1.0 meter difference

(e) -0.5 meter difference

(f) -1.0 meter difference

Figure 4.6: Difference between with and without line algorithm of one test

Figure 4.6 illustrates a distinction between positive and negative errors in the localization, which can be attributed to the layout of the test environment shown in figure 4.7. In the scenario of positive error (figure 4.7a), three lines are utilized to assess the localization error. However, due to two of these lines being projected within the corresponding wall, the localization error in that direction remains undetectable. Consequently, only one line contributes to recording the history of negative errors per LiDAR scan, in contrast to the three lines in the positive situation. This discrepancy leads to a faster rate of change in the mean of the positive error compared to the negative localization error.



(a) 1.0 meter difference (b) -1.0 meter difference

Figure 4.7: Difference in negative and positive localization error. The green box is the robot, and the red line segments are the detected lines in the measurements, which are the green dots. The green lines are the wall to which the line segments are compared to. In figure A all three line segments are taken into account by calculating the pose error. In figure B only 1 line segment is taken into account since the others are in the wall

The different parameter sets are divided into 5 changes compared to the default values of parameter set 0: change in line detection, change in part of particles from line detection error, change in weight of added particles, change in line error history, and change in the standard deviation of added particles. The next sub-sections will discuss the influence of the different changes on the recovery ability.

Change in line detection

To test the effect of the line segments on the performance of the recovery ability, four different situations were considered. The first situation is the baseline where the line algorithm is turned off. The second situation called 'original' in figure 4.8 (parameter set 10), is with the original parameters of the line algorithm as used online by other researchers with a LiDAR scan [22]. The situation 'some' (parameter set 11) has to do with the increase in the default maximum allowed deviation of the measurement points in the line segments. The name 'some' is a reference to the amount of line segments detected by the line algorithm. The 'default' (parameter set 0) setting is the starting point to experiment with the other parameter sets. The difference from the 'original' set is that here the parameters have been changed to filter out fewer outliers and given a wider area to merge lines back together.

The 'original' settings are the parameters used with a real LiDAR. The robot in the barn uses pseudo-LiDAR which has more noise in the measurement points so almost no line segments can be found with the original parameters. This is also reflected in the results which on average are almost the same as the test without the line algorithm. When the default deviation is changed to the 'some' setting it can be seen that indeed some line segments are detected and the recovery ability also improves. With further adjustment of the parameters in the line algorithm, the recovery ability improves even more with for all tests a lower driven distance before the recovery took place.

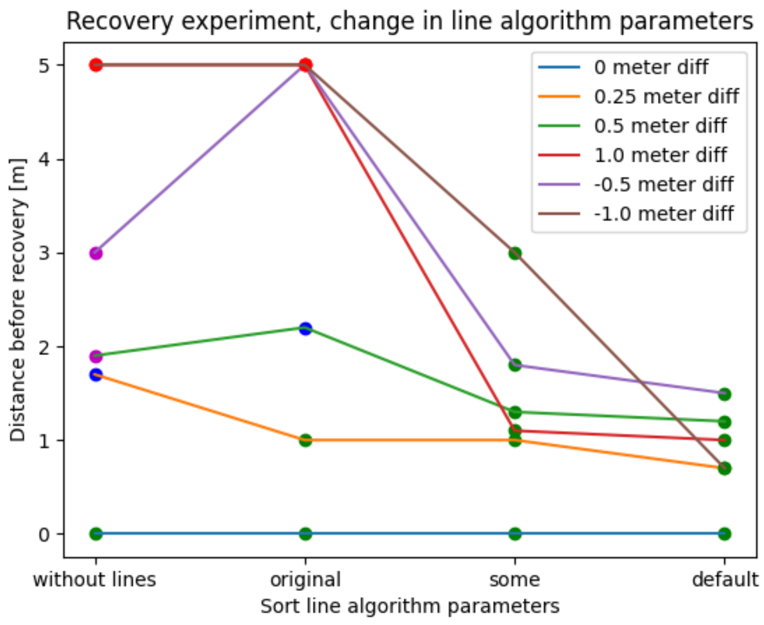


Figure 4.8: The number of meters the robot drove before the error is <0.1m. The dots are the measurement points. Each test is performed three times, and the data points represent the average distance from the tests that were passed. The green, blue, magenta and red dots indicate respectively 3, 2, 1, or 0 successful recoveries. For these tests, the distance the robot drove before recovery is set to 5 meters when no recovery succeeded.

Change in part of particles from line detection error

Parameter sets 20, 21, 22, and 23 deal with the difference in what percentage of the line detection error should come during resampling. This is shown in figure 4.9. The plot shows that the higher the percentage of resample coming from the line algorithm is, the more frequently the self-localization failed to perform a recovery. It can also be seen that with the default setting of 10%, the success rate is the highest.

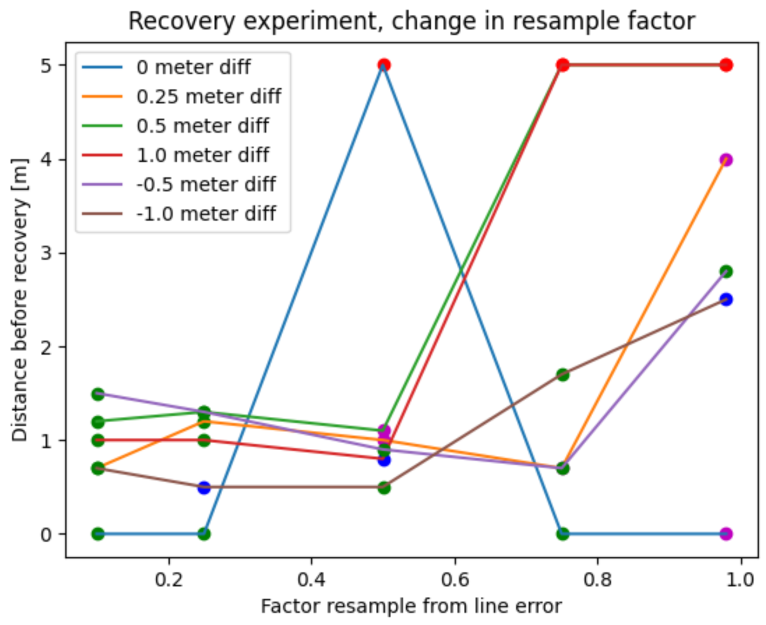
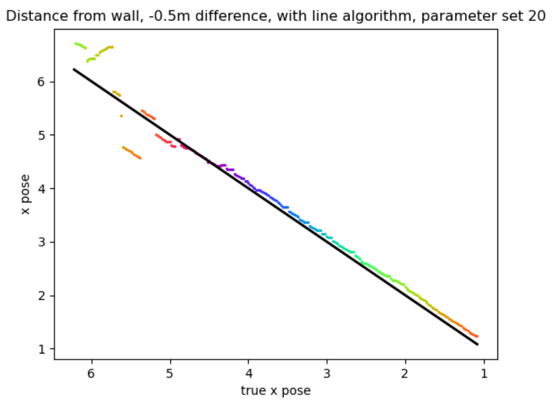
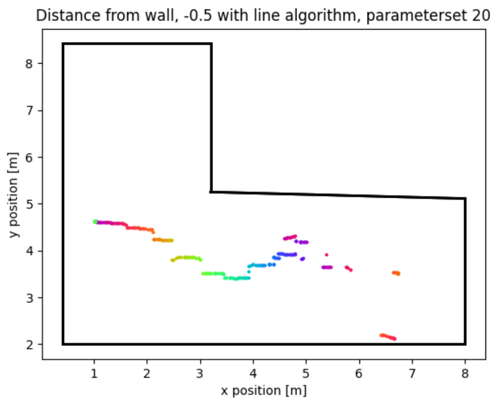


Figure 4.9: The number of meters the robot drove before the error is <0.1m. The dots are the measurement points. Each test is performed three times, and the data points represent the average distance from the tests that were passed. The green, blue, magenta and red dots indicate respectively 3, 2, 1, or 0 successful recoveries. For these tests, the distance the robot drove before recovery is set to 5 meters when no recovery succeeded.

Figure 4.10a indicates that the recovery succeeded according to the GT plot. However, the 2D plot, figure 4.10b shows that this is not the case. The robot should drive straight ahead in one straight line and end at y position 3.7m. This behavior is visible at a particle percentage of 98% and 75% which means that this range of values is not suitable for reliable localization. Only at max 25% particles from the line algorithm, LAMCL can recover the location almost all the time. Contrary to all recovery at the default setting with a rate of 10%. So, this shows that not all particles can be used from the line algorithm during resampling.



(a) GT plot



(b) 2D plot position

Figure 4.10: In the GT plot the robot looks like it is recovered, but the 2D plot shows differently. The colors of the routes are for indication in order to better follow the trajectory when the localization pose is staggered.

Change of weights of added particles

Parameter sets 30, 31, and 32 are about the difference in weight that the added particles from the line error have compared to the other resampled particles during resampling. It can be seen in figure 4.11 that with a larger weight division, the time to recover increases. This is due to the fact that the particles that first have a lower weight take longer to become the particles with the highest weight before LAMCL assigns it as the correct pose. This suggests that a lower division in the weight of the added particles is better.

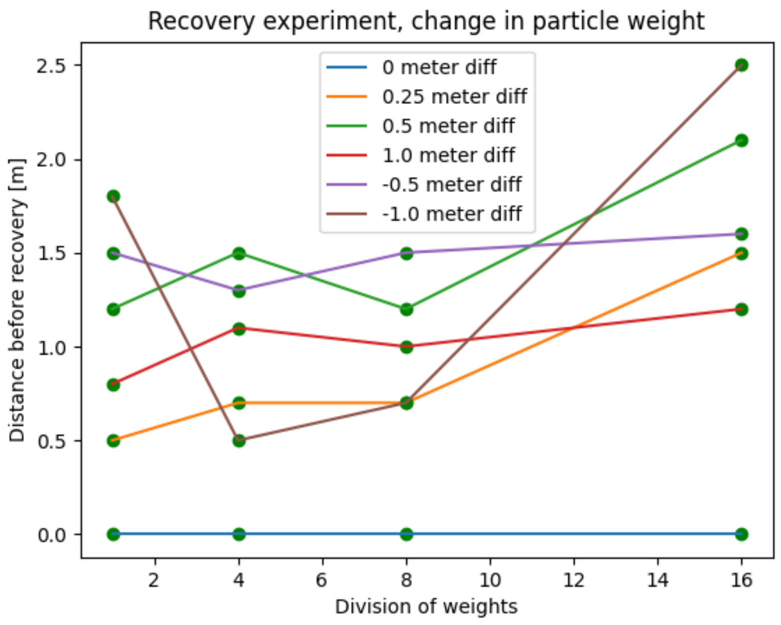


Figure 4.11: The number of meters the robot drove before the error is <0.1m. The dots are the measurement points. Each test is performed three times, and the data points represent the average distance from the tests that were passed. The green, blue, magenta and red dots indicate respectively 3, 2, 1, or 0 successful recoveries. For these tests, the distance the robot drove before recovery is set to 5 meters when no recovery succeeded.

Change of history queue size

Parameter sets 40, 41, 42, and 43 involve the size of the history with line segments that are included in calculating the average of the localization error. It can be seen in figure 4.12 that the change in the queue has an impact on how fast the robot does a recovery. The difference with the smaller history queue is not large and somewhat random. With a larger history size, it does show quite an increase in distance before a recovery takes place. In addition, it is expected that with a smaller queue, the extra particles will be put in the right location faster. However, AMCL also needs time to assess these new poses and change the weights. This probably takes more time than the time gained with a shorter queue. So, increasing the history queue size causes a longer time before recovery.

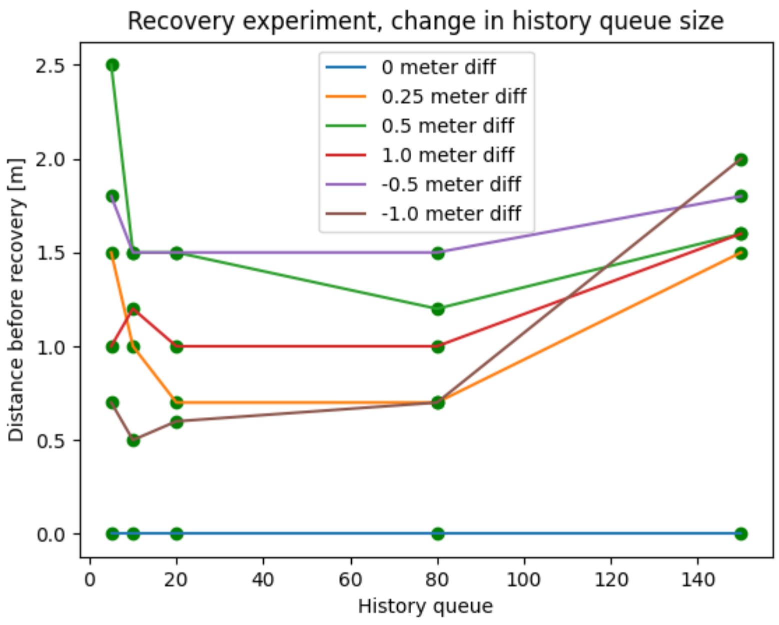


Figure 4.12: The number of meters the robot drove before the error is <0.1m. The dots are the measurement points. Each test is performed three times, and the data points represent the average distance from the tests that were passed. The green, blue, magenta and red dots indicate respectively 3, 2, 1, or 0 successful recoveries. For these tests, the distance the robot drove before recovery is set to 5 meters when no recovery succeeded.

Change of standard deviation of added particles

Parameter sets 50, 51, and 52 pertain to the standard deviation of particles when placed from a line error. Figure 4.13 shows that at the end when the standard deviation is higher, the recovery takes place less quickly. The reason for this is that the additional particles in a larger cloud around the error position reduce the likelihood of a particle in the correct position. This is because the same number of particles are distributed over a larger area. There is no indication in the graph that the distance before recovery of the 0.25 meter difference is not in line with the rest with a low standard deviation. During these two tests, it was not the line algorithm that caused the recovery to take place, but AMCL itself. The small error caused the current particles to have a high enough weight to remain active. Eventually, the extra uncertainty added during the update step with moving the particles by odometry caused the particles to move to the right place. To conclude, when the standard deviation of the placed particles according to the line error is higher, the recovery time will also be higher.

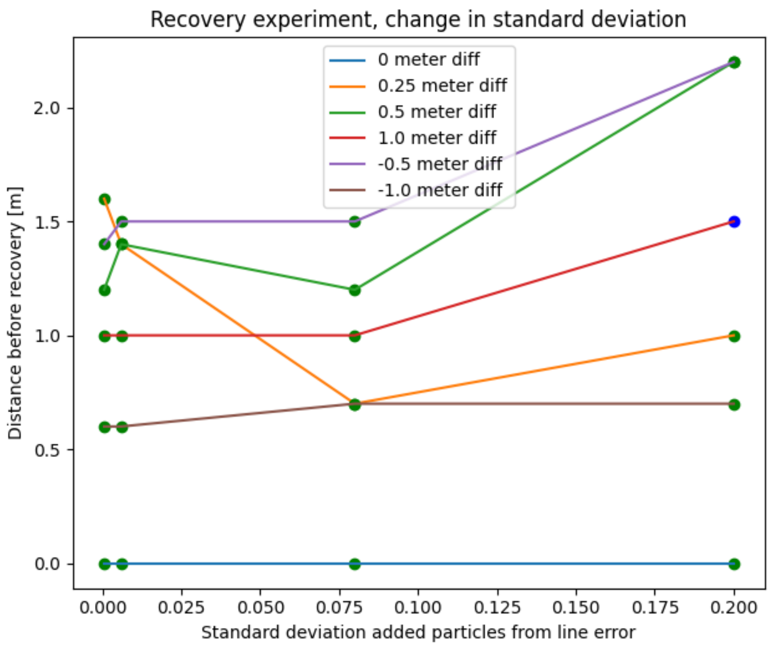


Figure 4.13: The number of meters the robot drove before the error is <0.1m. The dots are the measurement points. Each test is performed three times, and the data points represent the average distance from the tests that were passed. The green, blue, magenta and red dots indicate respectively 3, 2, 1, or 0 successful recoveries. For these tests, the distance the robot drove before recovery is set to 5 meters when no recovery succeeded.

The full list of all the exact distances driven until a recovery was performed at the various tests can be seen in appendix C.

4.4. Accuracy

The accuracy is about what the offset of the calculated robot pose is from the real robot pose. Testing the accuracy is done with two different types of tests. The second test is described in chapter 4.6 where only the pose is computed which tests the algorithm with an open loop. In this section, the closed-loop accuracy is tested. Here, each time the current pose is used to let the robot drive its route. The robot starts at a known location A and must drive to location B. On position B is measured how much the robot differs from that location in real life.

4.4.1. Experiment setup

In this experiment, the robot drives a pre-programmed route from location A to B. These locations are marked on the ground in the test area. The robot was placed at location A for each individual test in this experiment and automatically driven to location B. At location B, the x and y positions [m] were measured with an accuracy of 0.005m. By measuring the displacements between the robot's corners, the orientation [rad] of the robot was also determined. The experiment was conducted in a static environment and aimed to test whether the addition of the line algorithm did not significantly decrease accuracy, as additional particles are introduced in case of localization errors, which could potentially degrade accuracy. Only the default parameters were used in this experiment due to the time-consuming process of setting up the robot at location A, driving it, and accurately measuring the pose at location B. The route of this experiment is shown in figure 4.14. The route was driven 3 times with and without the addition of the line algorithm.

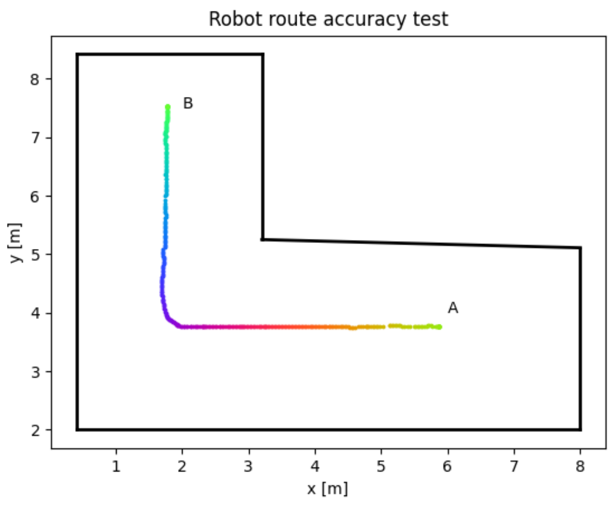


Figure 4.14: The route of the accuracy test with start point A and endpoint B. The colors of the route are for indication in order to better follow the trajectory when the localization pose is staggered.

4.4.2. Results

Figure 4.15 shows the result of the accuracy experiment. As can be seen, the self-localization accuracy is higher when no additional particles are sampled based on the detected line errors. The figure also reveals that adding particles with a standard deviation reduces the average accuracy by approximately 0.015m compared to when no line algorithm is used. Considering that the deviation can already be 0.11m, this is not a significant deterioration (see appendix D). In the appendix can also be seen that the measured Δtheta increases. This is due to the fact that the robot was not fully aligned after maneuvering through the corner and had not traveled straight for a sufficient distance yet. A detailed overview of the accuracy test can be found in appendix D. The large difference in localization error in the x- and y-axis is because there is a difference between the real test environment and the map that the localization uses. The leftmost wall in figure 4.14 is 0.08m more to the left in the real world.

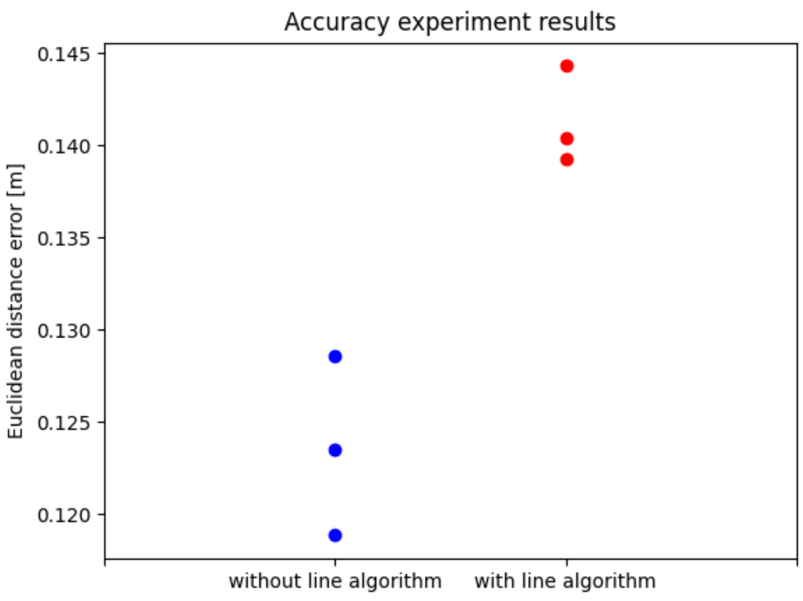


Figure 4.15: Accuracy difference between with and without line algorithm. The Euclidean distance between the Δx and Δy of the tests in table 4.4.

Without line algorithm			
	Measured difference		
	Δx [m]	Δy [m]	Δtheta [rad]
Test 1	0.115	0.045	0.044
Test 2	0.125	0.030	0.042
Test 3	0.110	0.045	0.053
Average	0.117	0.041	0.047

With line algorithm			
	Measured difference		
	Δx [m]	Δy [m]	Δtheta [rad]
Test 1	0.130	0.053	0.041
Test 2	0.140	0.035	0.033
Test 3	0.120	0.060	0.047
Average	0.130	0.050	0.041

Table 4.4: The pose difference of the 3 different accuracy tests with and without line algorithm. The average is the average of the three numbers in the column above. According to the coordinate system of figure 4.14.

4.5. Precision

This experiment looks at the precision of the robot. This is done by driving the same route several times and overlaying the trajectories of the routes. Because of the large amount of time it takes to drive the route a number of times with all the different parameter sets, it was chosen to do this only with the default setting of the line algorithm and without the line algorithm. For all other parameter sets, the robot drove the same route 5 times where the input from AMCL was a recorded rosbag. These were replayed with all different parameter sets later in the simulation to simulate the localization of the route. An important fact is that for this experiment, the base_link of the robot was used as the robot's position. This is different from the given route in figure 4.16 where the front of the robot is used for the waypoints.

4.5.1. Experiment setup

The robot starts its route each time from the charger and drives counterclockwise, shown in figure 4.16. The pose of the robot is saved at 10Hz during the tests. When the robot is in the charger, the pose is automatically reset to the location of the charger. When recording the rosbags, the route is recorded 2 times. One in a static environment and one with a person walking through to see how the line algorithm handles this and what the effect is on the robot's pose. A cow in the test environment was unfortunately not possible, for this, the experiment in chapter 4.6 was performed in the barn.

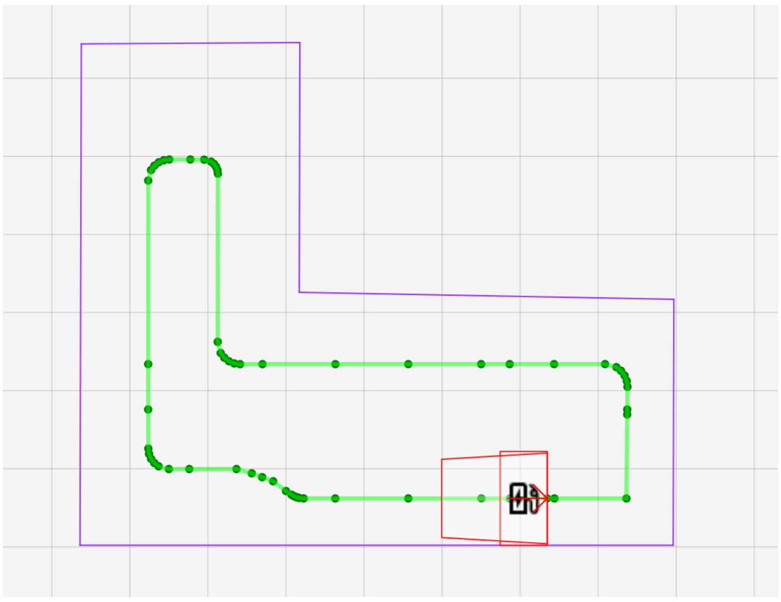
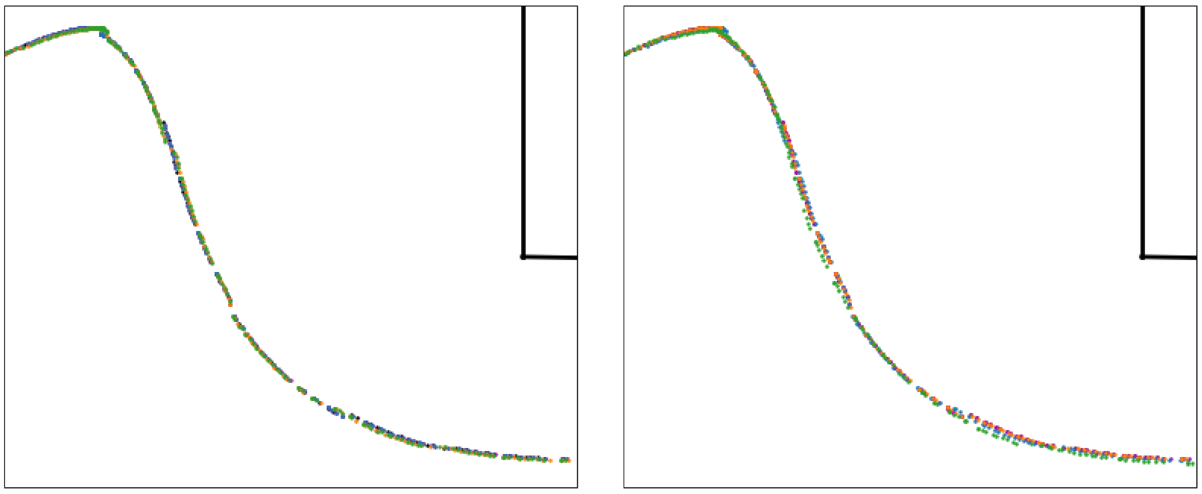


Figure 4.16: Route and environment of precision experiment. The route with the waypoints for the front of the robot is represented in green. The red lines are the outline of the robot in its charging pose. The size of the grid is 1x1m.

4.5.2. Results

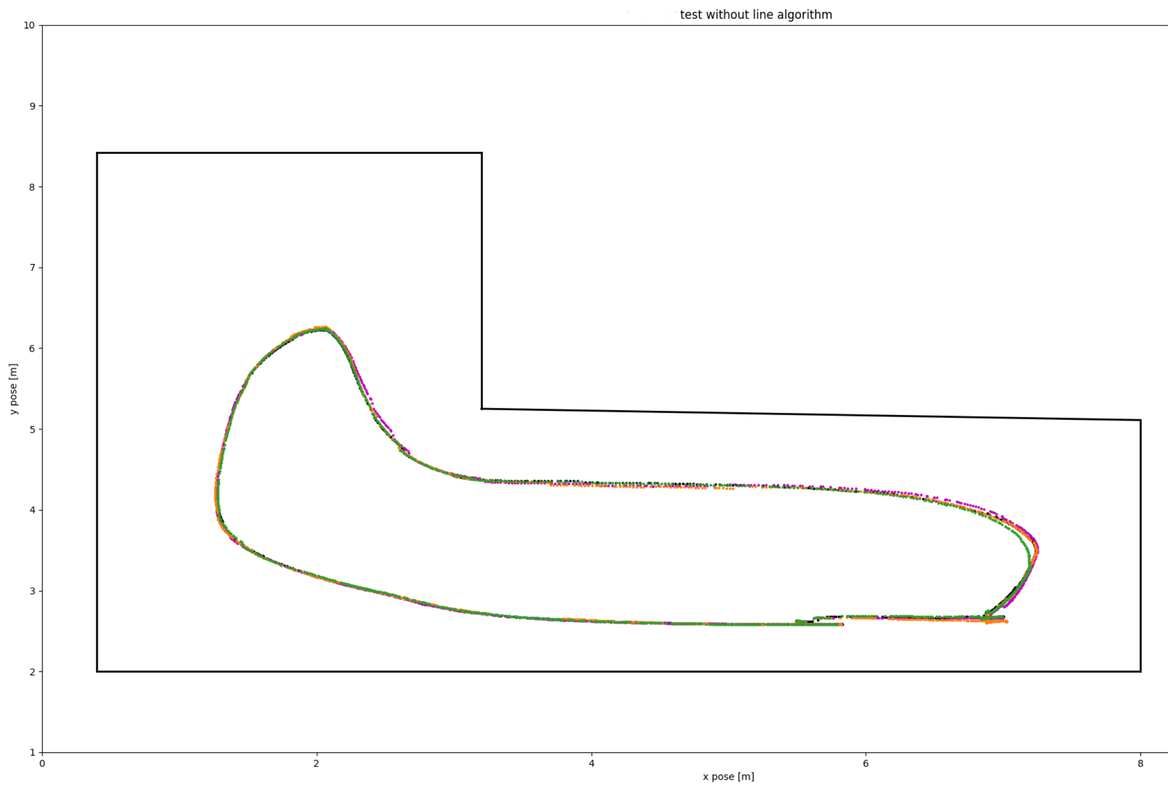
Figure 4.18 shows the comparison of the precision with and without the line algorithm. Note that this is in the closed loop. So, in these two tests, the robot drives around 5 times in real life. It can be seen that the localization is more precise when the correction of the localization with the line algorithm is not used.

This is because particles are added with a standard deviation of 0.08 when there is a line localization error. This is a distribution of up to 0.16m around the error position. A consequence of this is that the robot's pose may also deviate more than when no extra particles are added. When the dispersion of route lines is measured at location (4,4) it can be seen that without the line algorithm, it is 0.1m and with the line algorithm it is 0.15m. Thus, adding additional particles based on the line algorithm lowers the precision of the closed-loop system. For the parameter comparison of the system is tested in an open loop. This means that there is no correction when the robot's location is not according to the route. The input is the same for both tests with and without the line algorithm. In figure 4.17 (crop at location (3,5) of whole test hall) can be seen that there is a much lower variation in the different routes for both the tests. The difference without is 0.02m and 0.045m with the line algorithm. This could be due to that these tests have the same input compared to the closed-loop tests.

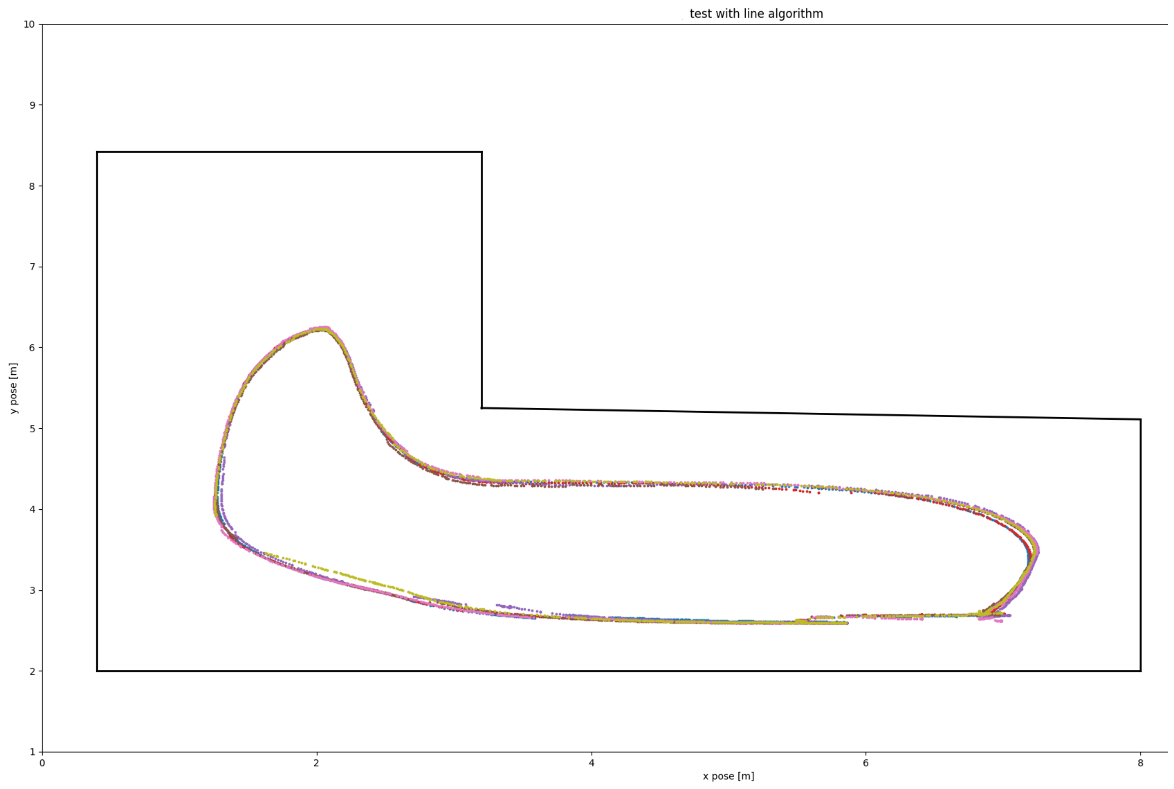

(a) Multiple routes without line algorithm

(b) Multiple routes with line algorithm

Figure 4.17: Precision experiment with and without line algorithm in open-loop. Cropped figures of the corner at the top of the route. In black is the corner of the environment.



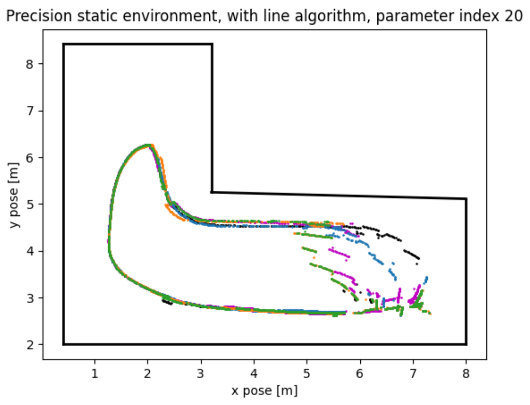
(a) Multiple routes without line algorithm



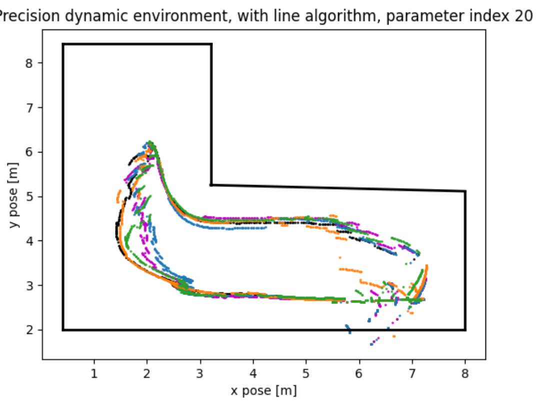
(b) Multiple routes with line algorithm

Figure 4.18: Precision experiment with vs without line algorithm in closed-loop

The different routes for each parameter set can be found in Appendix E. For almost all parameters, it is not noticeable that the different routes per set differ from each other compared to the other sets. However, this is visible when the percentage of particles from the line algorithm is changed. In parameter set 20, where the part from the line algorithm is 98%, it can be seen that the different routes are significantly different from each other (figure 4.19). For the route in the dynamic environment, it is even more clear that the error of the lines does change the software position of the robot. Also, in parameter set 22, with a part from the lines at 75%, a variance like this can be seen in the routes.



(a) Multiple routes with parameter set 20, static environment



(b) Multiple routes with parameter set 20, dynamic environment

Figure 4.19: Precision experiment parameter set 20

4.6. Real world setting

This experiment focuses on how the algorithm performs in the environment of a barn with cows. The accuracy is taken as a method to calculate the performance of the system. Using the DataCow dataset, all different parameter sets are compared with each other. Thus, this experiment also tests how the algorithm deals with a dynamic environment. A total of 4 different routes are recorded in the barn, see figure 4.2.

4.6.1. Experiment setup

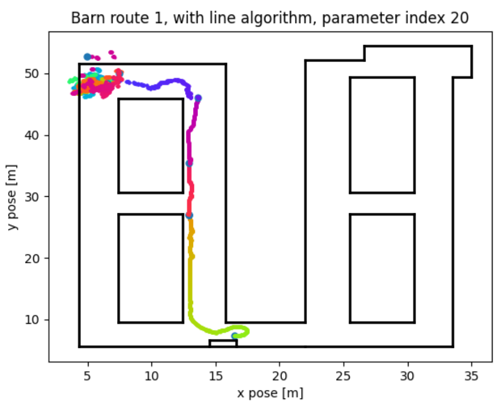
In this experiment, the DataCow dataset was used to analyze the performance of the system. The process of recording the data is described in section 4.1.1. The dataset is the input to the simulation that is replayed with each parameter set. This makes it look like the robot drives all 4 routes with a different parameter set each time. The robot only stops at each measurement point, this allows the measured positions in the barn to be automatically linked to the localization positions.

4.6.2. Results

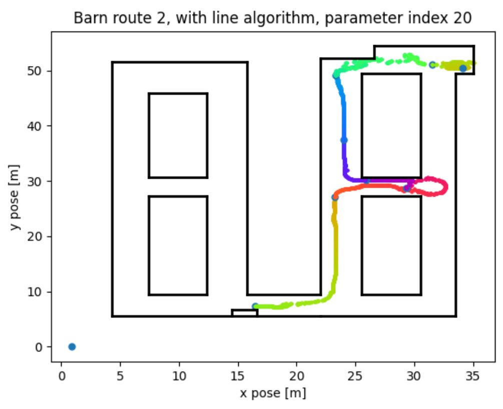
The metric used in the barn experiment is the mean absolute error (MAE, formula (4.1)) for the x, y, and orientation of the robot. Not all routes were driven correctly. For example, figure 4.20 shows that when all particles are resampled based on the localization error from the line algorithm, the routes are not complete. This was also the case for 0.75% and 0.5% where 2 out of the 4 routes are not complete. This can be seen in the error of the routes which is higher than the others. Appendix F shows an overview of the average errors of the routes per parameter set. Figure 4.21 shows a cut-out of the upper right half of route 4 in the barn. The localization in this turn gives variation between the different parameter sets because there was a cow in front of the robot before this turn was made. Besides the different error values in the localization, it can also be seen here that there is a variation between all parameter sets which might not be visible from the mean absolute error. The recap of the corresponding parameters can be found in an overview in appendix B.

$$\text{MAE}(y, \hat{y}) = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|}{N} \tag{4.1}$$

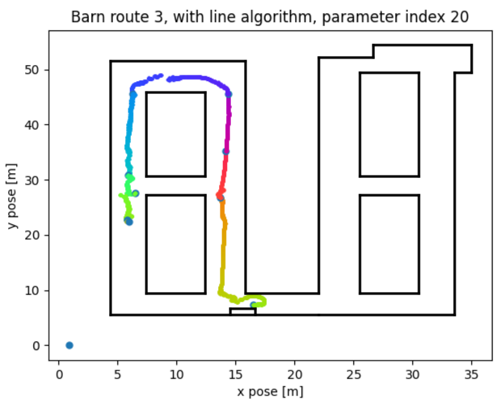
$y(i)$: Ground truth (GT) position of the robot
$\hat{y}(i)$: Calculated position of the robot
N : Number of positions



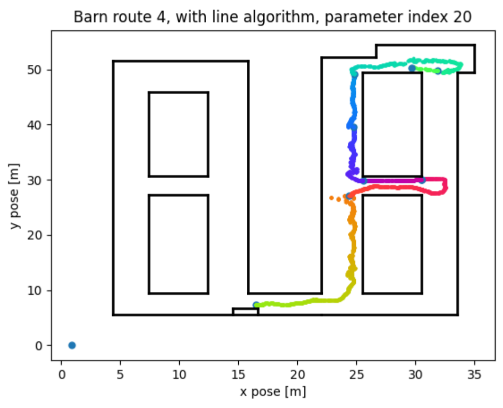
(a) Barn route 1 with all particle resamples from line algorithm



(b) Barn route 2 with all particle resamples from line algorithm



(c) Barn route 3 with all particle resamples from line algorithm



(d) Barn route 4 with all particle resamples from line algorithm

Figure 4.20: Routes in the barn with all particle resamples from line algorithm. The dots represent the position where the robot stood still. The colors of the routes are for indication in order to better follow the trajectory when the localization pose is staggered.

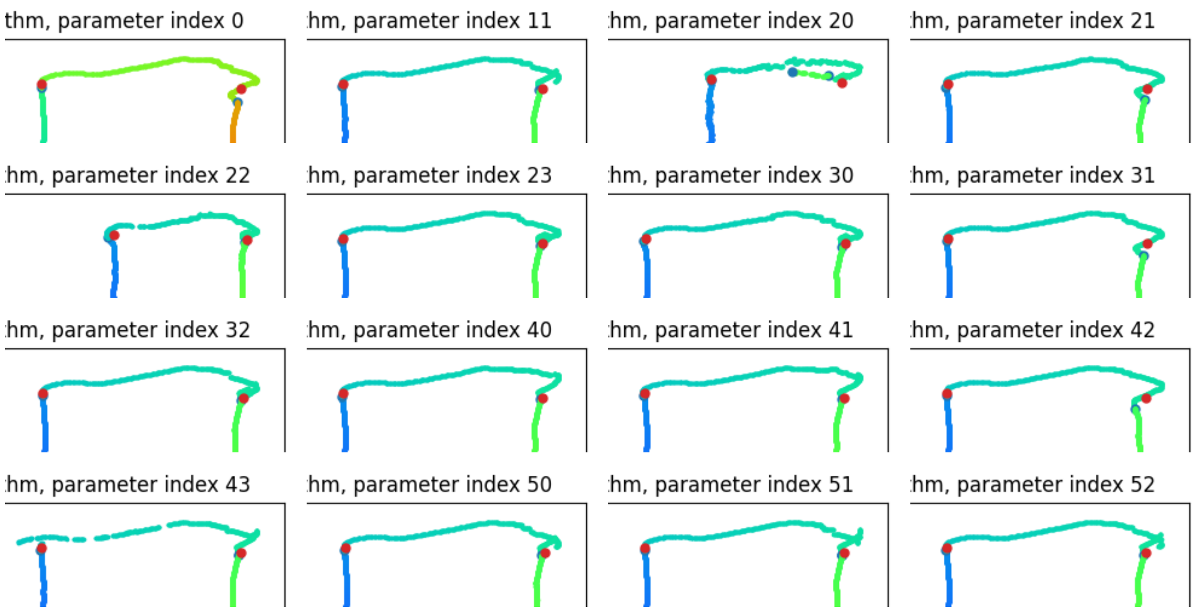


Figure 4.21: Cutout top right corner of route 4 in the barn with different parameter sets/index. The colors of the routes are for indication in order to better follow the trajectory when the localization pose is staggered. The red dots is where the robot stood still in real life. The blue dots are where the self-localization thinks the robot stood still.

In order to examine the effect of the line algorithm and the different parameters on the accuracy, the Euclidean distance of the mean absolute error was measured for each change in parameter when using the Datacow dataset to evaluate the different configurations.

Change in line detection

To test the effect of the lines in the performance in a real-world situation, four different parameter sets were considered. These are the same four parameter sets as in the recovery ability experiment in chapter 4.3.3. In general, there is no major variation in the different situations. However, it can be seen that the errors of routes 2 and 4 are higher than routes 1 and 3. Both routes 2 and 4 were made in the same area of the barn. This shows that one side of the barn is more difficult than the other. It also shows that the routes with more dynamics (routes 3 and 4) have a higher error than those with less dynamic obstacles (routes 1 and 2). As explained in chapter 1.2, if cows block the camera's view, they cause an increase in localization uncertainty at those moments. This is probably also the reason that for routes with more cows, the error is much higher.

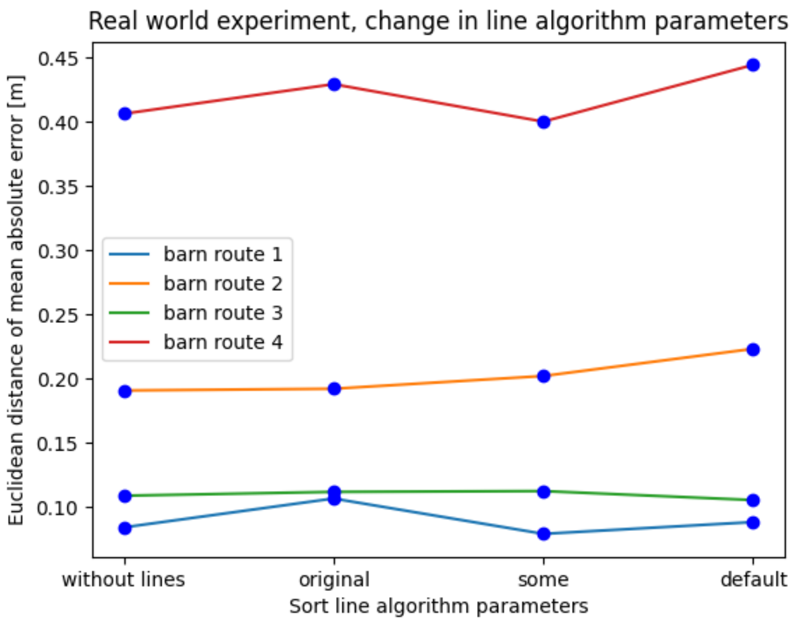


Figure 4.22: Real-world experiments in which 4 different routes were driven in the barn. The routes are shown in figure 4.2

Change in part of particles from line detection error

Parameter sets 20, 21, 22, and 23 deal with the difference in what percentage of the line detection error should come during resampling. This is shown in figure 4.23. The plot shows that the greater the percentage of resample coming from the line algorithm, the higher the localization error is. The significantly larger error at a resampling factor of 0.98 is caused by the route not being completed as shown in figure 4.20.

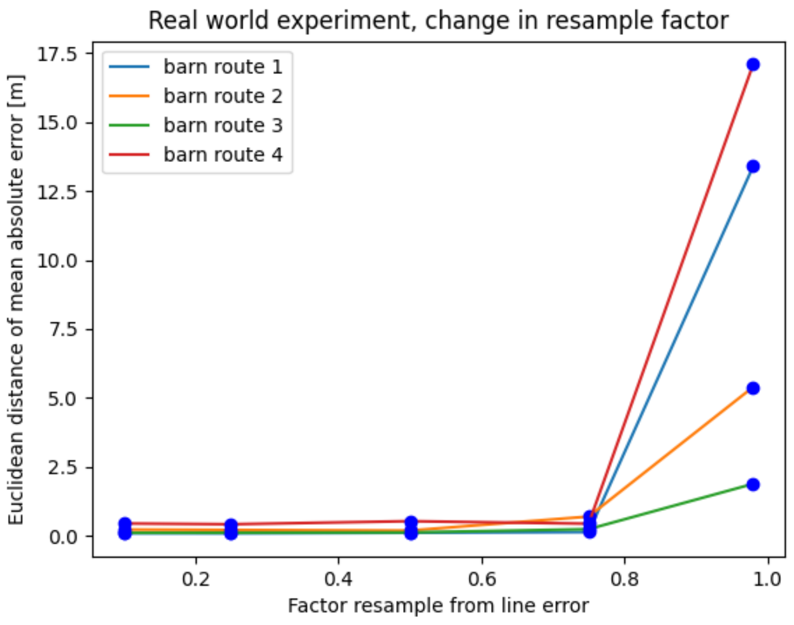


Figure 4.23: Real-world experiments in which 4 different routes were driven in the barn. The routes are shown in figure 4.2

Change in weights of added particles

Parameter sets 30, 31, and 32 are about the difference in weight that the added particles from the line error have, compared to the other resampled particles during resampling. Figure 4.24 shows the localization error relative to the weight of the particles. There is no clear correlation. However, the order of the different routes is the same as the order of the comparison with the different line detection situations. This suggests that there could be indeed a difference in difficulty in the different barn routes.

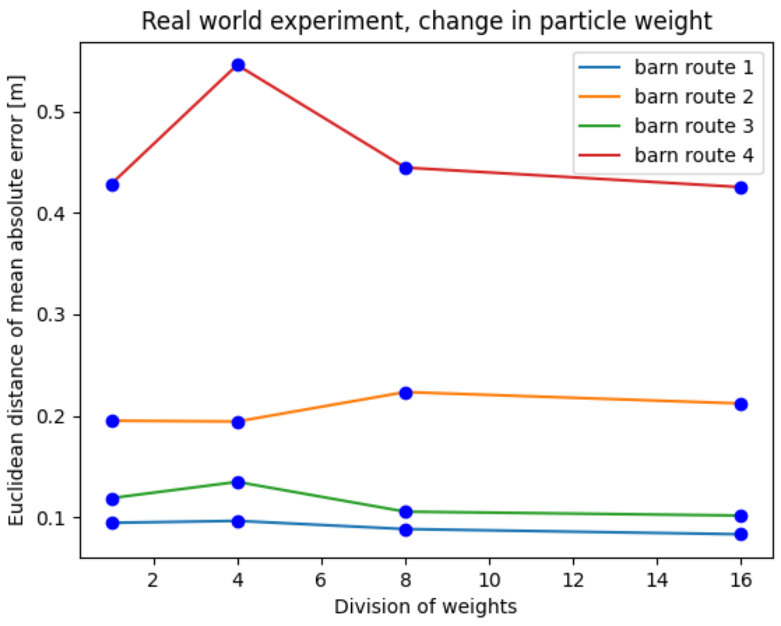


Figure 4.24: Real-world experiments in which 4 different routes were driven in the barn. The routes are shown in figure 4.2.

Change in history queue size

Parameter sets 40, 41, 42, and 43 involve the size of the history with line segments that are included in calculating the average of the localization error. It can be seen in figure 4.25 that the change in the queue has a minor impact on the localization error. Only in route 4, there is a positive relation between the history queue size and the error. This could mean that the history queue has more effect when there are more dynamic obstacles since route 4 has the most dynamism of the four routes.

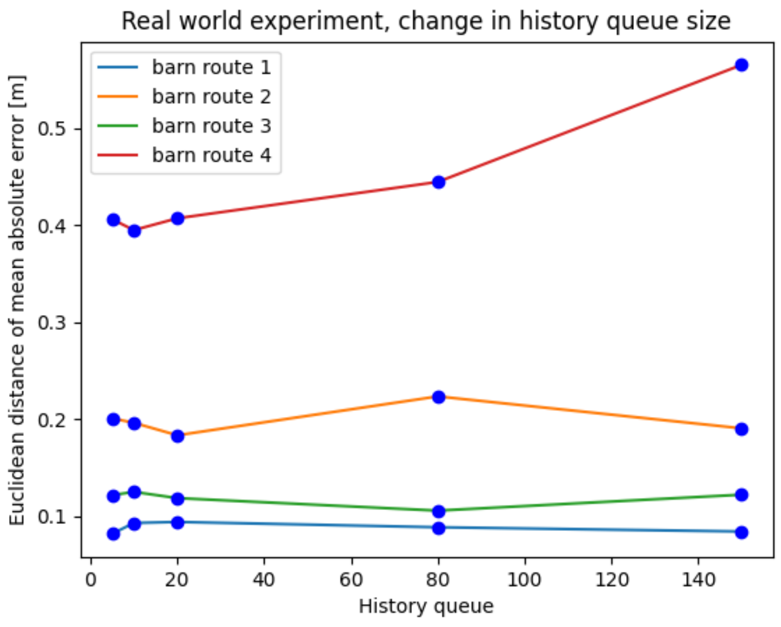


Figure 4.25: Real-world experiments in which 4 different routes were driven in the barn. The routes are shown in figure 4.2

Change in standard deviation of added particles

Parameter sets 50, 51, and 52 pertain to the standard deviation of particles when they are placed based on a line error. The relation between the standard deviation and the average localization error can be seen in figure 4.26. Between the different routes, the same ranking is visible as for the other parameter sets with a higher error to due a more dynamic route. The effect of changing the standard deviation is not affecting the result that much. It could be seen that with more dynamic obstacles in the route, a higher standard deviation has a positive influence on the localization error. This could be because with a more dynamic environment, more (false) errors are detected (line segments), but with a higher standard deviation there are enough checks in the environment of the error, and therefore a better position is found.

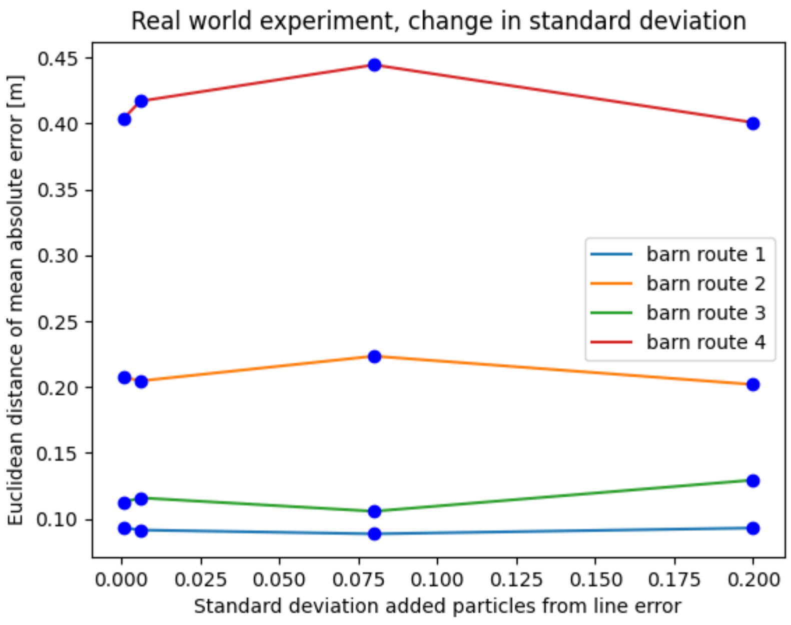


Figure 4.26: Real-world experiments in which 4 different routes were driven in the barn. The routes are shown in figure 4.2

4.6.3. Real world recovery example

Figure 4.27 shows an example of a recovery during a route in the barn. The localization error of 1.2 meters had occurred because cows blocked the robot's view and AMCL chose the wrong part of the particle cloud when a part of the view was clear again. With this example, the steps of the algorithm can also be followed: using line detection and comparison with the walls to find a localization error, and place particles at the location of the error, after which AMCL itself decides that this location better suits the LiDAR scan and choose the right pose.

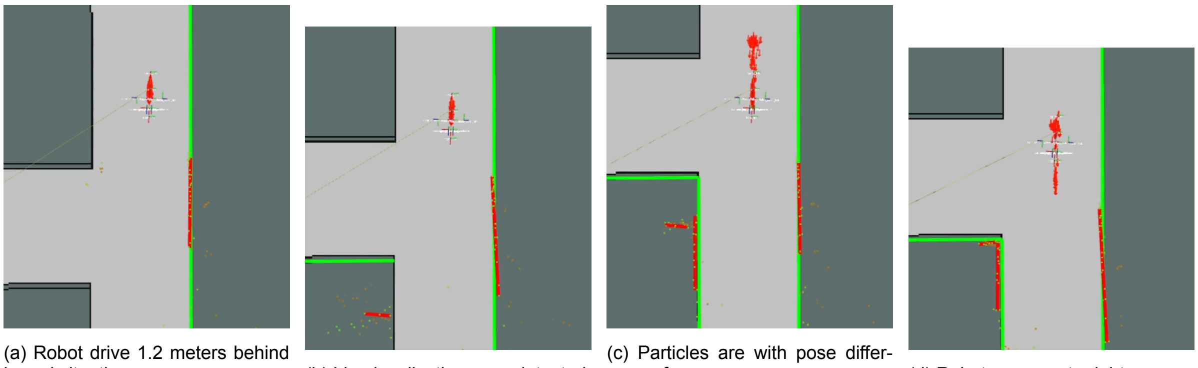

(a) Robot drive 1.2 meters behind in real situation
(b) Line localization error detected
(c) Particles are with pose difference of error
(d) Robot recovers to right pose

Figure 4.27: Process of recovery in the barn during route

5 Conclusion

The main research question of this master thesis is:

Does using extracted lines from a planar LiDAR improve the accuracy, precision, and robustness of self-localization with AMCL for a mobile robot in a barn?

This research question is separated into multiple sub-questions to answer the main research question.

1. *Can the localization with AMCL be improved by correcting for offsets by resampling particles through additional line extraction and matching between those line segments and map?*
2. *How can the accuracy, precision, and robustness of self-localization be tested and evaluated in a real-world barn environment?*
3. *How sensitive is the new self-localization to hyperparameter choices?*

Different experiments examined the impact of using a line algorithm along with AMCL. The experiments say something about the precision, accuracy, and recovery ability of the robot's self-localization. This gives more insight into the answer to sub-question 1 which will be answered in the next section.

5.1. Quality of robot self-localization

To evaluate the self-localization, three different methods were used: the precision, accuracy and the recovery ability of the robot.

5.1.1. Precision

To evaluate the precision an experiment was done in which the robot drove the same route several times in section 4.5. When AMCL is compared to LAMCL, the precision decreases, as shown in figure 4.17, from 0.1m to 0.15m in the closed loop and from 0.02m to 0.045m in the open loop. An explanation for this could be that adding extra particles with a Gaussian distribution gives an extra deviation around the position of the robot. As a result, there is a larger spreading where the robot might be, which lowers the precision.

5.1.2. Accuracy

For evaluating the accuracy, the first thing considered was how LAMCL performed compared to AMCL. It was shown in table 4.4 that the accuracy deteriorates from 0.117m to 0.130m in the x direction and from 0.041m to 0.050m difference in the y direction. This, like the deterioration in precision, may be due to the fact that extra particles are added with a standard distribution. The rotation did improve in LAMCL compared to AMCL from 0.047 to 0.041 rad. However, this average improvement is negligible given the variation in the Δtheta of the experiment.

In Kang's paper [28] in which they also used a line algorithm to improve the localization, they did have improved the accuracy in position from (0.880, 0.421) in (x[m], y[m]) to (0.273, 0.329) error. Note that there was already a larger improvement to be made here at the beginning, as this error was much larger without a line algorithm than the robots' in the barn. In addition, in Kangs' paper, they adjust the location instead of adding additional particles.

In chapter 4.6 the accuracy was also measured at different locations in the barn. On average, the accuracy is better without the line algorithm. The minimal difference is probably because AMCL can already handle the majority of the difficult situations that occurred in the barn experiment.

5.1.3. Recovery ability

LAMCL has a slightly worse result than AMCL considering the precision and accuracy. This is due to the addition of extra particles in a distribution. However, this addition greatly improves the recovery ability of the system. The experiment in chapter 4.3 shows that LAMCL is faster and performs more recoveries when there is a localization error. Especially when the error is ≥0.5 meters. This is where LAMCL has the biggest advantage over AMCL. Even with a localization error of 1.5 meters, LAMCL manages to recover the location, which AMCL fails to do. However, the speed of recovery does correlate with the layout of the environment at how many lines from the line algorithm an error is measured.

5.2. Effect of parameters

Sub-question 3 is about the influence of the parameters on the system. Therefore, the influence of certain parameters in the localization was examined in 3 different experiments. For a number of parameter changes, no significant variation can be seen in the results. This is related to the observation that most of them do not have a significant influence on the overall system when they change within the range in which they were tested.

5.2.1. Change in line detection

The parameter sets with decade 1 looked at the effect of changing the line algorithm. When the original parameters of the algorithm [22] are used there are no lines detected. This is not easily seen in the barn test. For example, where the errors can still be caught by AMCL itself. In the recovery test, it is only seen that no recoveries are performed, because no lines are detected. So the maximum standard deviation in the measurement points of the pseudo LiDAR scan is certainly an important parameter and it is crucial that it is set well enough so that the lines are detected.

5.2.2. Change in part of particles from line detection error

The original paper by Kang [28] moves the location of the robot when a localization occurs. By adjusting the percentage of particles from the localization error the effect on the localization of the robot in the barn is examined. By moving almost all particles based on the localization error it can be seen in the barn test, in figure 4.20, that the localization does not function properly. This is probably due to the fact that there are also line segments of dynamic obstacles among them that have a large localization error. With 75%, the accuracy and recovery rate were still lower than average relative to a low percentage of particles from the line algorithm. However, when recovery is done with 50% it does go a bit faster than with a lower percentage, as seen in figure 4.9. This is due to more particles being in the right location. So, it is important that not all particles are adjusted according to the line error. Only when the percentage is ≤25% the algorithm can work properly to perform recoveries.

5.2.3. Change of weights of added particles

The new particles that are placed based on the line algorithm do not have the same weight as the particles from resampling. In the recovery experiment, in figure 4.11, it can be seen that when the weight is higher (division is smaller) the algorithm recovers faster. This is logical since less time is needed for the newly placed particles to reach the highest weight. In the barn test, it varies between the different parameters, so there is no optimum in terms of accuracy. The weight of the added particles has the most influence on the speed of recovery of the algorithm.

5.2.4. Change of history queue size

The change in queue size does show a less obvious linear relationship between the speed of recovery and the size of the history queue. The relation is expected because, with a shorter queue size, the average of the queue is more likely to be the line error. This limited relation between the history queue and distance before recovery is due to that during the experiment, the robot first stood still in front of the wall before driving towards it. During the standstill, the wall and thus the localization error was already visible so the queue was already partly filled before the robot started driving. So it is uncertain what effect adjusting the history queue size has in real-world operations.

5.2.5. Change of standard deviation of added particles

The recovery experiment shows in figure 4.13 that the larger the standard deviation is, the longer it takes for the recovery to take place. This has to do with the fact that with a smaller standard deviation, there is a bigger chance that a particle is located at exactly the right pose and matches the LiDAR scan extremely well. The barn test shows that a higher standard deviation gives a slightly higher localization error. This would be in the trend that the standard deviation brings additional inaccuracy to the system.

It was mentioned in chapter 3 that it is also possible to add random particles around the robot's pose with a normal distribution during resampling, instead of based on the line errors. This would be equivalent to making the standard deviation very large. When the trend of the distance before recovery is extended from figure 4.13, it can be seen that with the addition of random resamples, the distance before recovery becomes even larger than resampling based on the line detection error. In the real-world experiment, there is no clear trend in the tests to say anything about accuracy. So, adding random particles in a standard distribution about the robot would result in a slower recovery time.

5.3. To conclude

It can be concluded from the experiments that adding extra particles to the resample state of AMCL improves the recovery ability and thus the robustness of the self-localization. The disadvantage is that this comes at a small cost in terms of precision and accuracy performance. This is because the new line error particles are placed with a Gaussian distribution around the error location. From the parameters, the percentage of particles resampled based on line error has the most influence. If all the particles are resampled from the line error, the self-localization will not work. Regarding the impact of the parameters, reducing the weights or increasing the history queue size or standard deviation will result in a decrease in the performance of the new localization method LAMCL. It is shown that precision, accuracy, and robustness can be tested and evaluated by experiments in chapter 4. Chapter 4.1 has shown that a dataset can be used for evaluation in a real-world barn environment if a ground truth is granted.

5.4. Future work

There are still several areas that can be improved in LAMCL. For example, the effect of another line algorithm could still be looked into. As an example, in the related work section Over-segmentation, undirected graph, and line extraction method [31] has been mentioned. This line algorithm shows an improvement in performance over the currently used Split and Merge. Another method to check out would be Evolving Principal Component Clustering [11], where EPCC is used together with Split-and-Merge for better results. However, it would be good to test the line algorithm in isolation instead of the whole system as it is done now. This would require more time and a dataset with line segments and their GT. This would also allow testing more different parameter sets of Split-and-Merge. In addition, Split-and-Merge uses a threshold for the maximum deviation in the measurements. The pseudo LiDAR has an increasing deviation in the measurements as the distance to the measurement points increases, as seen in chapter 4.3.2. Therefore, it may be worth considering whether the range deviation in the line algorithm can be varied with the distance at which the measurements are detected.

Another area to improve is when comparing line segments to walls. For example, a line segment is now compared to each wall of the map to find the best matching wall. This could also be just the walls in the direct surroundings to make the algorithm run faster. In addition, right now it does not take into account when a wall is cut in two but continues. This sometimes causes line errors to be noticed even though the line detected is also part of another wall that is an extension of the current compared wall.

Currently, the particles are added based on the line errors with a maximum frequency of 2 Hz. This could also be made dependent on how much the robot has driven in the meantime.

The different parameter sets are limited by how much time there was to test. This could also be investigated further to see how much influence they have and what their limits are. For example, in figure 4.25 it can be seen that a lower history queue decreases the localization error. It could be interesting to test a history queue of 1. In this way, particles will be placed based on every single line error. Another example is the fraction of the particles from the estimated localization error based on the line algorithm. There is no data between 75% and 98% to check where the fail threshold is.

As a final thought, the new algorithm is only tested with 4 routes in the barn. To test the full potential of the new functionality, it is useful to run the LAMCL on the robot for several days in a barn. This will allow us to see how the new algorithm handles situations that did not occur in the DataCow dataset.

A
Barn layout

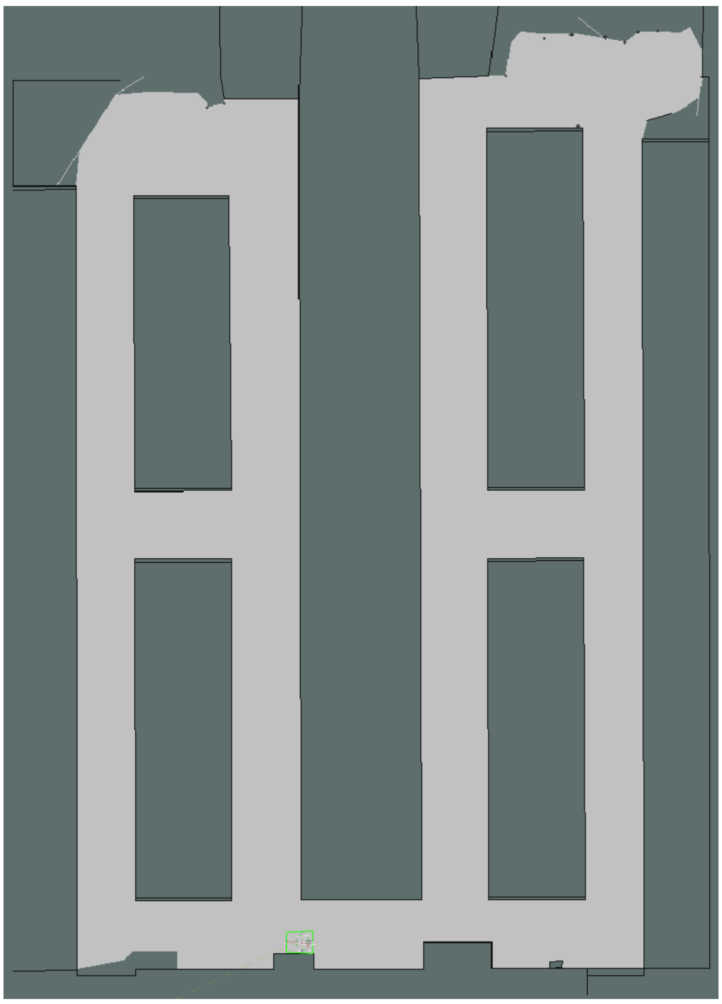


Figure A.1: The layout of the barn. Route 1 and 3 were driven on the left wing and route 2 and 4 on the right hand-side. The start pose of the robot in the routes is visible in green at the bottom.

B
Parameters

Line algorithm																	
Parameter	**Unit**	**par0**	**par10**	**par11**	**par20**	**par21**	**par22**	**par23**	**par30**	**par31**	**par32**	**par40**	**par41**	**par42**	**par43**	**par50**	**par51**	**par52**
bearing std dev	rad	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001
range std dev	m	1	**0,02**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
least sq angle thresh	rad	0,1	**0,0001**	**0,0001**	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
least sq radius thresh	m	0,1	**0,0001**	**0,0001**	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
max line gap	m	1	**0,4**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
min line length	m	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5
min range	m	0,1	**0,4**	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
max range	m	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4	6,4
min split dist	m	0,5	**0,05**	**0,3**	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5	0,5
outlier dist	m	0,6	**0,05**	0,05	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6
min line points	-	6	**9**	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
Line-wall compare																		
wall max history	-	80	80	80	80	80	80	80	80	80	80	**20**	**10**	**150**	**5**	80	80	80
max allowed angle difference	degrees	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
max allowed pose difference x	m	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
max allowed pose difference y	m	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
distance start	m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
distance end	m	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
angle start	degrees	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
angle end	degrees	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60
Place particles on error pose																		
min time between lamcl resamples	sec	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
std dev pose x	-	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	**0,006**	**0,0006**	**0,2**
std dev pose y	-	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	0,08	**0,006**	**0,0006**	**0,2**
std dev orientation	-	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,002	**0,0002**	**0,2**
fraction resample from line error	-	0,1	0,1	0,1	**0,98**	**0,5**	**0,75**	**0,25**	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
division weight added particles	-	8	8	8	8	8	8	8	**1**	**4**	**16**	8	8	8	8	8	8	8

Table B.1: All the parameter sets and their changes relative to the default parameter set 0 in bold.

C
Recovery experiment result overview

Distances	0		0.25		0.5		1.0		-0.5		-1.0	
Parameter sets	**success**	**distance**	**success**	**distance**	**success**	**distance**	**success**	**distance**	**success**	**distance**	**success**	**distance**
No line algorithm	3	0	2*	1.7	1*	1.9	0	-	1*	3.0	0	-
0	3	0	3*	0.7	3	1.2	3	1.0	3	1.5	3	0.7
10	3	0	3*	1.0	2*	2.2	0	-	0	-	0	-
11	3	0	3*	1.0	3	1.3	3	1.1	3	1.8	3	3.0
20	1	0	1	4.0	0	-	0	-	3	2.8	2	2.5
21	0	-	1*	1.0	1	1.1	2	0.8	3	0.9	3	0.5
22	3	0	1	0.7	0	-	0	-	3	0.7	3	1.7
23	3	0	3*	1.2	3	1.3	3	1.0	3	1.3	2	0.5
30	3	0	3*	0.5	3	1.2	3	0.8	3	1.5	3	1.8
31	3	0	3*	0.7	3	1.5	3	1.1	3	1.3	3	0.5
32	3	0	3*	1.5	3	2.1	3	1.2	3	1.6	3	2.5
40	3	0	3*	0.7	3	1.5	3	1.0	3	1.5	3	0.6
41	3	0	3*	1.0	3	1.5	3	1.2	3	1.5	3	0.5
42	3	0	3*	1.5	3	1.6	3	1.6	3	1.8	3	2.0
43	3	0	3*	1.5	3*	2.5	3	1.0	3	1.8	3	0.7
50	3	0	3*	1.4	3	1.4	2	1.0	3	1.5	3	0.6
51	3	0	3*	1.6	3	1.2	3	1.0	3	1.4	3	0.6
52	3	0	3*	1.0	3	2.2	2	1.5	3	2.2	3	0.7

Table C.1: Results recovery experiment. Recovery is when the robot location is within an error of 0.1m. Successes are the number of times out of 3 tests. Distance is the distance [m] traveled before recovery. *(recovery by change of AMCL, not lines algorithm). The tens in the numbers of the parameter sets are an indicator of what is being varied in those parameter sets by that decimal.

D
Accuracy experiment result overview

pose B	
x [m]	1.77
y [m]	7.47
theta [rad]	1.5708

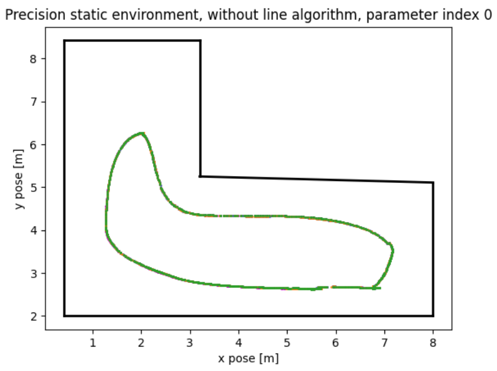
Table D.1: Pose B of accuracy experiment with in bold the most important comparison of the accuracy experiment

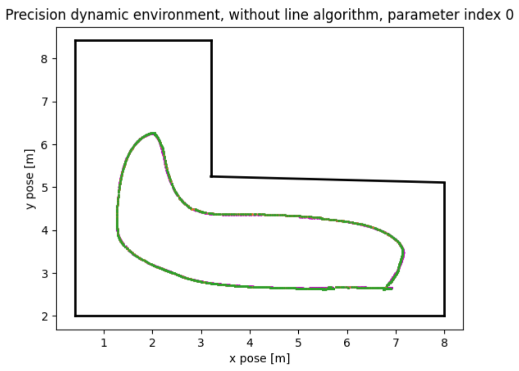
	Without line algorithm																		
	Measured pose				Measured difference				Software pose				Software pose real, difference				software-measured difference		
	x [m]	y [m]	theta [rad]		Δx [m]	Δy [m]	Δtheta [rad]		x [m]	y [m]	theta [rad]		x [m]	y [m]	theta [rad]		x [m]	y [m]	theta [rad]
Test 1	1,885	7,515	1,615		0,115	0,045	0,044		1,771	7,522	1,573		0,001	0,052	0,003		0,114	-0,007	0,042
Test 2	1,895	7,500	1,612		0,125	0,030	0,042		1,772	7,508	1,574		0,002	0,038	0,003		0,123	-0,008	0,039
Test 3	1,880	7,515	1,624		0,110	0,045	0,053		1,767	7,508	1,583		-0,003	0,038	0,013		0,113	0,007	0,041
Average	1,887	7,510	1,617		**0,117**	**0,041**	**0,047**		1,770	7,513	1,577		0,002	0,043	0,008		0,116	0.007	0,040
	With line algorithm																		
	Measured pose				Measured difference				Software pose				Software pose real, difference				software-measured difference		
	x [m]	y [m]	theta [rad]		Δx [m]	Δy [m]	Δtheta [rad]		x [m]	y [m]	theta [rad]		x [m]	y [m]	theta [rad]		x [m]	y [m]	theta [rad]
Test 1	1,900	7,523	1,612		0,130	0,053	0,041		1,767	7,519	1,580		-0,003	0,049	0,010		0,133	0,003	0,031
Test 2	1,910	7,505	1,604		0,140	0,035	0,033		1,767	7,498	1,579		-0,003	0,028	0,008		0,143	0,007	0,025
Test 3	1,890	7,530	1,617		0,120	0,060	0,047		1,774	7,527	1,569		0,004	0,057	-0,002		0,116	0,003	0,048
Average	1,900	7,519	1,611		**0,130**	**0,050**	**0,041**		1,770	7,515	1,576		0,003	0,047	0,007		0,131	0,005	0,036

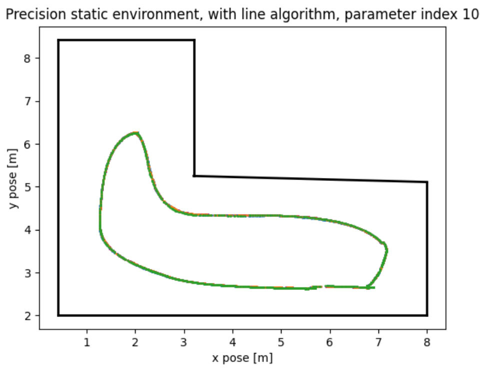
Table D.2: Accuracy experiment results. Six different tests including 3 with line algorithm and 3 without. The values in bold are the most important values of the table to compare the accuracy between with and without the addition of adding particles based on the detected line error.

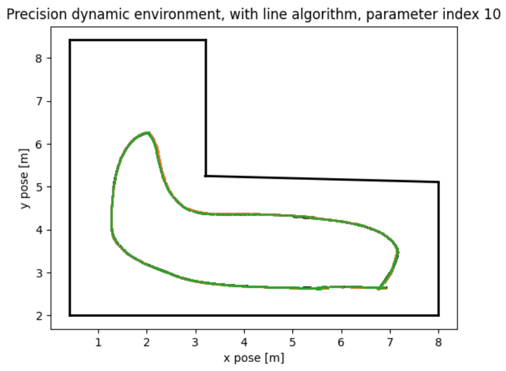
Table D.2 shows the results of the accuracy test. Here the ’measured difference’ is the measured difference of the robot with respect to the marked point B in the real world. Together with the values of pose B, this calculates the ’measured pose’. The ’software pose’ is the pose the robot thought it had. The ’software pose real, difference’ is the difference between pose B in the real world and the ’software pose’. The ’software-measured difference’ is the difference between the ’software pose’ and the ’measured difference’. An additional note about the software ’pose real, difference’ is that the difference between where it should be and where the robot thinks it is in the y-axis is more than 0.04m on average, while the error in the x-axis is extremely small. This is due to the fact that in pose B the robot drove in a positive y direction. The waypoints of the robot route are every 0.1m with interpolation in between. The robot must first drive over the end waypoint before it gets the signal that it is in position. This is reflected in these values that the robot drove further ahead in the y-axis than requested.

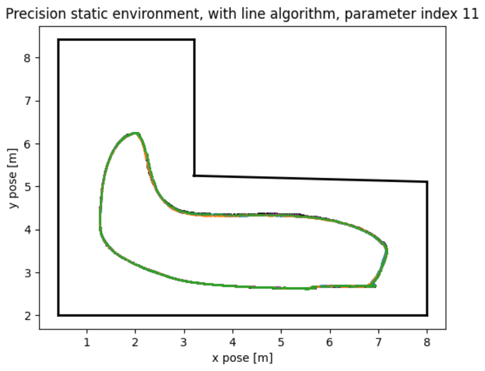
E
Precision experiment all plots

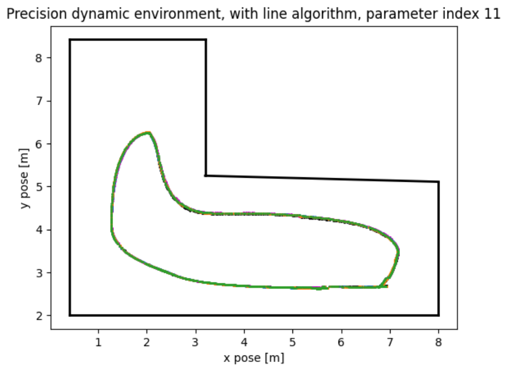




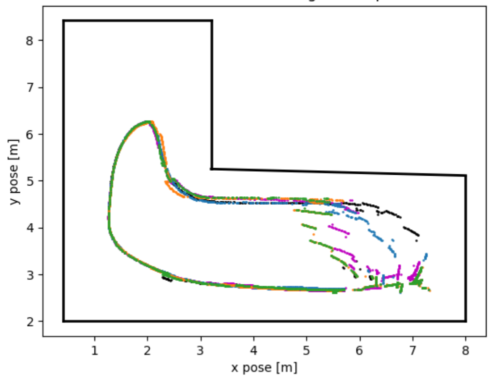




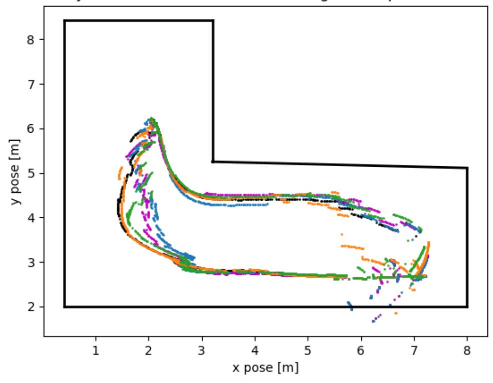




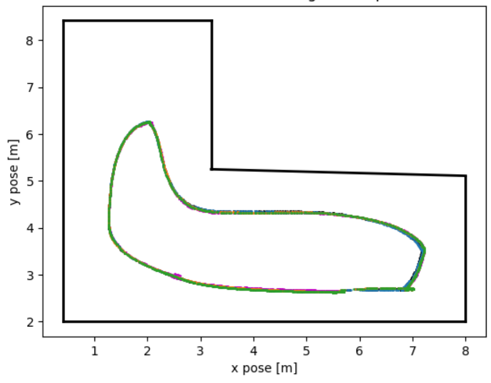
Precision static environment, with line algorithm, parameter index 20



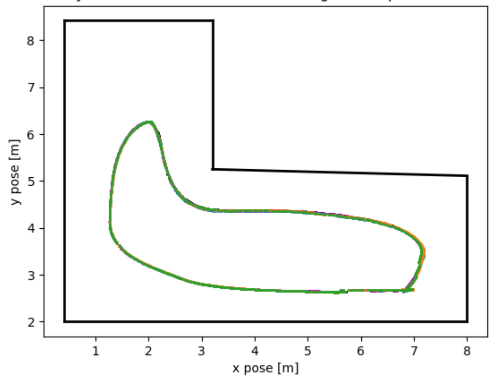
Precision dynamic environment, with line algorithm, parameter index 20



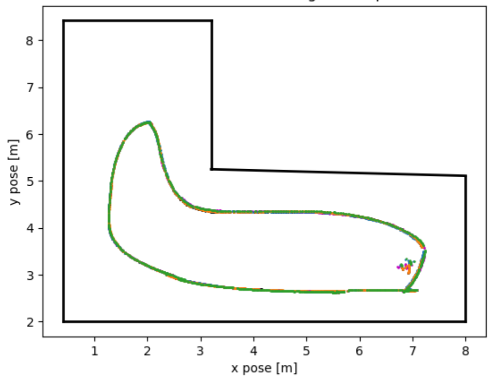
Precision static environment, with line algorithm, parameter index 21



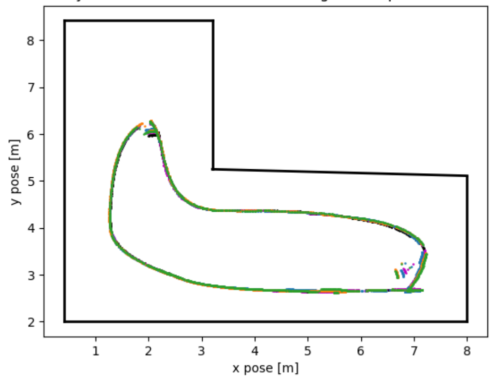
Precision dynamic environment, with line algorithm, parameter index 21



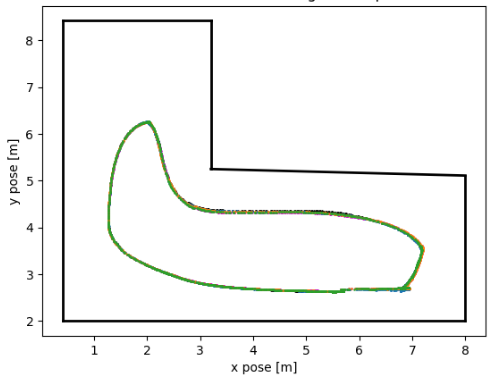
Precision static environment, with line algorithm, parameter index 22



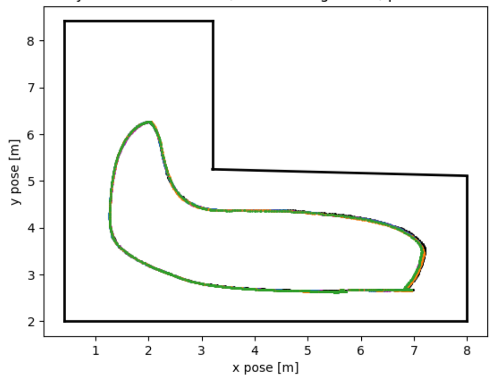
Precision dynamic environment, with line algorithm, parameter index 22



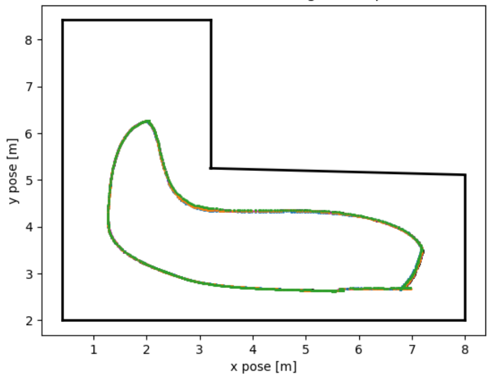
Precision static environment, with line algorithm, parameter index 23



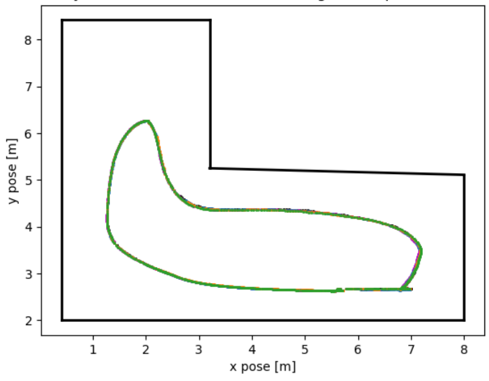
Precision dynamic environment, with line algorithm, parameter index 23



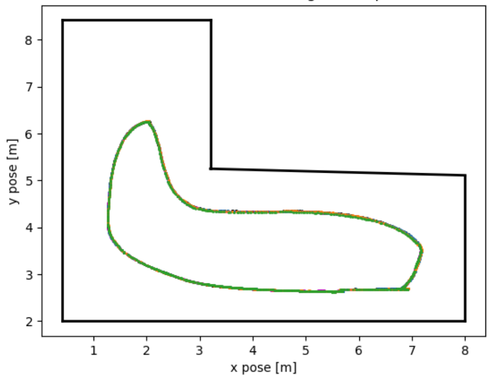
Precision static environment, with line algorithm, parameter index 30



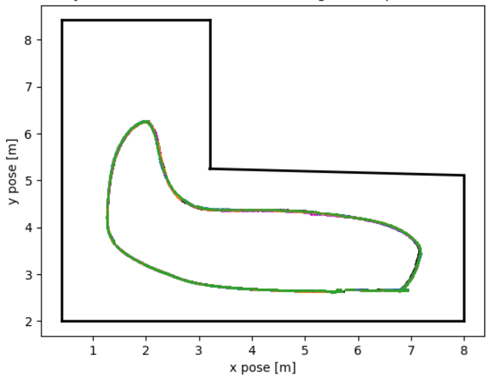
Precision dynamic environment, with line algorithm, parameter index 30



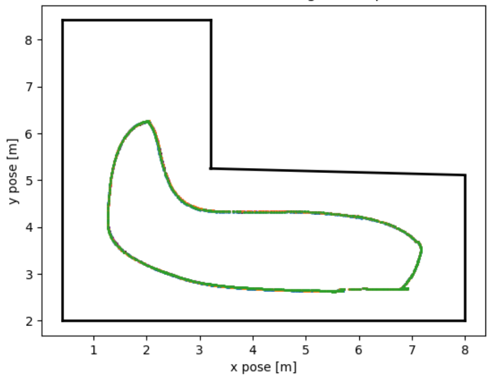
Precision static environment, with line algorithm, parameter index 31



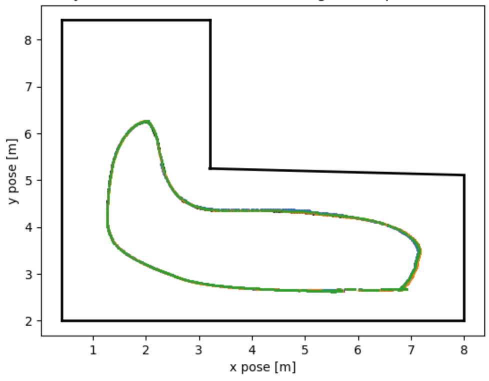
Precision dynamic environment, with line algorithm, parameter index 31



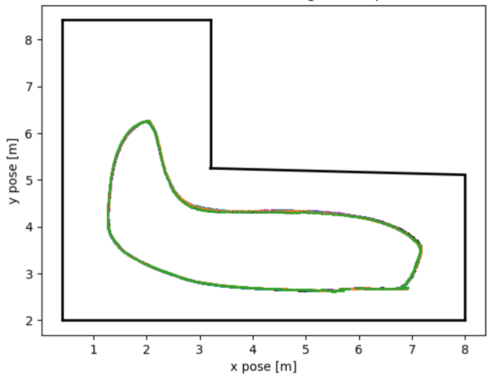
Precision static environment, with line algorithm, parameter index 32



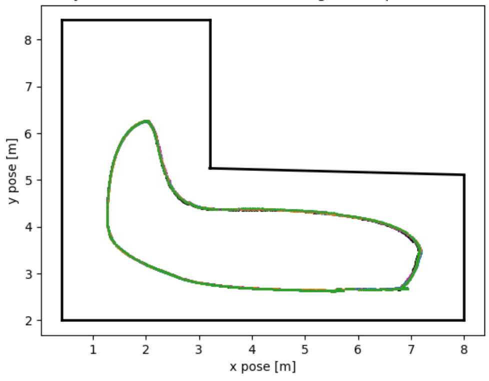
Precision dynamic environment, with line algorithm, parameter index 32

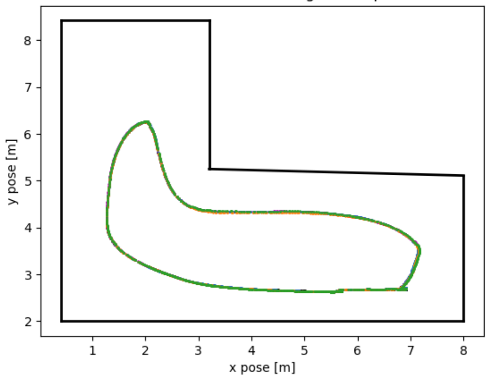


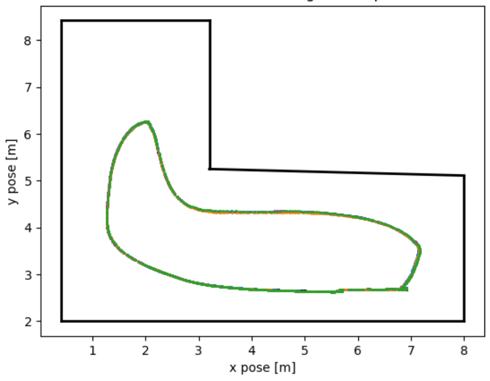
Precision static environment, with line algorithm, parameter index 40

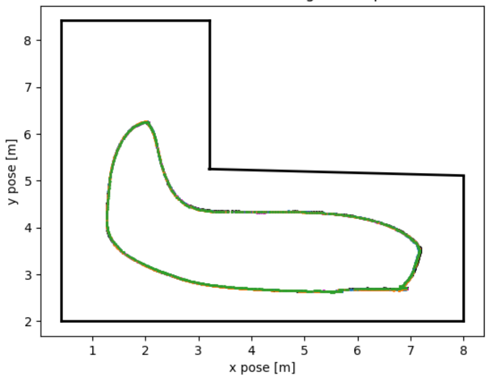


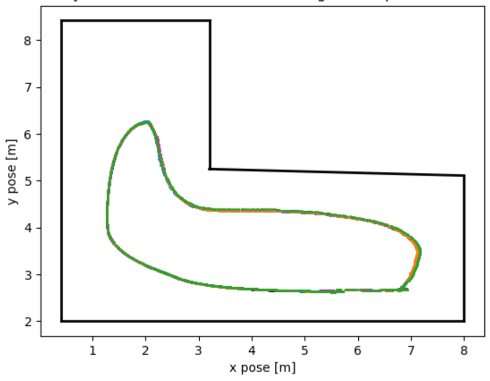
Precision dynamic environment, with line algorithm, parameter index 40

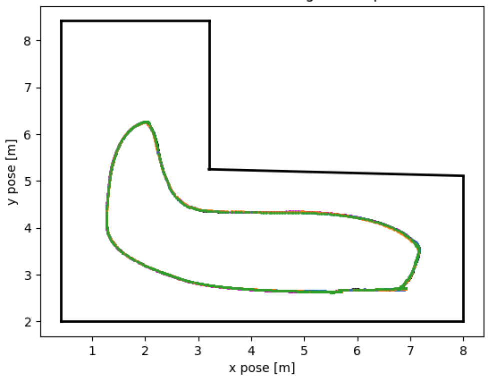


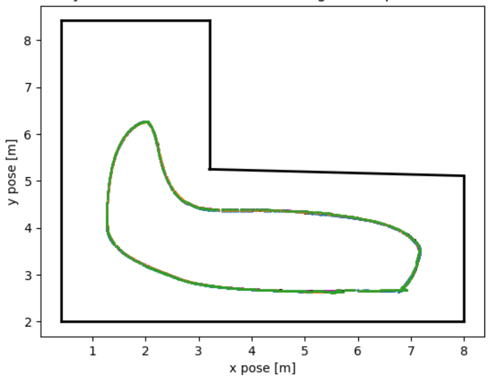
Precision static environment, with line algorithm, parameter index 41


Precision static environment, with line algorithm, parameter index 41


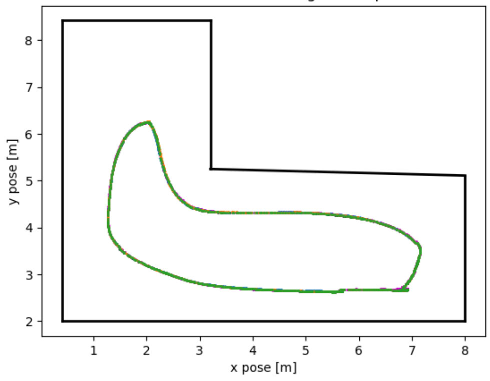
Precision static environment, with line algorithm, parameter index 42


Precision dynamic environment, with line algorithm, parameter index 42


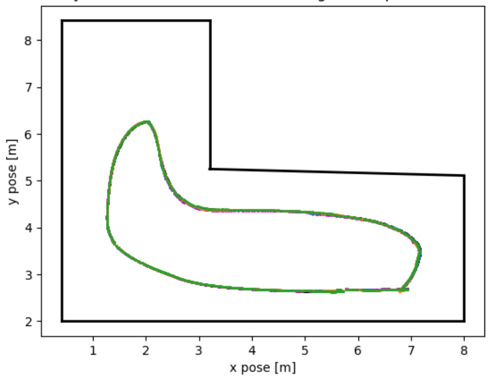
Precision static environment, with line algorithm, parameter index 43


Precision dynamic environment, with line algorithm, parameter index 43


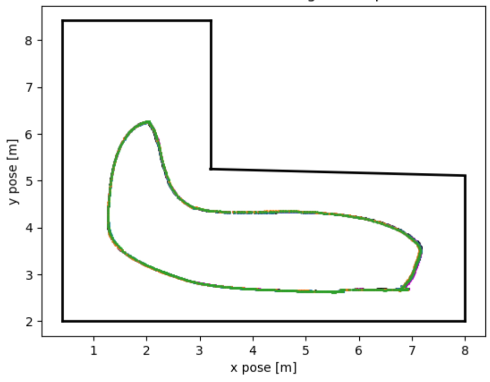
Precision static environment, with line algorithm, parameter index 50



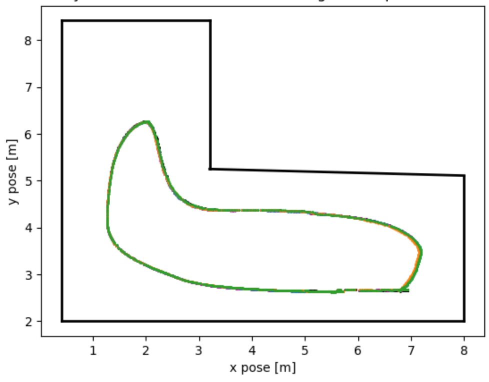
Precision dynamic environment, with line algorithm, parameter index 50



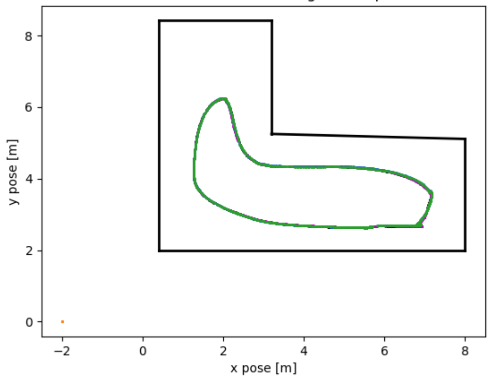
Precision static environment, with line algorithm, parameter index 51



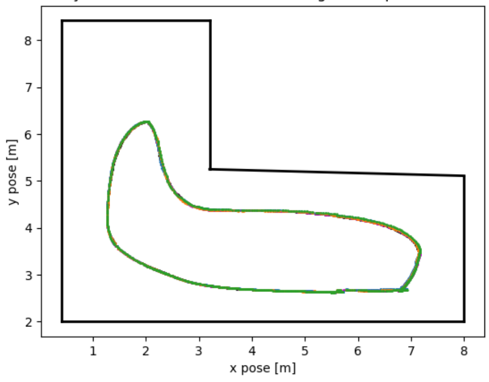
Precision dynamic environment, with line algorithm, parameter index 51



Precision static environment, with line algorithm, parameter index 52



Precision dynamic environment, with line algorithm, parameter index 52



F

Real world experiment result overview

	Barn route 1			Barn route 2			Barn route 3			Barn route 4		
Parameter set	**x [m]**	**y [m]**	**theta [rad]**	**x [m]**	**y [m]**	**theta [rad]**	**x [m]**	**y [m]**	**theta [rad]**	**x [m]**	**y [m]**	**theta [rad]**
No line algorithm	0.020	0.082	0.011	0.131	0.139	0.710	0.057	0.093	0.012	0.060	0.402	0.028
0	0.028	0.084	0.012	0.158	0.158	0.018	0.057	0.089	0.012	0.070	0.439	0.029
10	0.020	0.105	0.010	0.134	0.138	0.720	0.061	0.094	0.011	0.062	0.425	0.028
11	0.026	0.075	0.013	0.146	0.140	0.363	0.059	0.096	0.011	0.071	0.394	0.027
20	0.066	13.396	0.847	0.770	5.339	0.392	0.282	1.860	0.246	3.884	16.648	1.210
21	0.044	0.098	0.016	0.140	0.124	0.063	0.053	0.106	0.012	0.086	0.520	0.031
22	0.061	0.117	0.018	0.175	0.672	0.029	0.044	0.233	0.028	0.126	0.418	0.040
23	0.034	0.079	0.015	0.156	0.138	0.023	0.057	0.094	0.013	0.073	0.410	0.028
30	0.049	0.081	0.015	0.142	0.134	0.023	0.056	0.105	0.014	0.079	0.421	0.027
31	0.044	0.086	0.017	0.139	0.136	0.019	0.051	0.125	0.012	0.071	0.541	0.029
32	0.032	0.077	0.012	0.154	0.146	0.020	0.059	0.083	0.010	0.073	0.419	0.029
40	0.038	0.086	0.015	0.123	0.136	0.017	0.057	0.104	0.011	0.070	0.401	0.026
41	0.030	0.088	0.014	0.147	0.130	0.018	0.054	0.990	0.011	0.068	0.389	0.025
42	0.019	0.082	0.011	0.148	0.120	0.018	0.053	0.110	0.012	0.102	0.556	0.033
43	0.021	0.079	0.011	0.142	0.142	0.015	0.054	0.109	0.010	0.064	0.401	0.026
50	0.021	0.089	0.011	0.152	0.137	0.366	0.055	0.102	0.011	0.076	0.410	0.029
51	0.028	0.089	0.013	0.156	0.137	0.366	0.057	0.097	0.011	0.072	0.397	0.030
52	0.030	0.088	0.015	0.157	0.127	0.021	0.063	0.113	0.008	0.068	0.395	0.027

Table F.1: The average of the absolute localization errors compared to the GT in the barn. The parameters per parameter set can be found in appendix B.

Bibliography

[1] *Adaptive Monte Carlo localization*. https://roboticsknowledgebase.com/wiki/state-estimation/adaptive-monte-carlo-localization/. Feb. 2020.

[2] Eman Alhamdi and Ramdane Hedjar. “Comparative Study of Two Localization Approaches for Mobile Robots in an Indoor Environment”. In: *Journal of Robotics* 2022 (June 2022), pp. 1–13. DOI: 10.1155/2022/1999082.

[3] Deddy El Amin, Karlisa Priandana, and Medria Kusuma Dewi Hardhienata. “Development of Adaptive Line Tracking Breakpoint Detection Algorithm for Room Sensing using LiDAR Sensor”. In: *International Journal of Advanced Computer Science and Applications* 13 (7 2022). ISSN: 21565570. DOI: 10.14569/IJACSA.2022.0130732.

[4] Andrzej Pronobis, Kaiyu Zheng, Kousuke Ariga, Rajesh P. N. Rao. *COLD dataset*. https://www.coldb.org/site/overview. [Online; accessed 8 March, 2023]. 2023.

[5] Jan Bayer, Petr Čížek, and Jan Faigl. “On construction of a reliable ground truth for evaluation of visual SLAM algorithms”. In: *Acta Polytechnica CTU Proceedings* 6 (Nov. 2016), p. 1. DOI: 10.14311/APP.2016.6.0001.

[6] Filippo Bonaccorso, Francesco Catania, and Giovanni Muscato. “Evaluation of Algorithms for indoor mobile robot self-localization through laser range finders data”. In: *IFAC Proceedings Volumes* 43 (16 2010), pp. 563–568. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20100906-3-IT-2019.00097. URL: https://www.sciencedirect.com/science/article/pii/S1474667016351175.

[7] Filippo Bonaccorso, Francesco Catania, and Giovanni Muscato. “Evaluation of Algorithms for indoor mobile robot self-localization through laser range finders data”. In: *IFAC Proceedings Volumes* 43.16 (2010). 7th IFAC Symposium on Intelligent Autonomous Vehicles, pp. 563–568. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20100906-3-IT-2019.00097. URL: https://www.sciencedirect.com/science/article/pii/S1474667016351175.

[8] G A Borges and M . -J. Aldon. “A split-and-merge segmentation algorithm for line extraction in 2D range images”. In: vol. 1. 2000, 441–444 vol.1. ISBN: 1051-4651. DOI: 10.1109/ICPR.2000.905371.

[9] Geovany Araujo Borges and Marie-José Aldon. “Line Extraction in 2D Range Images for Mobile Robotics”. In: *Journal of Intelligent and Robotic Systems* 40 (3 2004), pp. 267–297. ISSN: 1573-0409. DOI: 10.1023/B:JINT.0000038945.55712.65. URL: https://doi.org/10.1023/B:JINT.0000038945.55712.65.

[10] M Bošnak. “Evolving principal component clustering for 2-D LIDAR data”. In: 2017, pp. 1–6. ISBN: 2473-4691. DOI: 10.1109/EAIS.2017.7954834.

[11] Matevž Bošnak. “Evolving principal component clustering for 2-D LIDAR data”. In: *2017 Evolving and Adaptive Intelligent Systems (EAIS)*. 2017, pp. 1–6. DOI: 10.1109/EAIS.2017.7954834.

[12] I. Bukhori, Z. H. Ismail, and T. Namerikawa. “Detection strategy for kidnapped robot problem in landmark-based map Monte Carlo Localization”. In: *2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*. 2015, pp. 75–80. DOI: 10.1109/IRIS.2015.7451590.

[13] Simone Ceriani et al. “Rawseeds ground truth collection systems for indoor self-localization and mapping”. In: *Autonomous Robots* 27 (4 2009), p. 353. ISSN: 1573-7527. DOI: 10.1007/s10514-009-9156-5. URL: https://doi.org/10.1007/s10514-009-9156-5.

[14] *Chi-Square Statistic: What It Is, Examples, How and When to Use the Test*. https://www.investopedia.com/terms/c/chi-square-statistic.asp. Accessed: 2023-07-25.

[15] R Domínguez et al. “LIDAR based perception solution for autonomous vehicles”. In: 2011, pp. 790–795. ISBN: 2164-7151. DOI: 10.1109/ISDA.2011.6121753.

[16] Richard O Duda and Peter E Hart. “Pattern classification and scene analysis”. In: 1974.

[17] Víctor J. Expósito Jiménez, Christian Schwarzl, and Helmut Martin. “Evaluation of an indoor localization system for a mobile robot”. In: *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*. 2019, pp. 1–5. DOI: 10.1109/ICCVE45908.2019.8965234.

[18] Jiang Xue Fei and Song Yu. “Conjugate Unscented Particle Filter Based Monte Carlo Localization for Mobile Robots”. In: *Applied Mechanics and Materials* 556-562 (2014), pp. 2266–2269.

[19] Jiang Xue Fei and Song Yu. “Conjugate Unscented Particle Filter Based Monte Carlo Localization for Mobile Robots”. In: *Applied Mechanics and Materials* 556-562 (2014), pp. 2266–2269.

[20] Giulio Fontana, Matteo Matteucci, and Domenico G Sorrenti. “Rawseeds: Building a Benchmarking Toolkit for Autonomous Robotics”. In: *Methods and Experimental Techniques in Computer Engineering* (2014). Ed. by Francesco Amigoni and Viola Schiaffonati, pp. 55–68. DOI: 10.1007/978-3-319-00272-9_4. URL: https://doi.org/10.1007/978-3-319-00272-9_4.

[21] Dieter Fox. “Adapting the Sample Size in Particle Filters Through KLD-Sampling”. In: *The International Journal of Robotics Research* 22 (12 Dec. 2003). doi: 10.1177/0278364903022012001, pp. 985–1003. ISSN: 0278-3649. DOI: 10.1177/0278364903022012001. URL: https://doi.org/10.1177/0278364903022012001.

[22] M Gallant. *Laser Line Extraction*. https://github.com/kam3k/laser_line_extraction. 2021.

[23] Haiming Gao et al. “A line segment extraction algorithm using laser data based on seeded region growing”. In: *International Journal of Advanced Robotic Systems* 15 (1 Jan. 2018). doi: 10.1177/1729881418755245, p. 1729881418755245. ISSN: 1729-8806. DOI: 10.1177/1729881418755245 URL: https://doi.org/10.1177/1729881418755245.

[24] Gengyu Ge et al. “Text-MCL: Autonomous Mobile Robot Localization in Similar Environment Using Text-Level Semantic Information”. In: *Machines* (2022).

[25] Chen Gu, Ahmed Shokry, and Moustafa Youssef. *The Effect of Ground Truth Accuracy on the Evaluation of Localization Systems*. 2021. arXiv: 2106.13614 [eess.SP].

[26] Shubh Gupta, Adyasha Mohanty, and Grace Gao. “Getting the Best of Particle and Kalman Filters: GNSS Sensor Fusion using Rao-Blackwellized Particle Filter”. In: (2022).

[27] International Society of Information Fusion et al. *Combining KLD-Sampling with Gmapping Proposal for Grid-Based Monte Carlo Localization of a Moving Robot*. 2017. ISBN: 9780996452700.

[28] S -W. Kang, S -H. Bae, and T -Y. Kuc. “Feature Extraction and Matching Algorithms to Improve Localization Accuracy for Mobile Robots”. In: 2020, pp. 991–994. ISBN: 2642-3901. DOI: 10.23919/ICCAS50221.2020.9268393.

[29] A R Khairuddin, M S Talib, and H Haron. “Review on simultaneous localization and mapping (SLAM)”. In: 2015, pp. 85–90. DOI: 10.1109/ICCSCE.2015.7482163.

[30] *Lely Lely presenteert visie Boerderij van de Toekomst 2035*. https://www.lely.com/be/nl/persberichten/2023/06/06/lely-presenteert-visie-boerderij-van-de-toekomst-2/. Accessed: 2023-08-09.

[31] Xinzhao Li et al. “A line segments extraction based undirected graph from 2D laser scans”. In: 2015, pp. 1–6. DOI: 10.1109/MMSP.2015.7340851.

[32] Yanjie Liu, Changsen Zhao, and Yanlong Wei. “A Robust Localization System Fusion Vision-CNN Relocalization and Progressive Scan Matching for Indoor Mobile Robots”. In: *Applied Sciences* 12.6 (2022). ISSN: 2076-3417. DOI: 10.3390/app12063007. URL: https://www.mdpi.com/2076-3417/12/6/3007.

[33] Yanjie Liu et al. “Improved LiDAR Localization Method for Mobile Robots Based on Multi-Sensing”. In: *Remote Sensing* 14 (23 Dec. 2022). ISSN: 20724292. DOI: 10.3390/rs14236133.

[34] Roberto Martín-Martín et al. “JRDB: A Dataset and Benchmark of Egocentric Robot Visual Perception of Humans in Built Environments”. In: (Oct. 2019). DOI: 10.1109/TPAMI.2021.3070543.

[35] Miguel Ángel de Miguel, Fernando García, and José María Armingol. "Improved LiDAR Probabilistic Localization for Autonomous Vehicles Using GNSS". In: *Sensors* 20.11 (2020). ISSN: 1424-8220. DOI: 10.3390/s20113145. URL: https://www.mdpi.com/1424-8220/20/11/3145.

[36] V T Nguyen et al. *A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics*. Sept. 2005, pp. 1929–1934. DOI: 10.1109/IROS.2005.1545234.

[37] Viet Nguyen et al. "A comparison of line extraction algorithms using 2D range data for indoor mobile robotics". In: *Autonomous Robots* 23 (2 2007), pp. 97–111. ISSN: 1573-7527. DOI: 10.1007/s10514-007-9034-y. URL: https://doi.org/10.1007/s10514-007-9034-y.

[38] David Obregón et al. "Adaptive Localization Configuration for Autonomous Scouting Robot in a Harsh Environment". In: *2020 European Navigation Conference (ENC)*. 2020, pp. 1–8. DOI: 10.23919/ENC48637.2020.9317366.

[39] M Ogaz, R Sandoval, and M Chacon. "Data processing from a Laser Range Finder sensor for the construction of geometric maps of an indoor environment". In: 2009, pp. 306–313. ISBN: 1558-3899. DOI: 10.1109/MWSCAS.2009.5236093.

[40] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. "Localization strategies for autonomous mobile robots: A review". In: *Journal of King Saud University - Computer and Information Sciences* 34 (8, Part B 2022), pp. 6019–6039. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2021.02.015. URL: https://www.sciencedirect.com/science/article/pii/S1319157821000550.

[41] T Pavlidis and S L Horowitz. "Segmentation of Plane Curves". In: *IEEE Transactions on Computers* C-23 (8 1974), pp. 860–870. ISSN: 1557-9956. DOI: 10.1109/T-C.1974.224041.

[42] David Portugal, André G. Araújo, and Micael S. Couceiro. "A Reliable Localization Architecture for Mobile Surveillance Robots". In: *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* (2020), pp. 374–379.

[43] Andrzej Pronobis and Barbara Caputo. "COLD: The CoSy localization database". In: *The International Journal of Robotics Research* 28 (May 2009). DOI: 10.1177/0278364909103912.

[44] Kun Qian et al. "Improved Rao-Blackwellized particle filter for simultaneous robot localization and person-tracking with single mobile sensor". In: *Journal of Control Theory and Applications* 9 (4 2011), pp. 472–478. ISSN: 1993-0623. DOI: 10.1007/s11768-011-9105-7. URL: https://doi.org/10.1007/s11768-011-9105-7.

[45] Abhijeet Ravankar et al. "On a Hopping-Points SVD and Hough Transform-Based Line Detection Algorithm for Robot Localization and Mapping". In: *International Journal of Advanced Robotic Systems* 13 (3 Jan. 2016). doi: 10.5772/63540, p. 98. ISSN: 1729-8806. DOI: 10.5772/63540. URL: https://doi.org/10.5772/63540.

[46] J Saarinen et al. "Normal distributions transform Monte-Carlo localization (NDT-MCL)". In: 2013, pp. 382–389. ISBN: 2153-0866. DOI: 10.1109/IROS.2013.6696380.

[47] J Saarinen et al. "Normal distributions transform Monte-Carlo localization (NDT-MCL)". In: 2013, pp. 382–389. ISBN: 2153-0866. DOI: 10.1109/IROS.2013.6696380.

[48] Viktor Schmuck and Oya Celiktutan. *RICA: Robocentric Indoor Crowd Analysis Dataset*. May 2020, pp. 63–65. DOI: 10.31256/Io1Sq2R.

[49] Ali Siadat et al. "An Optimized Segmentation Method for a 2D Laser-Scanner Applied to Mobile Robot Navigation". In: *IFAC Proceedings Volumes* 30 (7 1997), pp. 149–154. ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)43255-1. URL: https://www.sciencedirect.com/science/article/pii/S1474667017432551.

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

[51] Jürgen Sturm et al. "A benchmark for the evaluation of RGB-D SLAM systems". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 573–580. DOI: 10.1109/IROS.2012.6385773.

[52] Niko Sünderhauf et al. “Visual Odometry Using Sparse Bundle Adjustment on an Autonomous Outdoor Vehicle”. In: Jan. 2005, pp. 157–163. ISBN: 3-540-30291-3. DOI: 10.1007/3-540-30292-1_20.

[53] Jaroslaw Szrek et al. “Accuracy evaluation of selected mobile inspection robot localization techniques in a gnss-denied environment”. In: *Sensors (Switzerland)* 21 (1 Jan. 2021), pp. 1–23. ISSN: 14248220. DOI: 10.3390/s21010141.

[54] Qin Tang and Jing Liang. *Grid-Based Monte Carlo Localization for Mobile Wireless Sensor Networks*. Jan. 2020, pp. 1339–1346. ISBN: 978-981-13-6503-4. DOI: 10.1007/978-981-13-6504-1_159.

[55] J. Tardos. “Introduction to Mobile Robotics”. In: *Uni Freiburg DE* (2011), p. 9. URL: http://ais.informatik.uni-freiburg.de/teaching/ss09/robotics/slides/feature_extraction.pdf.

[56] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[57] *United Nations Our growing population*. https://www.un.org/en/global-issues/population. Accessed: 2023-08-09.

[58] Rafael Valencia et al. *Localization in highly dynamic environments using dual-timescale NDT-MCL*. 2014.

[59] J Vandorpe, H Van Brussel, and H Xu. “Exact dynamic map building for a mobile robot using geometrical primitives produced by a 2D range finder”. In: vol. 1. 1996, 901–908 vol.1. ISBN: 1050-4729. DOI: 10.1109/ROBOT.1996.503887.

[60] Regis Vincent, Benson Limketkai, and Michael Eriksen. “Comparison of indoor robot localization techniques in the absence of GPS”. In: ed. by Russell S. Harmon, Jr. John H. Holloway, and J. Thomas Broach. Apr. 2010, 76641Z. DOI: 10.1117/12.849593.

[61] E.A. Wan and R. Van Der Merwe. “The unscented Kalman filter for nonlinear estimation”. In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*. 2000, pp. 153–158. DOI: 10.1109/ASSPCC.2000.882463.

[62] Jibo Wang et al. “High-precision and robust localization system for mobile robots in complex and large-scale indoor scenes”. In: *International Journal of Advanced Robotic Systems* 18 (5 Sept. 2021). doi: 10.1177/17298814211047690, p. 17298814211047690. ISSN: 1729-8806. DOI: 10.1177/17298814211047690. URL: https://doi.org/10.1177/17298814211047690.

[63] Miin-Shen Yang and Yessica Nataliani. “Robust-learning fuzzy c-means clustering algorithm with unknown number of clusters”. In: *Pattern Recognition* 71 (2017), pp. 45–59. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.05.017. URL: https://www.sciencedirect.com/science/article/pii/S003132031730208X.