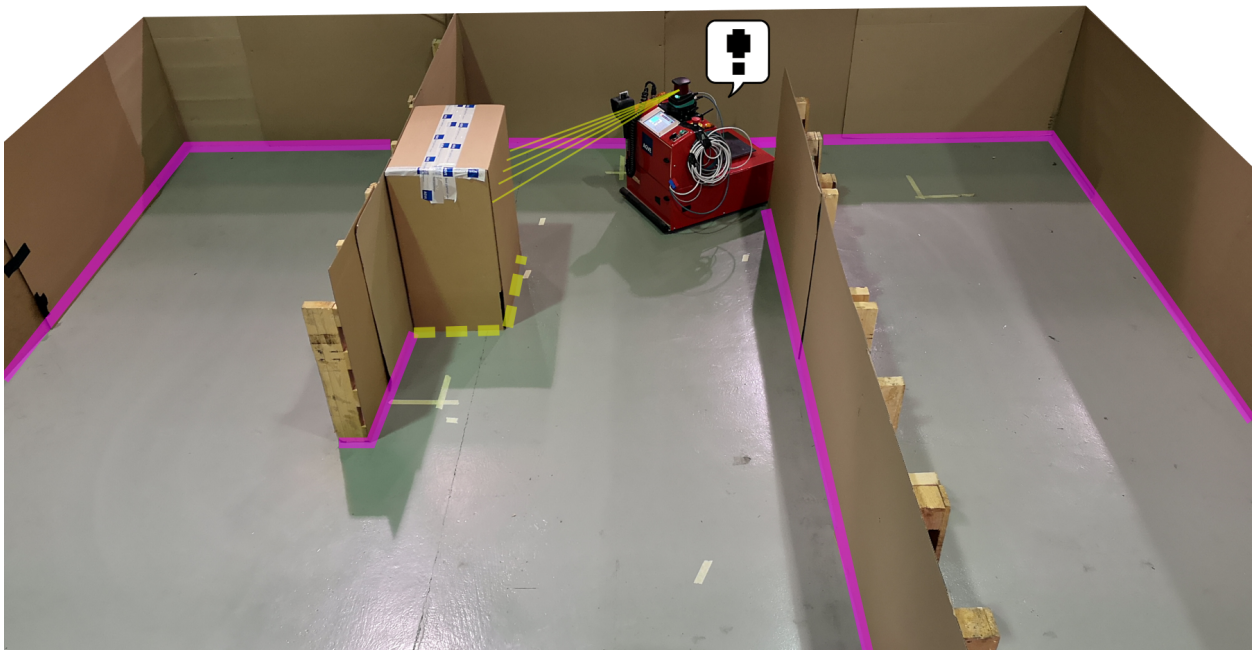
CHALMERS
UNIVERSITY OF TECHNOLOGY



Localizing in semi-static & cluttered environments with map updates

An implementation based on open-source libraries in ROS

Master's thesis in Systems, Control & Mechatronics

HANNES JUBRO KOOL

SIMON LILLSKOG

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Localizing in semi-static & cluttered environments with map updates

An implementation based on open-source libraries in ROS

HANNES JUBRO KOOL
SIMON LILLSKOG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Automation
Chalmers University of Technology
Gothenburg, Sweden 2019

Localizing in semi-static & cluttered environments with map updates
An implementation based on open-source libraries in ROS
HANNES JUBRO KOOL
SIMON LILLSKOG

Supervisor: Jonas Martner, AGVE Group
Examiner: Martin Fabian, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Division of Systems and Control
Automation
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 10 00

Cover: Illustration of the intended change detection and mapping functionality. The dashed yellow line represents a detected change to be added to the map, its borders represented by solid pink lines.

Typeset in LaTeX
Gothenburg, Sweden 2019

Localizing in semi-static & cluttered environments with map updates
An implementation based on open-source libraries in ROS
HANNES JUBRO KOOL
SIMON LILLSKOG
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Automated guided vehicles, AGVs, are a common element in many industrial facilities due to both increasing automation and to make the use of space more efficient. A typical application is transportation of material from point A to B autonomously, making navigation and localization a crucial part of its system. Current solutions use LiDAR sensors to navigate in a static map of the environment, filtering out moving objects before finding the best measurement-map match. In factories and other real applications, dynamic environments are inevitable, causing most solutions to fail over longer periods of time when relying on a static map.
With present research introducing the concept of lifelong mapping, this report aims to improve current algorithms by considering semi-static objects. That is, objects appearing static for shorter periods of time. The proposed algorithm is implemented in ROS, an open-source framework providing tools and libraries to aid the creation of complex robotic systems. The end result is an algorithm building upon well known particle filters and mapping algorithms to incorporate a way to detect changes in the environment and adding this new information to the most recent representation of the environment.
Our findings show that using the algorithm, an AGV can successfully localize in changed environments, where objects have been added or removed. It also shows that the algorithm works for cluttered environments, containing many small objects.

Keywords: AGV, SLAM, LiDAR, Localization, Semi-static, ROS, Hough Transform, Map Merging.

Acknowledgements

First of all we would like to thank AGVE for making this thesis possible and for providing us with both material and space to carry out our ideas. A special gratitude is directed to our supervisor Jonas Martner for his genuine interest and helpful guidance regarding everything from AGV systems to trail running. We also want to thank our examiner Martin Fabian for his insights on the thesis process. Finally we want to acknowledge our family and friends for their infinite support and patience during this period of time.

Hannes Jubro Kool & Simon Lillskog, Gothenburg, June 2019

Abbreviations

AGV Autonomous Guided Vehicle
AMCL Adaptive Monte Carlo Localization

BRF Beam Range Finder

CPU Central Processing Unit

GPU Graphics Processing Unit

LiDAR Light Detection and Ranging

OS Operating System

RADAR Radio Detection and Ranging
RAM Random-Access Memory
RANSAC Random Sample Consensus
ROS Robot Operating System

SLAM Simultaneous Localization and Mapping
SONAR Sound Navigation Ranging

UDP User Datagram Protocol

Contents

List of Figures

List of Tables

1
Introduction

Automated guided vehicles (AGVs) are a standard feature in many industrial facilities both due to an increasing demand for automation and to limit the usage of space. A typical application for these robots is transportation of materials from one point to another autonomously, making navigation and localization a crucial part of its system. There has been multiple solutions to navigate AGVs, with early methods consisting of paths of inductive wires or colored tape to magnetic grids. Today, laser rangefinders (LiDARs) have become a frequently used sensor for AGV navigation. The most common solution up-to-date is LiDAR tracking with reflective targets. These targets are fixed and when in the region of sight, they are used to triangulate the position of the AGV.

This is the current solution used by the AGV manufacturing company AGVE. The company wants to investigate a navigation solution that is independent of reflective targets with both cosmetic aspects and the elimination of difficult and costly assembling as basis. This type of navigation is commonly referred to as *natural navigation*, briefly explained as navigating without aid from specific objects located in the facility but instead scanning the environment and locating using these scans.

AGVE has implemented a functioning natural navigation algorithm using LiDAR and odometry as input data. However, it is limited to environments that contains many line features and that is uncluttered. The algorithm needs LiDAR measurements of walls and corners that have not been polluted with other objects blocking the laser beams. Therefore, there is a need for a navigation algorithm capable of localizing an AGV in a large and cluttered environment. Since cluttered environments, such as factories, tend to be dynamic, there is also a need for the algorithm to handle objects being moved in the vicinity of the AGV's route.

1.1 Purpose

The purpose of this project is to develop a prototype of a LiDAR based localization algorithm capable of localizing in a cluttered environment without pre-defined landmarks such as reflectors. Furthermore, the algorithm should be able to handle objects being added or removed in its environment. Dynamic objects should be filtered out while semi-static objects, that is, objects that are static for some period of time, should be used to improve the localization accuracy. The prototype should be evaluated on one of the standard AGVs owned by AGVE.

1.2 Scope

This project only focuses on localization, that is providing a position and heading in a two dimensional map while not considering route planning. The localization algorithm will be based on open-source software available in the Robot Operating System (ROS). It should be capable of localizing with the aid of all static objects in the environment, and semi-static objects such as pallets. The latter are static for shorter periods of time, enough to be able for them to be mapped.

The developed prototype will be tested on an AGV. However, the software will run on a separate computer and communicate with the AGV and LiDAR via Ethernet. No particular optimization of the software will be considered.

The evaluation of the prototype will be performed in an enclosed area of approximately 20 m^2.

1.3 Contribution

This project presents an approach to conveniently detect changes and, using a specific map management scheme, allowing mapping of the changes. Additionally, it presents a straightforward way to let a ROS-based system communicate with existing systems using Ethernet communication. This bypasses the issues of having to implement the algorithm directly onto an embedded system. It also presents a motion model used in the localization algorithm customized for tricycle AGVs.

1.4 Thesis Outline

The thesis is divided into the following chapters:

Chapter 2 introduces previous work related to the research topic and presents a variety of approaches regarding life-long navigation, which is allowing to navigate mobile robots without human supervision.

Chapter 3 provides the necessary theoretical background. This includes derivations of the probabilistic framework adapted to localization and navigation of robotic systems (SLAM), common mapping techniques to capture the state of the environment and the concept of map merging (here, a specific approach to cope with dynamic environments). It also sets the foundation on how to build robotic systems using ROS, an open-source framework to reduce the gap from idea to product when working with robotics systems.

Chapter 4 describes the methodology of implementing various algorithms as a solution to dynamic environments and defines the decisions made in order to achieve functionality.

Chapter 5 defines the testing setup for verification of the complete system.

Chapter 6 presents the results of the algorithm, both with and without accounting for changes.

Chapter 7 discusses the indications of the result and reflects on its reliability and performance before proposing future developments.

Chapter 8 summarizes the key findings and reflects on the results of the thesis.

2
Related Work

Research covering localization and mapping of mobile vehicles has been increasing in recent years. The idea of simultaneously localizing and mapping gained attention around the beginning of the 2000s and has since then been formulated in many ways, for various environmental settings.

Managing changes in the environment and allowing for life-long navigation is a major research topic as this is essential to autonomously navigate in real environments over longer periods of time.

A popular approach and probably the most straightforward is to detect and discard dynamic objects. This way the environment is reduced to a static form and standard SLAM algorithms can be utilized as usual. This works well for settings accommodated by highly dynamical objects such as moving cars or humans, however it does not manage semi-static objects (for example cars in a parking garage and pallets in a warehouse) well[3].

Biber & Duckett [4] provides an approach that represents the environment by multiple maps of different time-scales. The maps are composed of sets of samples from laser observations and by introducing different time-scale parameters, each map is able to forget previous information at a particular rate. When localizing, the time-scale with the best fit according to the perceptual model is chosen as a current state of the environment.

Tipaldi et. al [5] captures dynamic behaviour by introducing a transition model into the occupancy grid definition. Each cell is associated to a Hidden Markov Model (HMM) allowing for its state to change with respect to the transition model. With this extended occupancy grid, a Rao-Blackwellized particle filter [6] is further applied for the robot to locate itself in the map.

Meyer-Delius et. al [7] proposed a combination of temporary maps and a static map for localization in a semi-static environment. Organizing the temporary maps in a KD-tree allows to search for and trust the closest temporary map whenever the observations are not consistent with the static map. KD-trees are a graph structure of increasing dimensionality similar to how branches of a tree spread out. For this application, the poses corresponding to a temporary map is stored and the tree then allows efficient search for the nearest neighbour. A temporary map is initialized when the current observations cannot be explained by either the static map or previous temporary maps i.e. when the amount of outliers exceeds a certain threshold.

Shaik et. al [8] presents an approach that detects changes in the environment by continuously evaluating the probabilistic uncertainty of the model. Temporary maps are built whenever the current map representation does not coincide with the observed laser data to capture these changes. After building the temporary

map it is merged with the current representation, which is a combination of a pre-recorded static map and previous temporary maps to provide the most up-to-date representation. This report sets the foundation of techniques to be implemented in this project.

3

Theory

For a mobile robot to drive autonomously it needs to localize itself relative to its surroundings. This requires a way to gather information about the environment near the robot. There are many types of sensors that can achieve this, such as LiDARs, RADARs and SONARs. However, they all have one thing in common; the information they present has an inherent uncertainty. This uncertainty is due to many factors. The dynamic nature of the robot's environment results in unpredictability and uncertainty, limitations in sensors and sensor noise produces varying measurements and model errors and approximations of algorithms a fairly accurate but not always true description of reality [9].

The uncertainty can be managed by using the calculus of probability. Within robotics, this field has grown large during recent time and is known as *probabilistic robotics*. Summarized, instead of relying on single guesses, probability distributions are used to represent the current location of the robot, the received sensor data and all other information. Below, this theory is further explained, along with additional theory applied in this report.

3.1 Recursive Bayesian Estimation

The state of a mobile robot navigating in a two-dimensional environment is represented by two coordinates and a heading direction in some global coordinate system. Since the robot is unaware of its true state, the concept of *belief* is used to represent the currently estimated state. This is written $bel(\mathbf{x_t})$ where $\mathbf{x_t} = [x\ y\ \theta]$ is the current state. In probabilistic robotics the belief is represented by a conditional probability distribution

$$bel(\mathbf{x_t}) = p(\mathbf{x_t} \mid \mathbf{z_{1:t}}, \mathbf{u_{1:t}}) \tag{3.1}$$

which is a posterior of the state $\mathbf{x_t}$ conditioned on all past measurement data $\mathbf{z_{1:t}}$ and all past control inputs $\mathbf{u_{1:t}}$. In other words, given all available data this assigns a probability to all possible states (position and heading). Another name for equation (3.1) is *measurement update* since the current estimated state is assigned a probability using all available information [9].

Another important belief is called *prediction*, this is similar to the conditional probability above with the exception of not incorporating the latest measurement $\mathbf{z_t}$. This belief is written

$$\overline{bel}(\mathbf{x_t}) = p(\mathbf{x_t} \mid \mathbf{z_{1:t-1}}, \mathbf{u_{1:t}}) \tag{3.2}$$

and is called prediction since it predicts the state $\mathbf{x_t}$ only based on the previous measurement information $\mathbf{z_{1:t-1}}$.

For calculating beliefs a fundamental algorithm is *Bayes filter*. This is based on *Bayes rule*, which relates the two conditional probabilities $p(x|y)$ and $p(y|x)$

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)} \tag{3.3}$$

In the case of a mobile robot, x can be a position that is to be inferred from sensor data y. Bayes rule then states that the posterior $p(x|y)$ is calculated by multiplying the prior $p(x)$, which could be an estimated position, with the posterior $p(y|x)$, which is the probability of estimated position x measuring sensor data y. Since $p(y)$ is independent of x it is seen as a normalizer and Bays rule is then written

$$p(x \mid y) = \eta p(y \mid x)p(x) \tag{3.4}$$

In alg. 1 the Bayes Filter algorithm is written as pseudo code. This algorithm is applied recursively and consists of two steps. In the first one, the prediction step, the probability that the control input causes a transition from $\mathbf{x_{t-1}}$ to $\mathbf{x_t}$ is multiplied with the prior information of $\mathbf{x_{t-1}}$ and the product is integrated. In the second one, the measurement update, the probability of observing measurement $\mathbf{z_t}$ from state $\mathbf{x_t}$, $p\left(\mathbf{z_t}|\mathbf{x_t}\right)$, is multiplied with the belief $\overline{bel}(\mathbf{x_t})$ [9].

Algorithm 1: Bayes Filter

Input: $bel(\mathbf{x_{t-1}}),\ \mathbf{u_t},\ \mathbf{z_t}$
Output: $bel(\mathbf{x_t})$

forall $\mathbf{x}_t$ **do**
  $\overline{bel}\left(\mathbf{x_t}\right) = \int p\left(\mathbf{x_t} \mid \mathbf{u_t}, \mathbf{x_{t-1}}\right) bel\left(\mathbf{x_{t-1}}\right) d\mathbf{x_{t-1}}$
  $bel\left(\mathbf{x_t}\right) = \eta p\left(\mathbf{z_t} \mid \mathbf{x_t}\right) \overline{bel}\left(\mathbf{x_t}\right)$
end
return $bel(\mathbf{x_t})$

An important property of Bayes Filter is that it makes an *Markov assumption*. This means that if the current state $\mathbf{x_t}$ is known, past and future data are independent. However, there are multiple factors that causes violations of this assumption. Examples in mobile localization are errors when approximating belief functions and unmodeled dynamics in the environment. These violations could be avoided by having a more complex state representation where for example dynamics in the environment are included, however, the increased computational complexity this introduces in the Bayes Filter algorithm is not desired. It has been shown that in practice, if the unmodeled state variables cause random effects, the Bayes Filter algorithm is robust against the violations of the Markov assumptions [9].

3.2 Particle Filtering

There are different ways of implementing a Bayes filter. In the general case, there are no exact methods for calculating beliefs, they need to be approximated and can

be so using different types of posterior distributions.

One way is by using a *particle filter* or *sequential Monte Carlo* method. This is a non-parametric implementation of Bayes filter, as opposed to implementations where probability distributions described by parameters such as mean and variance are used (for example Gaussians). Particle filters approximate a distribution by drawing a set of samples from it and can therefore represent a very big number of distributions, including multi-modal distributions. The filter consists of a set of M particles where each particle is a hypothesized state, for instance position of a robot, at the current time. The larger number of particles is used, the more likely it is that the true state is represented by a particle. However, increased number of particles means a higher computational cost, therefore many implementations defines M as a function of the current variance of the filter. The simplest version of a particle filter is shown in alg. 2 [9].

Algorithm 2: Particle Filter

Input: $\chi_{\mathbf{t-1}},\ \mathbf{u_t},\ \mathbf{z_t}$
Output: $\chi_{\mathbf{t}}$

1 **for** $m = 1$ *to* M **do**
2 | *sample* $x_t^m \sim p(\mathbf{x_t}|\mathbf{u_t}, \mathbf{x_{t-1}}^m)$
3 | $\omega_t^m = p(\mathbf{z_t}|\mathbf{x_t}^m)$
4 | $\overline{\chi}_t = \overline{\chi}_t + < \mathbf{x_t}^m, \omega_t^m >$
5 **end**
6 **for** $m = 1$ *to* M **do**
7 | *draw* i *with probability* $\propto\ \omega_t^i$
8 | *add* x_t^i *to* χ_t
9 **end**
10 **return** χ_t

Here a set of M particles $\mathbf{x_{t-1}}$ in a set χ are input together with the control input $\mathbf{u_t}$ and measurement $\mathbf{z_t}$. When initializing the filter, the particles at time $t = 0$ are distributed according to a predefined distribution. In line 2, the hypothetical states x_t^m are generated from the state transition distribution $p(\mathbf{x_t}|\mathbf{u_t}, \mathbf{x_{t-1}})$ with the control input and previous particles as input. This will later be referred to as the *motion model* and this step corresponds to the prediction step in the Bayes Filter algorithm. In line 3, the measurements $\mathbf{z_t}$ are incorporated into the particles by for each particle calculate the probability of the measurement given the state of the particle and last control input. This probability is usually referred to as the *weight* of the particle. The distribution $p(\mathbf{z_t}|\mathbf{x_t})$ is later referred to as the *measurement model* and the step corresponds to the measurement update in the Bayes Filter algorithm. In line 6-9, the so-called resampling step is performed. M particles are drawn with replacement from the temporary set $\overline{\chi_t}$ with the probability of drawing each particle equal to the weight of the particle. This results in the particles being distributed according to the posterior $bel(\mathbf{x_t})$, while being distributed as $\overline{bel(\mathbf{x_t})}$ before the resampling. If resampling would not be performed, many particles would eventually have a negligible weight and not contribute to the approximation of the distribution.

By resampling the computational power is used in the region of importance in the state space [9].

3.2.1 Odometry Motion Model

An essential component of the particle filter is the motion of the robot. Given a model of possible motions, the filter can provide more accurate estimations by sampling particles from a defined probability distribution. Thrun et. al. [9] derives two common probabilistic motion models to define the distribution $p(\mathbf{x}_t|\mathbf{u}_t,\ \mathbf{x}_{t-1})$: the velocity model and the odometry model. The velocity model assumes that the robot can be controlled by a pair of velocity commands: a translational and a rotational velocity. The odometry model instead relies on measurements from moving sensors such as wheel encoders. Integrating the observed displacement then provides insight on how the robot has moved. Thrun et. al. further argues that the odometry motion model generally is the most accurate model. Both representations suffer from drift and slippage, however the velocity model also has a mismatch between the actual control signals and its mathematical model. Since the odometry model relies on motion sensors and odometry measurements are only available after the robot has moved, this model has a slight delay. It is also worth noting that even if odometry represent sensor measurements rather than control signals such as velocity commands, it is treated as control signals to avoid a larger state space.

The odometry provides relative motion in the interval $(t-1,\ t]$ thus defines the movement from pose $\bar{\mathbf{x}}_{t-1} = [\bar{x}\ \bar{y}\ \bar{\theta}]$ to pose $\bar{\mathbf{x}}_t = [\bar{x}'\ \bar{y}'\ \bar{\theta}']$ measured in the internal coordinate frame (indicated by the bar notation). The key assumption is that the difference between $\bar{\mathbf{x}}_{t-1}$ and $\bar{\mathbf{x}}_t$ is a good estimation of the true state difference x_{t-1} and x_t defined in the global coordinate frame. Further note that the model, as defined by Thrun et.al. describes a differential wheeled mobile robot with its pose originating from its center. A tricycle differential wheeled robot consists of two separate driving wheels rotating about the same axis. These wheels control both linear and rotational velocity, either by rotating at same or different speeds in either same or opposite directions. For this model, the remaining wheel is usually a caster wheel, allowing free rotations at the same time as it is supporting the structure. This specific differential tricycle model is embraced by both the Turtlebot 3 Burger (utilized in the simulation environment Gazebo) and the Pioneer 3DX (physical platform for initial testing).

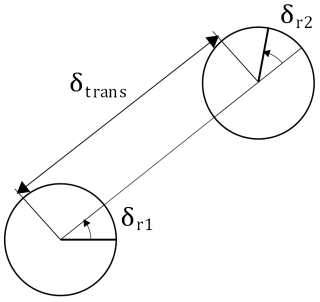


Figure 3.1: Illustration of the relative odometry in terms of the decomposed terms. Reproduced according to illustration in: *Probabilistic Robotics by Sebastian Thrun, Wolfram Burgard, and Dieter Fox, published by The MIT Press.*

To extract relative odometry between two poses, the displacement is decomposed into a sequence of a rotation δ_{r1}, a translation δ_{trans} and a second rotation δ_{r2}. The sequence is visualized in fig.3.1, the circles represents the robot and its poses obtained from the odometry measurements. The first rotation describes the angular change from the initial orientation to reach the coordinates $(\bar{x}', \bar{y}')$. The translation is derived as the linear displacement from $(\bar{x}, \bar{y})$ to the coordinate $(\bar{x}', \bar{y}')$. The second rotation represents the angle added to δ_{r1} in order to achieve the final orientation $\bar{\theta}'$. The mathematical expressions are presented as line 1-3 in alg.3.

Algorithm 3: Odometry Sampling Model

Input: $\mathbf{u}_t = \{\bar{\mathbf{x}}_{t-1}, \bar{\mathbf{x}}_t\},\ \mathbf{x}_{t-1} = [x, y, \theta]$
Output: $\mathbf{x}_t = [x', y', \theta']$

1 $\delta_t = \sqrt{(\bar{x}' - \bar{x})^2 + (\bar{y}' - \bar{y})^2}$
2 $\delta_{r1} = \texttt{atan2}\,(\bar{y}' - \bar{y},\ \bar{x}' - \bar{x}) - \theta$
3 $\delta_{r2} = \theta' - \theta - \theta_{r1}$

4 $\hat{\delta}_t = \delta_t - \texttt{sample}\,(\alpha_3\delta_t^2 + \alpha_4\delta_{r1}^2 + \alpha_4\delta_{r2}^2)$
5 $\hat{\delta}_{r1} = \delta_{r1} - \texttt{sample}\,(\alpha_1\delta_{r1}^2 + \alpha_2\delta_t^2)$
6 $\hat{\delta}_{r2} = \delta_{r2} - \texttt{sample}\,(\alpha_1\delta_{r2}^2 + \alpha_2\delta_t^2)$

7 $x' = x + \hat{\delta}_t \cos\left(\theta + \hat{\delta}_{r1}\right)$
8 $y' = y + \hat{\delta}_t \sin\left(\theta + \hat{\delta}_{r1}\right)$
9 $\theta' = \theta + \hat{\delta}_{r1} + \hat{\delta}_{r2}$

Alorithm reproduced from: *Probabilistic Robotics by Sebastian Thrun, Wolfram Burgard, and Dieter Fox, published by The MIT Press*

The control input $\mathbf{u}_t$ is a set of two pose estimates $\{\bar{\mathbf{x}}_{t-1},\ \bar{\mathbf{x}}_t\}$ with $\bar{\mathbf{x}}_{t-1} = \begin{bmatrix}\bar{x} & \bar{y} & \bar{\theta}\end{bmatrix}$ and $\bar{\mathbf{x}}_t = \begin{bmatrix}\bar{x}' & \bar{y}' & \bar{\theta}'\end{bmatrix}$.

In addition to the previously described decomposition of the odometry information, alg.3 also applies Gaussian noise, $\mathcal{N}(0, \sigma^2)$, to the components, assuming that

subtracting this noise represents the true pose $\mathbf{x}_t$. The parameters α_i define the accumulated error as a product of motion with the indices representing different cases of how the noise affect translation and rotation respectively. The function `sample` simply samples the distribution whose variance is defined by summing the products of the squared decomposed terms and their corresponding α.

3.2.2 Robot Perception - The Measurement Model

After the prediction step in the particle filter has been performed using the motion model, the measurement model is used to update the weights of the particles. The purpose of the measurement model is to describe how the measurements are generated in the physical world. If the model is accurate, the results will be better, however, it can often be very difficult to model a sensor correctly. The features of a laser range sensor (LiDAR) is for instance easy to describe, which can be based on time-of-flight measurements, but when it comes to the physical properties of light propagating in air and being reflected on an unknown surface, things become difficult to model. Instead, the advantage of probabilistic robotics having a non-deterministic part can be utilized to incorporate a simplified measurement model and model the uncertainties as sensor noise.

In order to model the generated measurements, the environment of the robot must be represented in a map. The details of map construction will be discussed in sec. 3.3, in short, a common way is to have a map consisting of a grid where each cell in the grid can either be occupied by an object, be free or have an unknown state. The map is denoted m and the density to be approximated by the measurement model is then $p(\mathbf{z_t}|\mathbf{x_t}, m)$.

The *Beam Range Finder model* (BRF) is a measurement model that can be used for laser range sensors. It consists of a mix of four densities with each one incorporating a type of measurement error. One corresponds to a correct measurement with added Gaussian noise from small measurement noise. This is modeled as a Gaussian with standard deviation σ_{hit} and is centered around the distance to the obstacle according to the map.

Another corresponds to a short measurement of an unexpected object (it is not in the map), for instance a person. This is modeled by an exponential distribution with parameter λ_{short}.

Another corresponds to a failed measurement where the result is the maximum allowable range value, for instance caused by a light-absorbing object, and this is modeled by a point-mass distribution at z_{max}.

The last one corresponds to a random, unexplainable measurements and is modeled by a uniform distribution.

The model written in pseudo code can be seen in alg. 4. For each particle with state x_t, it's weight is calculated by for each measured range z_t^k (each corresponding to a direction θ_k) in the measurement vector z_t, first calculating the true range z_t^{k*} for the predicted position with same direction according to the map by using ray casting. When using ray casting, the distance from the sensor to the nearest object in a given direction is calculated by propagating a simulated beam until an object is encountered. Then, the probability of z_t^k being one of the four measurement errors is

calculated. σ_{hit} is the standard deviation of the Gaussian measurement noise, λ_{short} is the parameter of the exponential distribution describing short measurements and η is a normalizer. Each calculated probability is then multiplied with a weighting factor and summed.

Algorithm 4: Beam Range Finder model

Input: x_t, z_t, m
Output: ω

$\omega = 1$
for $k = 1$ *to* n **do**
  $z_t^{k*} = ray\_casting(x_t, m, \theta_k)$
  $p_{hit}(z_t^k|x_t, m) = \eta \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{\frac{-1}{2}\frac{(z_t^k - z_t^{k*})^2}{\sigma_{hit}^2}}$
  if $z_t^k - z_t^{k*} \leq 0$ **then**
    $p_{short}(z_t^k|x_t, m) = \eta\lambda_{short}e^{-\lambda_{short}z_t^k}$
  if $z_t^k = z_{max}$ **then**
    $p_{max}(z_t^k|x_t, m) = 1$
  if $z_t^k < z_{max}$ **then**
    $p_{rand}(z_t^k|x_t, m) = \frac{1}{z_{max}}$
  $\mathrm{p} = \begin{bmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{bmatrix}^T \begin{bmatrix} p_{hit}(z_t^k|x_t, m) \\ p_{short}(z_t^k|x_t, m) \\ p_{max}(z_t^k|x_t, m) \\ p_{rand}(z_t^k|x_t, m) \end{bmatrix}$
  $\omega = \omega \cdot p$
end
return ω

A disadvantage with the BRF model is that $p(z_t^k|x_t, m)$ is highly discontinuous in x_t, that is, a small variation in position (especially in heading direction) can cause a vast change in detected range. In other words, the model lacks smoothness.

Another measurement model for laser range finders, which overcomes the problem with high discontinuity, is the *likelihood field range finder model*. This is an ad-hoc model that instead of using ray tracing, uses a nearest neighbor function that for the endpoints of the laser scan finds the nearest obstacle. Similarly to the BRF model, the likelihood based model is a mixture of three densities with each one incorporating a type of measurement error. The first one corresponds to measurement noise. If *dist* denotes the Euclidean distance between the coordinate of the measurement point (or end point) of a laser beam and the nearest obstacle in the map, then the probability of the laser beam hitting an obstacle in the map is given by a zero-centered Gaussian with standard deviation σ_{hit}:

$$p_{\mathrm{hit}}\left(z_t^k|x_t, m\right) = \varepsilon_{\sigma_{\mathrm{hit}}}(dist) \tag{3.5}$$

The second density corresponds to a measurement failure, where the max range of the sensor has been returned. This is modeled similar to the beam based model

as a point-mass distribution at z_{max}. The last density corresponds to a random, unexplainable measurement and is modeled as a uniform distribution, also similar to the beam based model. The model written in pseudo code can be seen in alg. 5. It can be seen that this model ignores max range readings. Furthermore, in line 5 and 6 the global coordinate of the measurement point is calculated. Then in line 7 the distance to the closest obstacle is calculated, and in line 8 the weight of the particle is calculated by mixing a normal and a uniform distribution. By using the Euclidean distance instead of ray tracing, small changes in a robot's state (position and heading) will not have a significant result on the distribution $p(z_t^k|x_t, m)$, which can make it a more suitable measurement model for cluttered environments.

Algorithm 5: Likelihood field range finder model

Input: x_t, z_t, m
Output: ω

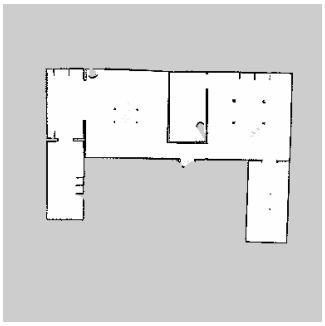
1 $\omega = 1$
2 **for** *all* k *do* **do**
3 **if** $z_t^k \neq z_{max}$ **then**
4 $p_{short}(z_t^k|x_t, m) = \eta\lambda_{short}e^{-\lambda_{short}z_t^k}$
5 $x_{z_t^k} = x + x_{k,\text{ sens}}\ \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{ sens}})$
6 $y_{z_t^k} = y + y_{k,\text{ sens}}\ \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{ sens}})$
7 $dist = \min_{x',y'} \left\{ \sqrt{\left(x_{z_t^k} - x'\right)^2 + \left(y_{z_t^k} - y'\right)^2} \,|\, \langle x', y' \rangle \text{ occupied in } m \right\}$
8 $\omega = \omega \cdot \left(z_{\text{hit}} \cdot \textbf{prob}\left(dist, \sigma_{\text{hit}}\right) + \frac{z_{\text{random}}}{z_{\text{max}}} \right)$
9 **end**
10 **return** ω

3.3 Environment Mapping

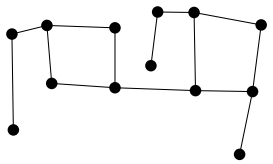
Maps play a central role in navigating mobile robots. Given that it is a correct representation of the surrounding environment it allows the robot to operate with high precision, using its sensors to localize itself on the map. However in most real applications, maps are built online, thus requiring a correct pose in order to produce a valid map. The problem with both mapping and localization depending on each other leads to what is referred to as SLAM.

3.3.1 Map Types

If solely focusing on mapping and assuming that the robot pose is known, describing the environment spatially can be split up into two classes: topological and metric.



(a) Metric grid map



(b) Topological map

Figure 3.2: Illustration of the differences in metric grid and topological mapping

Topological maps describe the environment by connecting significant locations (features) into a graph-like representation, nodes representing the features and edges the relationship between certain features. This is a compact representation of the environment as the size only depends on the complexity of the environment. This approach does however result in limited accuracy and also struggles when different features appear similar to the sensors [1].

Advantages	Disadvantages
Sparse representation	Allows less accuracy
Does not require accurate pose estimation	Difficult to build and maintain for larger environments
Convenient for path planning	Risk of mixing up features (incorrect data association)

Table 3.1: Advantages and disadvantages of topological maps [1],[2]

Metric maps instead relies on surrounding objects or obstacles at coordinates defined using sensor data such as odometry and LiDAR measurements. Contrary to topological maps, the maps produced by metric approaches are much denser and thus more computationally heavy to process. A common application of metric maps are grid maps, where the environment is discretized into cells of a defined resolution [m/cell]. Using sensor data, each cell is then classified as one of three states: *free*, *occupied* or *unknown*. This approach combined with and fairly low resolution allows for accurate mapping of the environment [1].

Advantages	Disadvantages
Easy to build and maintain	High dimensional space
Dense representation (small measurement loss)	Performance dependent on grid resolution
Allows high accuracy for small resolutions	Requires accurate pose estimations

Table 3.2: Advantages and disadvantages of grid maps [1],[2]

3.3.2 Probabilistic Occupancy Grid Mapping

Estimating maps is similar to estimating the pose of a moving robot. The current measurements and past information possesses uncertainty due to noise and filtering is necessary to provide a valid and robust representation of the environment. As introduced by Moravec and Elfes[10], the probabilistic approach of Bayes filtering (sec. 3.1) is applied for incremental learning of an occupancy grid. In other words, the consecutive measurements affecting the state of a cell are summed in a probabilistic fashion. Thus, if several measurements points to a certain cell being occupied, the probability of this being true is simply increased.

The limitations of occupancy grids has already been mentioned. The discretized view of the environment is high dimensional and in theory hold interdependencies between cells. In other words if a cell appear occupied, nearby cells would consequently achieve a higher probability of being occupied. However in practice, estimating these distributions would lead to an intractable problem. By instead treating each cell as conditionally independent, the dimensionality is limited and the problem becomes tractable. Since a single observation from a range finder sensor in reality spans over and affects several cells, this assumption is not reasonable. However it is well documented and provides satisfactory expressions. Forthcoming derivations will therefore focus on the individual distributions corresponding to cell $\mathbf{m}(i,j)$. It is also assumed that each cell is a binary random variable corresponding to either a *free* or *occupied* state. This allows for using the special case of conditional probability

$$p(\neg x \mid y) = 1 - p(x \mid y) \tag{3.6}$$

The sought distribution $p(\mathbf{m}(i,j)\mathbf{z}_{1:t},\ \mathbf{x}_{1:t})$ describes the current cell state given past and current measurements. The derivation below represents a compact version of the complete derivation in [11]. First the expression is extended using the *Bayes rule* (3.3) and further simplified to eliminate probabilities difficult to calculate

$$\begin{aligned} p(\mathbf{m}(i,j) \mid \mathbf{z}_{1:t},\ \mathbf{x}_{1:t}) &= \frac{p(\mathbf{z}_t \mid \mathbf{m}(i,j),\ \mathbf{x}_{1:t})p(\mathbf{m}(i,j) \mid \mathbf{z}_{1:t-1},\ \mathbf{x}_{1:t})}{p(\mathbf{z}_t \mid \mathbf{z}_{1:t-1},\ \mathbf{x}_{1:t})} = \\ &= \frac{p(\mathbf{m}(i,j) \mid \mathbf{z}_t,\ \mathbf{x}_t)p(\mathbf{z}_t \mid \mathbf{x}_t)p(\mathbf{m}(i,j) \mid \mathbf{z}_{1:t},\ \mathbf{x}_{1:t-1})}{p(\mathbf{z}_t \mid \mathbf{z}_{1:t-1},\ \mathbf{x}_{1:t})} \end{aligned} \tag{3.7}$$

The counterpart to the expression in (3.7), namely the probability of a cell being *free*, $\neg\mathbf{m}(i,j)$, is analogous to the same expression. In order to conveniently update the cells, the *log-odds* representation is used. It applies the *odds-ratio* in (3.8) to (3.7) and its counterpart before utilizing logarithmic rules to obtain the final update rule defined in (3.11).

$$odds(x) = \frac{p(x)}{p(\neg x)} \tag{3.8}$$

$$\begin{aligned} &\frac{p(\mathbf{m}(i,j) \mid \mathbf{z}_{1:t},\ \mathbf{x}_{1:t})}{p(\neg\mathbf{m}(i,j) \mid \mathbf{z}_{1:t},\ \mathbf{x}_{1:t})} = \\ &= \frac{p(\mathbf{m}(i,j) \mid \mathbf{z}_t,\ \mathbf{x}_t)p(\mathbf{m}(i,j) \mid \mathbf{z}_{1:t-1},\ \mathbf{x}_{1:t-1})p(\neg\mathbf{m}(i,j)}{p(\neg\mathbf{m}(i,j) \mid \mathbf{z}_t,\ \mathbf{x}_t)p(\neg\mathbf{m}(i,j) \mid \mathbf{z}_{1:t-1},\ \mathbf{x}_{1:t-1})p(\mathbf{m}(i,j))} \end{aligned} \tag{3.9}$$

$$\begin{aligned} \underbrace{\ln\frac{p(\mathbf{m}(i,j) \mid \mathbf{z}_{1:t},\ \mathbf{x}_{1:t})}{p(\neg\mathbf{m}(i,j) \mid \mathbf{z}_{1:t},\ \mathbf{x}_{1:t})}}_{l_t} &= \underbrace{\ln\frac{p(\mathbf{m}(i,j) \mid \mathbf{z}_t,\ \mathbf{x}_t)}{p(\neg\mathbf{m}(i,j) \mid \mathbf{z}_t,\ \mathbf{x}_t)}}_{l_{meas,t}} + \\ &+ \underbrace{\ln\frac{p(\mathbf{m}(i,j) \mid \mathbf{z}_{1:t-1},\ \mathbf{x}_{1:t-1})}{p(\neg\mathbf{m}(i,j) \mid \mathbf{z}_{1:t-1},\ \mathbf{x}_{1:t-1})}}_{l_{t-1}} - \underbrace{\ln\frac{p(\mathbf{m}(i,j))}{p(\neg\mathbf{m}(i,j))}}_{l_0} \end{aligned} \tag{3.10}$$

$$l_t = l_{meas,t} + l_{t-1} - l_0 \tag{3.11}$$

By rewriting the expression in (3.10) with respect to the conditional expression in (3.6), the only terms necessary for the update rule in (3.11) is the *inverse sensor model* $p(\mathbf{m}(i,j)|z_t,\ x_t)$ and the $p(\mathbf{m}(i,j)$ representing the prior information about the grid.

3.4 Map Merging

The idea of map merging originates from multi-robot exploration where each robot maintains a local submap of the environment. Instead of individual robots computing their own global map, information is shared cooperatively by combining all local maps into one. This is not only an efficient approach to globally map the surroundings but it also allows robots to gain access to local information without being present. For path planning this could be observing a future road block and re-routing the path. As proposed by Shaik et. al [8] map merging can also be a decisive component for coping with dynamic changes. By detecting changes and storing them as temporary maps, map merging is here used to combine the static representation with the observed changes to produce a current view of the environment. For this particular problem it instead becomes a matter of fine tuning the transformation between the maps as noise and negative effects influence the maps.

To further simplify the merging algorithm it is also assumed that the following assumptions hold

- Both maps are of equal size in pixels
- The cell resolution is equal i.e. each cell represents the same real world area
- The sought transformation is close to the previous estimate

A map merging algorithm can be split up into two major parts: finding the transformation between a pair of maps and combining them according to the same transformation.

3.4.1 Transformation

Given the assumptions in the previous section, the transformation sought is an Euclidean transformation including both rotational and translational terms and derived as followed for a 2D case [12]

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t} \tag{3.12}$$

or in matrix form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.13}$$

In (3.13), x and y are the coordinates to be transformed. θ corresponds to a counterclockwise rotation, while t_x and t_y to translations in respective directions. The prime notation indicates transformed coordinates.

The most common approach finding the transformation between two maps is by first converting them to images. Using techniques from the field of Image Processing is then applied to identify image features in the associated images before matching their correspondences. Image features appear in different forms: points, edges, lines, corners or other shapes. Not to confuse with features in topological maps (specific objects at a specific pose), image features present significant subsets of the image and its poses. This significance is typically extracted by different filtering and gradient techniques such as ORB[13] or SIFT[12]. Assuming that features extracted from the image possesses pairwise-unique properties, it can be matched to the corresponding feature in the other image. Solving for either all matches or using a sampling method such as RANSAC, an approximate transformation can be computed. A few solutions using the previously mentioned techniques exists, however in [14],[15] it is argued that these techniques are best suited for rich images (as obtained from cameras) as they contain far more distinctive features than occupancy grids. Instead the Hough Transform for extracting line features is proposed.

The Hough transform as proposed by Duda & Hart [16] is a technique to let edge points vote for reasonable lines described by the Hesse normal form

$$\rho(\theta) = x\cos\theta + y\sin\theta \tag{3.14}$$

here, ρ is the perpendicular distance from the image origin to the center of the line while θ defines the direction of this line. The parametrization is further illustrated by fig. 3.3.

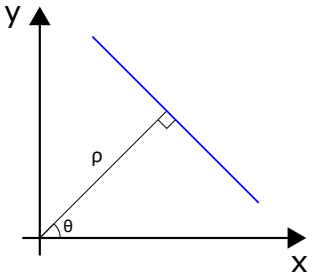


Figure 3.3: A straight line (blue) represented by ρ and θ

Each point votes for possible lines (3.14) by sweeping over a discrete interval of θ $[-\pi/2,\ \pi/2]$, forming a sinusoid in the $\rho - \theta$ space. Exploring all points and their parametrizations while incrementing the accumulator array for each corresponding $\rho - \theta$ pair results in a number of sinusoids with local maxima at their intersections. A maximum indicates the most probable $\rho - \theta$ pair for a certain line in the input space. Fig. 3.4 illustrates a simple input image of two crossing lines resulting in two distinct maxima in the Hough space, located at approximately $(45, 0)$ and $(-45, 175)$.

Advantages	Disadvantages
Easy implementation	Grows with image dimensions
Robust against noise and line gaps	Dense representation of lines

Table 3.3: Advantages and disadvantages of the Hough Transform

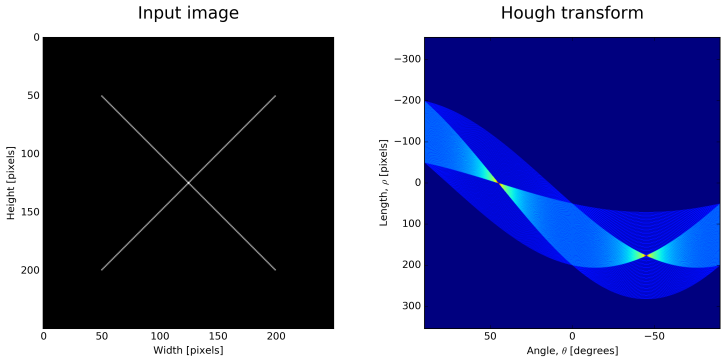


Figure 3.4: Simple example of line detection using the Hough transform

Censi et. al [17] managed to utilize the Hough transform for scan matching (matching consecutive LiDAR observations) before developed to occupancy grid merging by Carpin [15]. They achieve this by first introducing the *Hough spectrum* (3.15), a one-dimensional way to describe the Hough space. By squaring the elements and summing over the angles, the Hough spectrum indicates which directions is more frequent among the lines previously detected [15].

$$\mathbf{HS}(i) = \sum_{j=1}^{\rho_s} \mathbf{HT}(j,i)^2 \quad \forall i \in [1, \theta_s] \tag{3.15}$$

The Hough spectrum also holds a few important properties

- It is invariant to input translations
- An input rotation corresponds to a circular shift of the spectrum
- It is π-periodic

The properties makes the Hough spectrum a suitable choice for extracting rotational shifts from two inputs. As the spectral signals are both one-dimensional and periodical their similarities are further extracted by computing the *circular cross-correlation*

$$\mathbf{XC}_{HS}(i) = \sum_{j=1}^{\theta_s} \mathbf{HS}_1(i)\mathbf{HS}_2(i+j) \quad \forall i \in [1, \rho_s] \tag{3.16}$$

The subscripts in eq.(3.16) indicate the Hough spectra corresponding to a specific Hough space. Once correlated, the signal is the examined for local peaks, defining potential rotational shifts. Fig.3.5 provides an example utilizing the described procedure on two straight lines in contrasting directions.

To cope with relative translations, Carpin furthermore introduces a second spectral structure referred to as the X/Y-spectrum. Argued to be a fast approach to extract translational hypotheses, it is defined as

$$\mathbf{SX}(j) = \begin{cases} \sum_{i=1}^{r} \mathbf{m}(i,j) & 1 \leq j \leq c \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

$$\mathbf{SY}(i) = \begin{cases} \sum_{j=1}^{c} \mathbf{m}(i,j) & 1 \leq i \leq r \\ 0 & \text{otherwise} \end{cases} \tag{3.18}$$

In (3.17)-(3.18) the variable $\mathbf{m}$ represents an occupancy grid with $\mathbf{m}(i,j)$ pointing the the value of cell (i,j). Similar to the computation of the Hough spectrum, the X/Y-spectrum is the sum over the grid elements of each row/column. After obtaining the spectral information of two images it is correlated to find the shift corresponding to the maximum similarity using the global cross-correlation formula (analogous to Y)

$$\mathbf{XC}_{XS}(\tau) = \sum_{k=-\infty}^{+\infty} \mathbf{SX}_1(k+\tau)\mathbf{SX}_2(k) \tag{3.19}$$

When a suitable transformation is obtained the next step is to combine the images. This procedure and the implementation of the previous expressions are further presented in sec. 4.3.1.

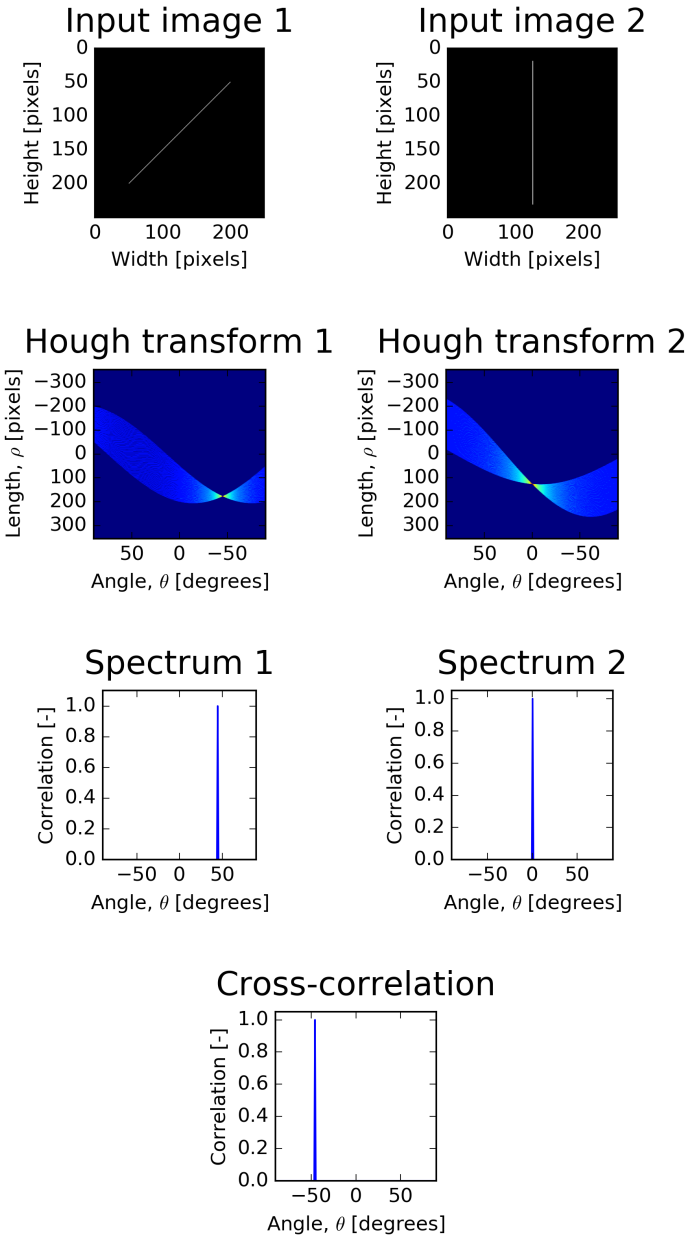


Figure 3.5: Illustration of the steps involved in extracting and correlating the spectra for a straight line and its rotated equivalence

3.5 Robot Operating System

Robot Operating System, ROS, is an open-source framework for writing code to support robotic systems. ROS was created with the intent of allowing to develop new ideas or systems without having to build complete and complex robotic systems from the ground up and it includes libraries and tools to seamlessly build on top of or combine existing solutions. The software providing specific functionality or even complete systems are referred to as packages and the development is driven by the

ROS community.

The structure can conveniently be described as graph trees, where nodes represents a computing process and the topics streams of data being passed between nodes. Fig.3.6 illustrates a simple scenario of a node subscribing to a topic being published by another node, however it is also possible to have a node both publish and subscribe i.e. when sequentially reading and processing data. The communication being carried out through topics consists of messages with defined data types for a specific topic and the transport of data is based on TCP/UDP protocols. This publisher-subscriber model is very convenient when handling continuous data streams such as motion data from sensors, however for certain scenarios a request-reply model is more suitable. In ROS, this two-way interaction is called a service and instead of one message being sent, a service is defined by a pair of messages: one for the request and one for the reply. A node offers a service that is initiated by the request message of another node and when the process is done, the reply message is sent back to the requesting node. The complete system of nodes is then managed by the ROS master, making sure that correct nodes are connected.

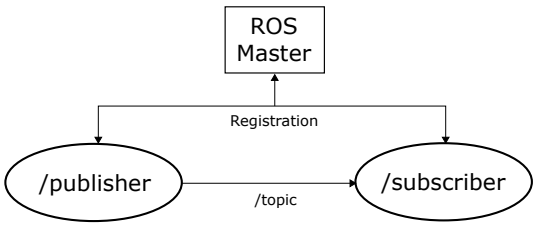


Figure 3.6: Illustration of the communications structure in ROS

Apart from providing an overall structure and various libraries, ROS also provides tools to analyze the system at run-time. It allows for introspection i.e. all data that is flowing under the hood can be presented and visualized. This can appear in multiple forms, from logging data streams in the terminal or plotting scalar data to form a visual graph of all processes and their connections. Together with the ability of getting access to every part of the system through the command line, robotic systems can efficiently be debugged and maintained using the ROS framework.

4
Software Implementation

In this project, open-source software from the ROS library was modified and used. New software was also written based on the theory in chapter 3. In the following sections, the different parts of the algorithm will be described and motivated.

4.1 Localization and Mapping

Today, there are many techniques to localize a mobile robot on a static map, as well as simultaneously localizing and building a map without a given map, referred to as SLAM (simultaneous localization and mapping). SLAM, however, suffers from the life-long SLAM problem that means that over time, more and more errors can be introduced during the mapping and as a result also in localization[18], [19]. To overcome this, it was decided to base the mapping on a static, pre-recorded map that only contains objects that will never be moved. This can be further motivated by the fact that in the industry, maps of the environment in which the AGVs operate are often used to create the routes for the AGVs and these maps can be used to create a static map for the localization. Still, to solve the problem of having semi-dynamic objects in the environment, it must be possible to modify this static map at certain times, which will be discussed later on. For localization with a LiDAR, there are two different approaches as mentioned in chapter 3.3.1. Either the map is based on a number of feature that is recurring in the mobile robot's environment, such as walls and pillars. In the other approach, occupancy grid mapping, no consideration to the objects in the environment has the be taken, the map only tells whether a space is occupied or free. Since the goal for this project is to navigate in a cluttered environment, features can be difficult to distinguish, hence the choice was to use the occupancy grid approach.

For localizing in static maps, two main approaches for approximating the state probability densities exist. Either Gaussian filters are used, such as the extended Kalman filter and unscented Kalman filter, or a particle based filter. Studies have shown that particle filters are far better at handling non-linearities and uncertainties caused by for instance model errors and algorithmic approximations [9]. The main disadvantage with particle filters is the computational power it requires, however, with modern technology this is not considered to be a hold back.

The localization algorithm that was used is called AMCL which stands for Adaptive Monte Carlo Localization. In other words, it is a particle filter as described in section 3.2 with the addition of the adaptive part that adapts the number of particles used in the filter. This is done by estimating the error between true posterior and

the approximation using Kullback–Leibler divergence. As a result, AMCL outperforms regular MCL and converges much faster by eliminating redundant particles[9]. It is available as an open-source ROS package. AMCL[1] is capable of using different measurement models, the one chosen in this project is described in section 3.2.2 and called Likelihood Range Finder Model. This model is good at handling cluttered environments because of the smoothness of the nearest neighbour function. The BRF Model would be a worse choice for handling cluttered environments.

To record the initial static map, containing only static objects, a SLAM algorithm named Gmapping[2] was used. It has been shown that this produces accurate maps in new environments[20]. However, since the initial static map is a binary image, it would also have been possible to manually construct this map. Since floor plans are often available when planing the routes of AGVs, these could be used to create the initial static map.

As previously mentioned, in order to localize with good accuracy in a changeable environment, these changes must be captured and added to the map. A SLAM algorithm named HectorSLAM[3] has been proven to perform well and also having low CPU usage and RAM usage[20]. This was modified to at command start mapping when a change in the environment was discovered. Additionally, it was modified so that it uses the position estimated by the AMCL algorithm, instead of performing localization as well.

4.2 Change Detection

To decide whether the environment has changed or not, the most recently updated map must be compared to the incoming LiDAR data. This is done by using the BRF Model, as described in section 3.2.2. This is chosen instead of the Likelihood Range Finder Model because of it's ray casting function. This function casts a ray in the map from the estimated position in the estimated direction and measures what the distance should be to the nearest object. The nearest neighbour function in the Likelihood Range Finder Model on the other hand measures the distance from the estimated endpoint of the beam to the closest object in the map. This means that if an object has been removed and is still present in the map, the nearest neighbour function might still find an object that agrees with the map, for instance a wall behind the removed object. The ray casting function on the other hand will not agree with the map, and is therefore a better choice for the change detector.

The BRF Model receives the estimated position and heading of the AGV and a LiDAR scan and assigns a likelihood using the map which corresponds to how much the scan agrees with the map. To track this agreement over time a method called Weighted Recency Average is used[21]. In this method, a short term and long term moving average of the likelihood is calculated, then the divergence between these moving averages is calculated. If the divergence is above zero, a temporary map starts to be recorded. When the divergence reaches zero again, the temporary map

[1] `http://wiki.ros.org/amcl`
[2] `http://wiki.ros.org/gmapping`
[3] `http://wiki.ros.org/hector_slam`

stops recording. The algorithm can be seen in alg.6. Since the BRF Model is highly discontinuous in the heading, that is, a small error in the estimated heading can cause a big difference in detected range, the change detection stops during turns.

Algorithm 6: Change Detection

Input: LiDAR observations ($\mathbf{z}_t$), odometry ($\mathbf{x}_t$) and map ($\mathbf{m}$)
Output: Divergence value (d)

$W_{avg}(t) \leftarrow p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m})$
$W_{slow}(t+1) \leftarrow W_{slow}(t) + \alpha_{slow}\left(W_{avg}(t) - W_{slow}(t)\right)$
$W_{fast}(t+1) \leftarrow W_{fast}(t) + \alpha_{fast}\left(W_{avg}(t) - W_{fast}(t)\right)$
$d \leftarrow \texttt{max}\left(0,\ 1 - \frac{W_{fast}(t)}{W_{slow}(t)}\right)$
if $d > 0$ **then**
    start mapping temporary map
end
else if $d \leq 0$ **then**
    stop mapping temporary map
end

4.3 Map Merging Node

Implementing the map merging functionality as a ROS node, communication with other parts of the system needs to be handled properly. Fig. 4.1 presents a graph representation of the merging node and associated topics and services (dashed line).

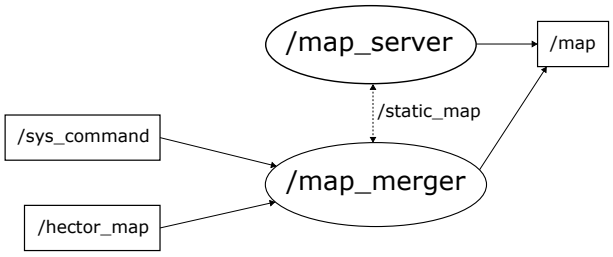


Figure 4.1: Visualization of the ROS components in direct contact with the map merging node

The merging node is called whenever the change detector announces that changes are captured as a temporary map. Communication between these nodes adopts the topic *sys_command*, maintained by *Hector Mapping* to decide when to trigger or reset the mapping procedure. The message is a string containing a commanding state: *'sleep'* sets Hector in a resting state, temporarily pausing the mapping procedure while *'resume'* resets the current map and initializes a new one. The former state message triggers the map merger as it indicates that the mapping of change is finished.

Next is to make sure the necessary maps are collected. Since the map merging node is governed by the *sys_command* topic and activated sporadically, the most convenient way to collect maps are services, meaning that a service call is sent

whenever the data is needed. For this specific type of map merging, a total of three maps are necessary: a static representing the definite boundaries of the environment and its fixed structures such as walls, a temporary representing the present state as observed by the LiDAR and finally a current being the latest combination of maps. As illustrated in fig. 4.1, the service *static_map* is provided by the map server node, returning the map topic data when called. This node is defined to hold the current representation and is also used by AMCL to compute pose estimates. It is therefore initialized with a pre-recorded static map before maintaining the merged maps as published by the map merging node. In order to keep the static map, it is stored as a global variable as part of the initialization. Using services would also be desirable for obtaining the map from the Hector map topic. However a consequence of setting the Hector state to *'sleep'* is that it temporarily disables the services provided. The topic containing the latest map representation is not affected by this command thus this topic is subscribed to, forcing an exception to the recent statement favouring services.

4.3.1 Map Merging Callback

The implementation of merging two occupancy grids is based on material from [15] and [22]. The desired outcome of these reports is to merge maps of multiple robots, finding the global offsets combining the maps. In this project the desired outcome is instead to adjust the rotational and translational offset of a static and a temporary map representing the same local environment at different times, with the offset being a product of a varying pose estimation. In other words instead of searching over the global environment, only a local search close to the pose estimate is necessary. The previous work consists of well known formulations, thus a variety of Image Processing libraries in Python can be used. In general, functions from these libraries has been used whenever available, allowing for faster prototyping. Unavailable and other supporting functions has been implemented while following the same structure as the existing libraries. The list below presents the complete list of python libraries used for map merging

- Numpy (common library for data structures and scientific computing)
- OpenCV (library consisting of computer vision algorithms)
- Scikit-Image (library dedicated to image processing algorithms)
- Scipy (support library for scientific computing)
- Matplotlib (visualization)

Alg. 7 presents a simplified description of the map merging procedure proposed

by Carpin, all steps to be covered by the following sections.

Algorithm 7: Simplified Map Merging Procedure

Input: Static and temporary occupancy grid
Output: Current occupancy grid

```
pre-process grids
compute Hough properties and locate local peaks (hypotheses)
tune rotation hypotheses
foreach tuned hypothesis do
    compute XY-spectrum and locate local peaks
    foreach translation hypotheses do
        transform current grid
        evaluate transformation
    end
end
overlay grids
post-process current grid
```

4.3.1.1 Grid Conversion and Processing

The occupancy grids message defined in ROS holds general information about the grid along with cell values of the type `int8`. Cell occupancy is defined by an integer in the range $[0, 100]$, 0 representing *free* and 100 *occupied*. There is also an *unknown* state represented by -1. In OpenCV images are of the type `uint8`, each pixel containing a value in $[0, 255]$ (for grayscale images), it is therefore necessary to perform a conversion between the two types. The navigation stack provides the package *map_server*, a package traditionally used to maintain a topic carrying map representations while also allowing for saving images of a map. Since it handles occupancy grids as inputs, this package already performs the desired conversion and this implementation is used without further modification. Furthermore an opposite conversion from image to occupancy grid is defined based on code from the same package.

When both images are in the correct format, they are further processed to detect their edges. The implementation of the Hough Transform does handle grayscale images, however by pre-processing for edges, more distinct local maxima are produced and the processing time is greatly reduced. This is a result of sparser images due to redundant cells not contributing with votes for the accumulated Hough Space. The chosen Canny edge detector constructs a binary image of edges by filtering the intensity gradients in the image. The result of Canny edge detection is visible in fig. 4.2.

4.3.1.2 Hough Rotation

The `hough_line` function that is part of the *Scikit-Image* library, implements the Hough Transform as presented in sec. 3.4.1. Unlike the implementation in *OpenCV* it also allows to keep the Hough Space as an array. Feeding the pre-processed static and temporary map through this function returns the necessary information to

pinpoint the lines of each image. Analyzing the Hough Spaces under the assumption that the contain shared features is pursued by custom functions equivalent to eq. (3.15)-(3.16).

Since the peaks in the cross-correlation of both images corresponds to possible shifts for which the images match, the most prominent ones will be used as hypothesis rotations. In order to limit the amount of peaks the function `find_peaks` from *Scipy* is used. This function supplies constraints for peaks, throwing away the ones not satisfying certain properties. The parameters defining these properties has been empirically defined during the process of examining multiple occupancy grids. As this approach limits the search space close to its estimated pose, additional filtering has been applied to the set of hypotheses, restricting the maximum angular deviation from the static map to 3 degrees. Worth noting is that if no hypotheses satisfies these constraints, the temporary and consequently the merged map is rejected. In order to limit the computational resources the process is shut down and returned empty. When one or more hypotheses however does satisfy the constraints each individual element is tuned according to an approach introduced by [22]. The approach matches peaks in Hough spaces located in the neighbourhood of the hypothesis. The space is obtained by creating a local Hough space centered around the hypothesis while spanning over a finer angular resolution in the interval $[\beta_1,\ \beta_2]$ where $\beta_1 = \theta_{hyp} - \Lambda$ $\beta_2 = \theta_{hyp} + \Lambda$. Λ representing the angular width defining the search boundary. After defining the new $\rho - \theta$ spaces, the maximum values or peaks in each space is extracted according to

$$\delta_\theta = \underset{\theta}{\texttt{argmax}}\ \texttt{max}\left(\mathbf{HT}_1^{\theta=\beta_1:\beta_2}\right) - \underset{\theta}{\texttt{argmax}}\ \texttt{max}\left(\mathbf{HT}_{2'}^{\theta=\beta_1:\beta_2}\right) \tag{4.1}$$

The interpretation of eq (4.1) is that by assuming the static reference image and the temporary map rotated by the hypothesis represent similar features, the maximum accumulator value in each space would correspond to the equivalent line feature. By extracting the angular component of the maximum values and subtracting the difference, a relative correction value is obtained. Adding the correction value from the previously calculated hypothesis then represents the tuned rotation

$$\theta = \hat{\theta} + \delta_\theta \tag{4.2}$$

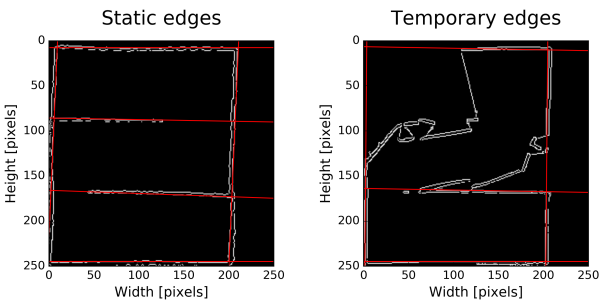


Figure 4.2: Most prominent lines identified by applying the Hough Transform to Canny edges

All hypotheses are then considered and evaluated with respect to a metric referred to as the acceptance index. Since the investigated range is small, there is usually no more than two hypotheses at once.

4.3.1.3 Translation

The initial approach to cope with relative translations between corresponding images utilized expression (3.17)-(3.19) as introduced by Carpin [15]. The approach is as previously noted straightforward as it simply counts the accumulated value of occupied space along the X and Y axis in an image. When searching for translational shifts in the global space, it provides valuable insight on what shifts are particularly interesting by using the cross-correlation function (3.19). Previous reports indicated viable results given a relatively large overlap of the corresponding maps.

4.3.1.4 Acceptance index

Even though constraints restraining the maximum merging transformation are satisfied, many factors might influence the merging procedure causing inadequate result. An inaccurate temporary map might cause great damage if its information is added to the current representation, thus some measurement to decide whether or not the temporary map should be discarded is necessary. Previous reports skip this completely or relies on a measure defined by Carpin [15], the *acceptance index*. It is important to note that this measure compares images cell-by-cell, focusing on defining the similarities between two representations of the same environment. As a result, this measure is not suitable for comparing representations where one might contain local changes. To cope with this specific problem, Andersone and Liekna [23] expands the ideas introduced by Carpin to further handle local changes. Later reports by Andersone [24] does however indicate that the algorithm might originally have been developed to handle local inconsistencies of the same environment, not changes in terms of large objects appearing or disappearing. Initial experiments has showed that it consistently produced better values than the original acceptance measure. The actual derivation of the extended acceptance index, makes use of concepts presented by Carpin. The difference is additional information from distance transforms. The distance transform in this case is essentially a grid of values corresponding to the distance of closest occupied cell in pixels. The distance calculation can be produced by several different measures, Andersone and Liekna recommend using the Manhattan distance (L1-norm) but other types such as the Euclidean distance (L2-norm) would work as well. They further apply the distance transform algorithm defined by Birk [25]. This was implemented and tested, showing the execution time compared to third-party implementations were extremely slow (even for smaller images). This adaptation instead uses the OpenCV function `distanceTransform` to produce distance maps. Alg. 8 presents the complete procedure given the distance maps as input. As mentioned implementations indicate that this approach is intended for inconsistencies of two maps of the same environment, however it appears more stable in terms of not causing false positives.

Algorithm 8: Modified acceptance index

Input: $m_{static},\ m_{temp},\ m_{dist,static},\ m_{dist,temp}\ \in \mathbb{R}^{rows \times cols}$
Output: Similarity measure

```
for r in rows do
    for c in cols do
        if the absolute difference between the static and temporary is lower
         than sim_thres then
            if static cell is free then
                sim_free += 1
            end
            else if static cell is occuied then
                sim_occ += 1
            end
        end
    end
    if the map cells are a combination of both free and occupied then
        if static cell is free then
            if cell in static distance map is lower than dist_thres then
                sim_occ += 1; sim_free += 1
            end
            else
                dis_occ += 1;dis_free += 1
            end
        end
        else if temporary cell is free then
            if cell in static distance map is lower than dist_thres then
                sim_occ += 1; sim_free += 1
            end
            else
                dis_occ += 1;dis_free += 1
            end
        end
    end
end
calculate occupied and free similarity s = sim/(sim+dis)
calculate weighted similarity S = w * s_occ + (1 - w) * s_free
```

4.3.1.5 Overlay

When a recent temporary map has been recorded it is necessary to combine the previous and new information into a current representation. For this purpose the following algorithm has been constructed

Algorithm 9: Occupancy Grid Overlay

Input: $m_{static},\ m_{temp},\ m_{current} \in \mathbb{R}^{rows \times cols}$
Output: m_{comb}

```
1  m_comb = m_static
2  for r in rows do
3  |  for c in cols do
4  |  |  if cell in static map is free then
5  |  |  |  if cell in temporary map is either free or occupied then
6  |  |  |  |  combined cell = temporary cell
7  |  |  |  end
8  |  |  |  else if cell in current map is occupied then
9  |  |  |  |  combined cell = current cell
10 |  |  |  end
11 |  |  end
12 |  end
13 end
```

The static map defines non-changing features and the enclosed space represents the possible locations, only *free* cells in the static map are proceeded with. Meaning that temporary or current cells are processed only if they corresponds to a free static cell. Cells are classified according to a value between 0 and 255, with unknown cells being a specific integer around 200. To allow looser limits around the unknown value, offset variables are added. The algorithm evaluates each cell in the grid space. When a cell of the static map is *free*, the objective is to determine whether or not new information is present. Given that the cell state of the temporary map (provided by *Hector Mapping*) is not *unknown*, this map being recently recorded represents new information and this value is obtained by the combined map. If the state in the temporary map is *unknown*, for example areas remained to be explored by the temporary mapping, the current map is instead observed. The current map represents the latest combination of a static and temporary map thus allows previously captured changes to be kept until removed by dominant information. If the temporary map lacks information about a certain cell, it is investigated if the corresponding cell in the current map is occupied. Since it already contains a combined representation together with the static map, an occupied cell is the only relevant new information. In other words, if a cell is *free* in the static map and the state in either the temporary or current map is known, the current value is updated.

4.4 ROS structure

As previously mentioned, ROS was chosen as the framework for the algorithm. This allowed to build on top of and combine useful algorithms into a fully functional localization and navigation system. The exact version of ROS is called Kinetic and is argued to be the most stable version, it runs on top of the Linux-based operating system Ubuntu 16.04 LTS and supports most packages. Several existing packages has been utilized to contribute to the complete algorithm. Described earlier, Gmapping and Hector SLAM constitutes the mapping algorithms while AMCL is used to localize in certain maps. AMCL is part of a larger package called Navigation Stack that contains other useful functionality such as a node to maintain maps (map server). In addition to these main packages, a separate node for facilitating the communication with the Pepperl+Fuchs R200 LiDAR to match what is expected by the ROS system was used. Before adapting the ROS system onto the existing AGVE system, initial testing of both the system and algorithm performance was carried out in simulation. This was made possible using Gazebo, a robotic 3D simulator supporting ROS, and the Turtlebot3 package, defining a complete robotic system similar to the one used in physical experiments. The complete ROS system is visualized by fig. B.1.

The following list presents all packages used:

- Navigation Stack[4]
- Hector SLAM[5] [26]
- Gmapping[6] [27]
- Pepperl+Fuchs R2000 Driver[7]
- Turtlebot3 [8]
- Gazebo[9]

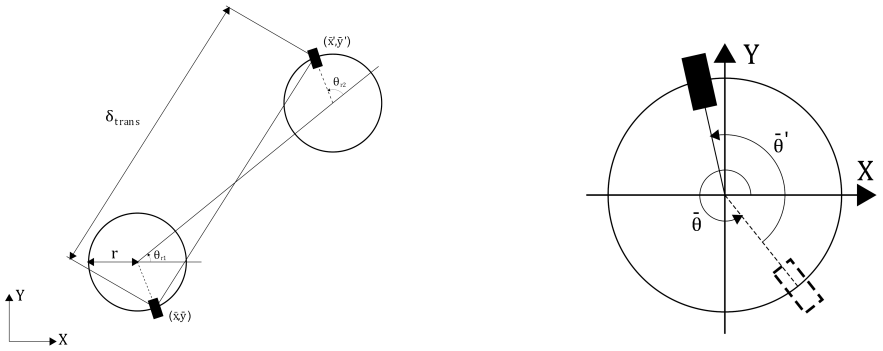
4.5 Adaptation to AGVEs System

Mentioned earlier, the AGV was run with the localization algorithm on a separate computer. The computer was connected to the AGV's computer and also to the LiDAR using Ethernet. Drivers for the Pepperl Fuchs R2000 LiDAR was available on ROS, however, to communicate with the AGV a script was written. The data that needed to be transferred was the odometry data from the AGV to the computer and the estimated position from the computer to the AGV.

[4] `http://wiki.ros.org/navigation`
[5] `http://wiki.ros.org/hector_slam`
[6] `http://wiki.ros.org/gmapping`
[7] `http://wiki.ros.org/pepperl_fuchs_r2000`
[8] `http://wiki.ros.org/turtlebot3`
[9] `http://wiki.ros.org/gazebo_ros_pkgs`

4.5.1 Odometry Model Compensation

As mentioned briefly in section 3.2.1 the odometry model defined by Thrun et. al. is formulated under the assumption that the robot is a differential wheeled robot and that the odometry is calculated from the center of rotation. The provided robot, AGVE model *Little Red Demo* is instead a tricycle with a layout similar to common forklifts. It has a single driving wheel responsible for both linear and rotational motion. The two remaining wheels acts as support wheels to maintain balance. The odometry is calculated from the single driving wheel. With this in mind fig. 4.3a describes this model and its differences in comparison to fig. 3.1. The main difference between the models is that a pure rotation leads to a translation, since the odometry in the tricycle motion model is not calculated from the rotation center. Fig. 4.3b further explains the displacements in X and Y respectively, as caused by rotation of the robot.



(a) Geometric components for retrieving the relative odometry of a tricycle robot

(b) Visualization of the total displacement in X and Y due to rotation

Figure 4.3: The geometric structure and behaviour of a tricycle robot

Supported by fig. 4.3b the combined displacement is calculated in the respective direction, utilizing basic trigonometric identities

$$\delta_x = r\left(\cos\left(\bar{\theta}\right) - \cos\left(\bar{\theta} + \bar{\theta}'\right)\right) \tag{4.3}$$

$$\delta_y = r\left(\sin\left(\bar{\theta}\right) - \sin\left(\bar{\theta} + \bar{\theta}'\right)\right) \tag{4.4}$$

Comparing the geometric interpretation in fig. 4.3a with the original illustration by Thrun et. al. (fig. 3.1), their differences are then managed by incorporating the expressions computed in (4.3)-(4.4) into the algorithm 3. The final odometry model is presented in alg. 10.

Algorithm 10: Modified Odometry Sampling Model

Input: $\mathbf{u}_t, \mathbf{x}_{t-1}$

Output: $\mathbf{x}_t = (x', y', \theta')$

$\delta_x = r\left(\cos\left(\bar{\theta}\right) - \cos\left(\bar{\theta} + \bar{\theta}'\right)\right)$

$\delta_y = r\left(\sin\left(\bar{\theta}\right) - \sin\left(\bar{\theta} + \bar{\theta}'\right)\right)$

$\delta_t = \sqrt{\left(\bar{x}' - \bar{x} - \delta_x\right)^2 + \left(\bar{y}' - \bar{y} - \delta_y\right)^2}$

$\delta_{r1} = \texttt{atan2}\left(\bar{y}' - \bar{y} + \delta_y,\ \bar{x}' - \bar{x} + \delta_x\right) - \theta$

$\delta_{r2} = \theta' - \theta - \theta_{r1}$

$\hat{\delta}_t = \delta_t - \texttt{sample}\left(\alpha_3\delta_t^2 + \alpha_4\delta_{r1}^2 + \alpha_4\delta_{r2}^2\right)$

$\hat{\delta}_{r1} = \delta_{r1} - \texttt{sample}\left(\alpha_1\delta_{r1}^2 + \alpha_2\delta_t^2\right)$

$\hat{\delta}_{r2} = \delta_{r2} - \texttt{sample}\left(\alpha_1\delta_{r2}^2 + \alpha_2\delta_t^2\right)$

$x' = x + \hat{\delta}_t \cos\left(\theta + \hat{\delta}_{r1}\right) + r\left(\cos\left(\theta + \hat{\delta}_{r1} + \hat{\delta}_{r2}\right) - \cos\left(\theta\right)\right)$

$y' = y + \hat{\delta}_t \sin\left(\theta + \hat{\delta}_{r1}\right) + r\left(\sin\left(\theta + \hat{\delta}_{r1} + \hat{\delta}_{r2}\right) - \sin\left(\theta\right)\right)$

$\theta' = \theta + \hat{\delta}_{r1} + \hat{\delta}_{r2}$

Algorithm reproduced from: *Probabilistic Robotics by Sebastian Thrun, Wolfram Burgard, and Dieter Fox, published by The MIT Press*

5
Test and Verification

This chapter describes the experiments constructed to evaluate the performance of the complete algorithm and its elements. The hardware specifications are established before describing the scenarios and what is set to investigate.

5.1 Experiment Setup

The on-board processing hardware (tab. 5.1) consists of a laptop that maintains a ROS server on top of Ubuntu. As previously mentioned this unit handles most of the data processing. It receives LiDAR and odometry messages via the Ethernet port, communicating with the embedded CB80 unit using UDP. This feed of messages is then processed by the various tools in ROS. After processing, the pose is sent using the same UDP protocol. Other than providing and receiving data from the laptop, the CB80 maintains all the existing features in the AGVE system, such as defining the correct control signals to trigger movement.

Unit	Computer	AGV (CB80[1])
OS	Ubuntu 16.04 LTS	Windows CE 6.0
CPU	Intel i5-8250U (1.60GHz x 8)	Intel Atom Z510/530 (1.1/1.6 GHz x 1)
GPU	Intel UHD Graphics 620	-
RAM	7.5 GB	-

Table 5.1: Computer setup

5.2 Experiment Scenarios

In order to evaluate the proposed algorithm, different scenarios (fig. 5.1) were investigated. The scenarios represent the same enclosed environment with different level of clutteredness. Answers to three specific questions are sought

- Performance comparison against regular AMCL
- Algorithm performance in highly cluttered environment
- Maximum amount of change at once

Using a route pre-defined according to AGVEs system, the robot is allowed to autonomously navigate using continuous pose feedback from the ROS system. Data

from the respective case was gathered using *bag files*, a file format in ROS to store message data from topics.

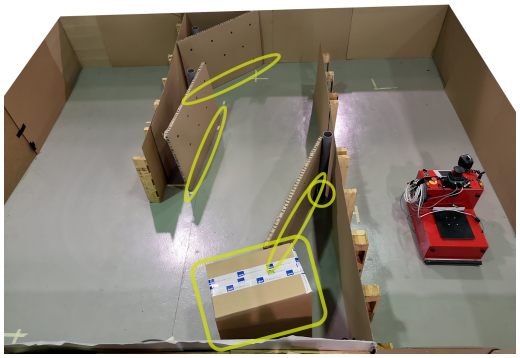
The first scenario, seen in fig. 5.1a, the accuracy and repeatability of the AMCL algorithm is evaluated in a static environment.

In the scenarios with the three increasing levels of clutteredness, fig. 5.1b, 5.1c and 5.1d the updated map from each level was kept when running the next level. Additionally, one scenario was evaluated where the AGV starts with the initial map (corresponding to fig. 5.1a) and then run in the level 3 environment to test if it could manage a big change.

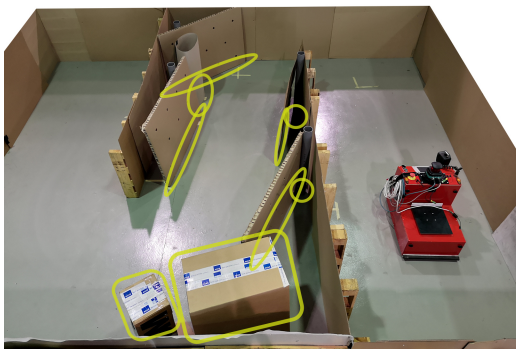
In the last scenario, fig 5.1e, the AGV was run in a environment with many small objects. The AGV started with the initial map in this case.

(a) Static

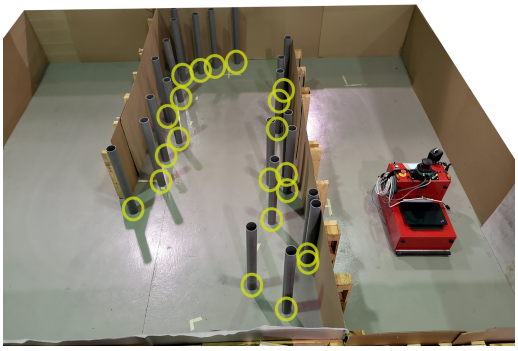
(b) Clutter level 1



(c) Clutter level 2



(d) Clutter level 3



(e) Cluttered

Figure 5.1: The testing environment and the different stages of clutteredness (changes marked with yellow)

6
Results

In this chapter, the results of the previously defined experiments are presented. First, the performance of using the AMCL algorithm with a static map is visualized. This is followed by results regarding the map merging. Finally, the results of both AMCL and the proposed algorithm exposed to the scenarios of varying clutteredness is presented, demonstrating their behaviour and the accuracy of their pose estimates.

6.1 Performance with Static Map

This test was made to evaluate the accuracy and repeatability of the AMCL algorithm in a static environment. In fig. 6.1 the map recorded by the Gmapping algorithm can be seen. In fig. 6.2a and 6.2b, the performance of AMCL can be seen. The left figure shows the 3σ covariance ellipses of the state estimation. These corresponds to a mean 3σ value in the x-coordinate of 0.0884 m, in the y-coordinate it's 0.0780 m and in the heading it's 0.0739 rad.

The right figure shows the path driven by the AGV when travelling 10 times between the top of the map to the bottom. The 3 smaller figures to the right shows that the maximum deviation in x- and y-coordinate was approximately 0.03 m.

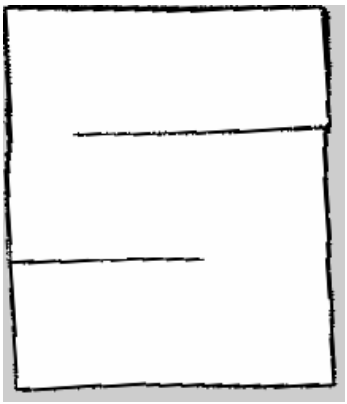
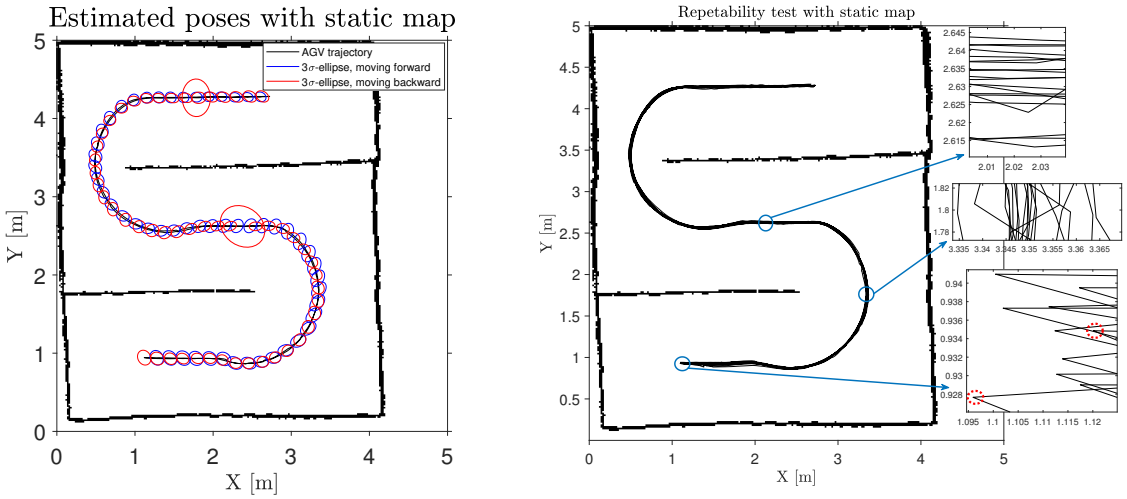


Figure 6.1: Recorded static map from Gmapping.



(a) The 3σ regions representing the uncertainty in the position estimation when using standard AMCL in a static environment

(b) Trajectory when localizing using standard AMCL for 10 laps

Figure 6.2: Performance of the algorithm in a static environment.

6.2 Map Merging Performance

In this section, results relating to the performance of the map merging algorithm are presented.

Change Level	Similarity
1	0.9947
2	0.9940
3	0.9897

Table 6.1: Similarity values for static map and the corresponding features after merging

Tab. 6.1 presents the acceptance index or measure of similarity, as a result of comparing the corresponding features in a static and a merged map. The acceptance index is the ratio of how much two maps are alike, thus the possible range is in $[0, 1]$. The similarity measures are high (corresponding to good merging quality) and decreases slowly with increasing environment complexity.

6.3 Performance with Adaptive Map

In this section the performance of AMCL with map updates is compared to AMCL without map updates in a non-static environment. In section 6.3.1 the results corre-

sponding to the scenario in fig. 5.1d is shown where the AGV begins with the map from the scenario corresponding to fig. 5.1c. In sec. 6.3.2 the scenario corresponding to fig. 5.1e is evaluated. The results for the other scenarios can be seen in the appendix (A.1-A.2). In all cases the AGV was driven autonomously from the start position to the other end of the map and then back again. For the case when the adaptive map is used, the AGV then had an updated map on the way back to the starting position.

6.3.1 Clutteredness Level 3

Fig. 6.3 shows the the output of the change detector when running in the cluttered level 3 environment. There is clear peak in the divergence when the AGV enters the changed area.

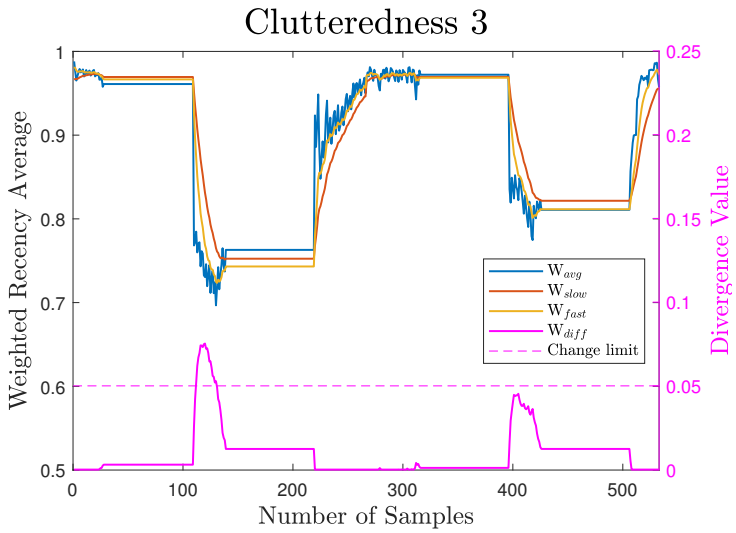


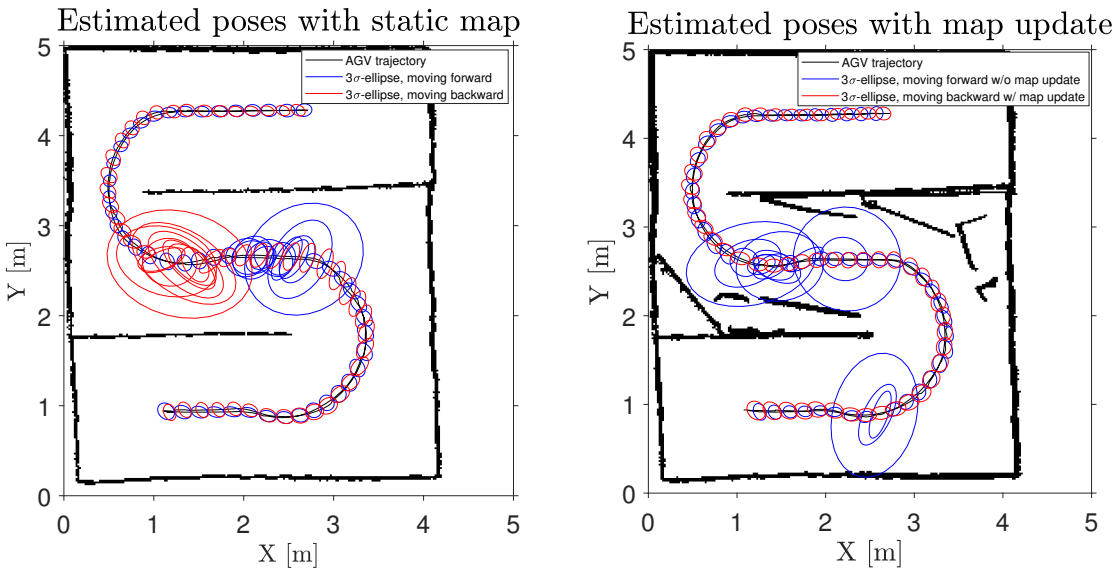
Figure 6.3: Weighted recency averages and divergence values obtained during clutteredness level 3.

Fig. 6.4 shows a comparison between the accuracy of the AMCL algorithm in static map case (left figure) and in the adaptive map case (right figure). The large 3σ ellipses (where sigma is the standard deviation) corresponds to approximately 0.75 m 3σ in the x-coordinate, 0.7 m in the y-coordinate and 33 rad in the heading.

Fig. 6.5 shows a comparison of the 3σ level for the x-position estimate, y-position estimate and heading estimate between the case with static map and adaptive map.

6.3.2 Environment with Many Small Objects

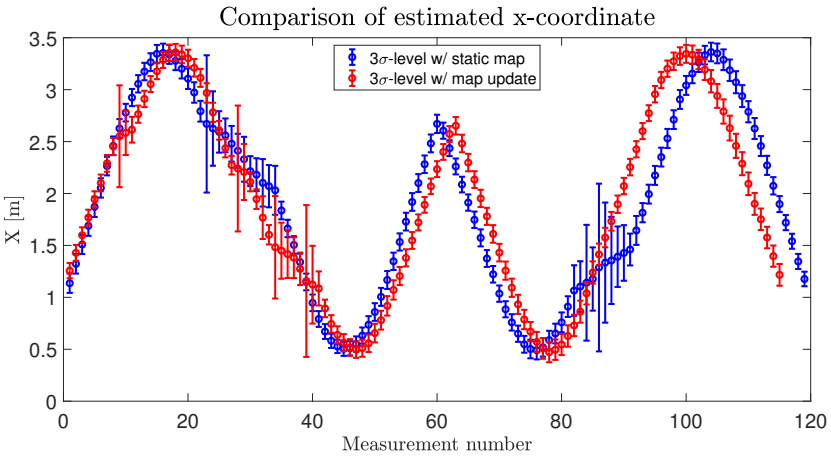
In this section, the results from the scenario when the environment is cluttered with many small objects is shown. Fig. 6.6 shows the results of the change detection and fig. 6.7 shows the 3σ regions of the position estimation.



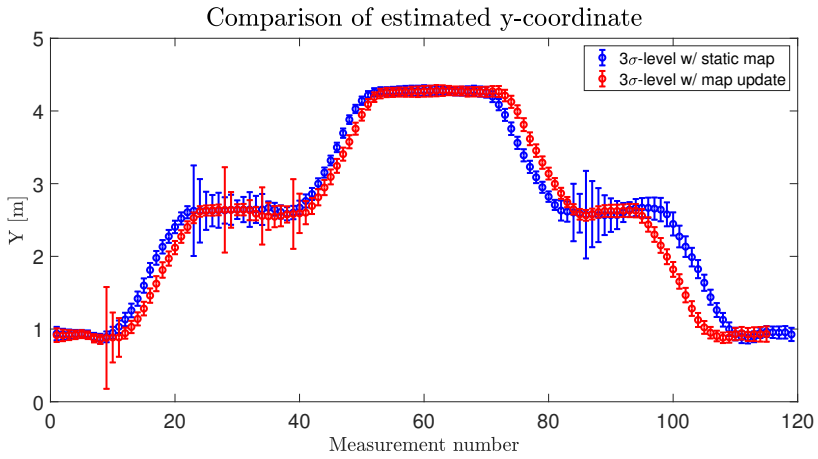
(a) Standard AMCL

(b) Proposed algorithm with adaptive map

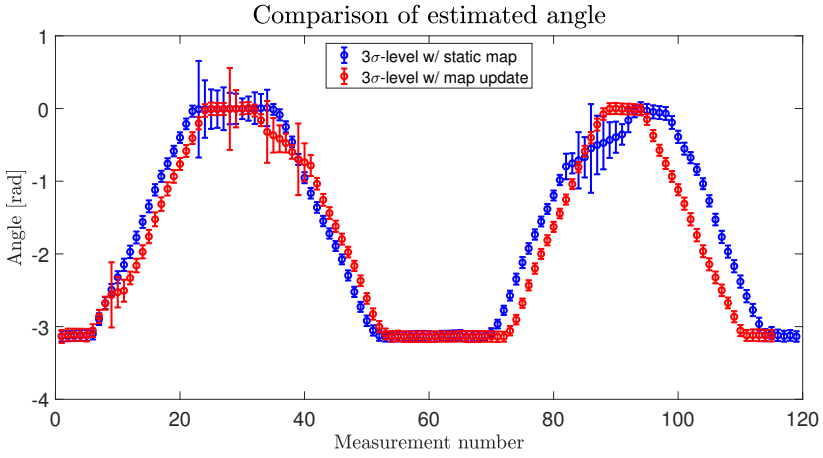
Figure 6.4: The 3σ regions and path driven by the AGV in the clutteredness level 3 environment.



(a) X



(b) Y



(c) Angle

Figure 6.5: The 3σ levels in the position and heading estimation compared when using a static and an adaptive map.

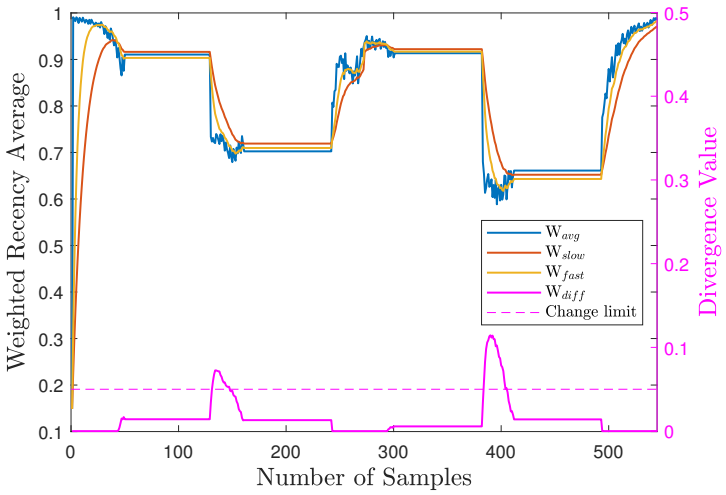


Figure 6.6: Weighted recency averages and divergence values obtained in an environment with many small objects.

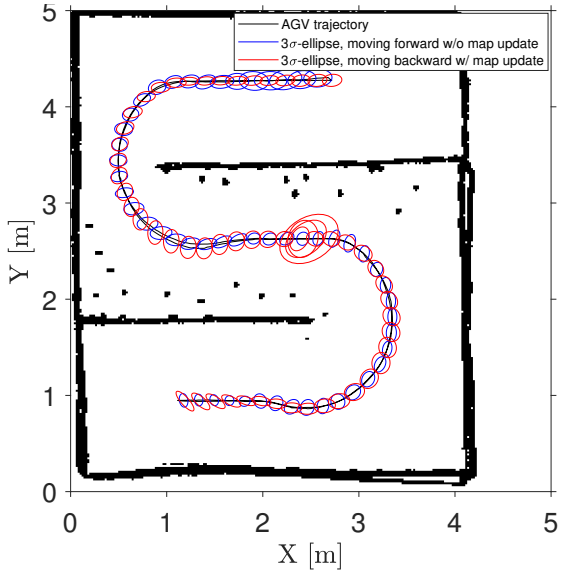


Figure 6.7: 3σ regions when the AGV runs in an environment with many small objects.

7
Discussion

The purpose of this chapter is to reflect on the results presented in chapter 6, explaining the differences between the regular AMCL localization approach and the proposed algorithm with map updates. Furthermore uncertainties and impact of limitations will be discussed before suggesting future developments.

7.1 Change Detector

From the Results chapter, it is clear that the change detector works as intended, that is, the divergence increases when there is a change in the environment. However, there is a small increase in the divergence in the changed environments, even though the map has been updated. This is probably caused by the discontinuous measurement model, the BRF model, described in section 3.2.2. Since the objects added to the environment are placed in front of the walls this increases the discontinuity of the map and the sensitive measurement model will then perform worse.

Besides the measurement model, the amount of measurements collected from the LIDAR has a big impact on the change detector. In order to fulfill the assumption of the beams being independent of each other, not too many measurements per revolution should be done, however, with too few measurement changes might not be detected. In other words, a more thorough investigation of this should be performed.

7.2 Map Merging

The map merging algorithm is not used in the same way as intended by Carpin [14], where multi-robot mapping is of most interest. For global merging, hypotheses plays a crucial part as a large map might contain several poses for where the temporary map match. In our case, the algorithm is limited to search locally around the pose estimate for a valid transformation between two grid maps. This way there is less need for hypotheses. The chosen maximum deviations in terms of rotations and translation defines the hard boundaries of the search space and with the use of correlation based matching just a few hypotheses are kept. Other reports comment on the need for tuning the rotational and translational components, all tuning approaches were implemented and tested. The method of tuning with respect to peak matching[22], performed well in terms of computational speed but continuously produced worse than the original hypotheses. Given the actual result of the algorithm and the fact that no tuning seemed to be needed in [8], the tuning functionality was deactivated. Related reports does not present performance in terms of ground truth deviation

other than in simulation environments. The reports focusing on multi-robot mapping consistently use the acceptance index[14] as a match-or-discard measure, not presenting the deviation in terms of rotation and translation. Shaik et. al. [8] identifies line segment matches and present their average pose difference. No similar evaluation was carried out, however a simple comparison in terms of image similarity is presented in tab. 6.1, an evaluation previously used by [21]. As lightly touched in the Implementation section there is an inherent uncertainty in the mapping algorithms. Two different algorithms are used, not necessarily providing equal results under perfect conditions. Experience suggests that *Hector mapping* produces less stable and accurate maps in comparison to Gmapping, thus using same algorithm to map changes might provide better results in terms of map merging. For more robust mapping, parameter tuning might be enough. Nevertheless, in its current state, it is problematic to define the optimal transformation given the previously mentioned limitations. Instead the original and transformed line features are evaluated manually. Further developments to the map merging algorithm might include investigating other types of features extractors. Even though the Hough Transform suffers from high dimensional spaces, the approach is capable of running real-time processes without major delays. More complex and larger facilities remains to be tested, but evaluating environments with large dimensions and possibly less line features would provide insight on the performance in real applications. Other implementations use the Hough Transform for real-time detection of driving lanes. These implementations use video feeds with images of far higher dimensions than our approach and this might point to satisfying performance in larger environments. If dimensionality does become an issue, the promising result of using image features might suggest the use of feature-based solutions.

7.3 Algorithm Performance

In order to evaluate the localization error, it is necessary to compare to a reference solution with high accuracy. Another report [28] rely on a high precision motion capture system with an accuracy up to 0.1 mm. This is not a method available at AGVE, instead the intended choice of evaluation was to compare relative a reflector solution. Performed directly by an onboard SICK NAV350 this was said to have an accuracy of ± 10 mm. However, later on it was decided by AGVE to use this project to evaluate another sensor, the Pepperl+Fuchs R2000. Due to inconvenience in switching between sensors and the fact that different mounting brackets cause an unknown displacement, this approach was rejected. With the industry often lacking high precision tracking systems, the most common way to evaluate localization performance is to map specific coordinates and perform several runs, measuring the deviations and computing the average. This is referred to as repeatability. Fig. 6.2b illustrates the result of 10 consecutive runs using only AMCL. This also defines the accuracy achievable by our algorithm.

Exposing the AGV to the scenarios of increasing clutteredness, the results for each level are all similar. The major differences is that the uncertainty grows larger each level, the final level therefore provides a much clearer comparison of both algorithms. This section therefore only discusses the plots visualizing the behaviour

in level 3. The results corresponding to the remaining levels can be found in A.1-A.2. First investigating the uncertainty plots in 6.4, the ellipsoids in blue represents both algorithms using a static map in the changed environment. This explains the sudden increase in uncertainty, while approaching the unmapped objects. As a consequence of this sudden increase, the change detector (fig. 6.3) is triggered and a temporary map is requested. When recorded it is later merged with the static map and used to localize in by the proposed algorithm. The differences of using a static and a updated map is visualized by the red ellipsoids, showing clear evidence that the estimates produced by the proposed algorithm are much more certain in semi-static environments. Comparing the actual trajectory (black line) of the AGV in both cases this does not seem to deviate too much in terms of accuracy, even though the uncertainty is very large. The reason for this behaviour has not been thoroughly investigated but a qualified guess is that the corridor where we apply changes is to short for large deviations to appear. When there is a mismatch between the most recent map and the LiDAR measurements, less weight is added to these measurements and they are not trusted. As a result the odometry readings becomes more trustworthy. Odometry is known to drift in time but both [9] and [29] have concluded that it provides rather accurate estimates for distances of at least 5m, a distance that is not violated by our tests. If subjecting the algorithm to longer sections containing changes, the differences would probably be much clearer. Other uncertainties that of course influence the overall performance is inherent errors from the static map, uneven floors and wheel slippage.

7.4 Future Developments

In addition to previous suggestions, the following actions related to mobile robot localization can be investigated.

With proven performance on smaller 5x5 m maps (2 cm resolution and 250x250 px images), a possible approach to cope with larger Hough spaces is to develop a system for map management where a global map is divided into smaller segments to avoid unnecessary computations. Segments might further be classified into levels in the range of static-dynamic, triggering different localization techniques depending on the region. This is an approach for implementing the solution in its current state to an embedded system, if computational efficiency is desired then it is encouraged to evaluate sparser alternatives as previously mentioned.

Most recent research on robot mapping and localization cover the use of multi-layered LiDAR sensors and cameras, generating 3D point clouds that with support of images allows detection and classifying of objects. A possible approach is to use detected objects to aid the localization algorithm or be added to/removed from the map in a similar fashion as in this thesis. The huge feed of information makes high demands on computational power thus increases the hardware cost but has the possibility of returning highly accurate estimates. Unlike outdoor environments, indoor environments remain stable in terms of weather and perception conditions, only influenced by light, an advantage of such solutions.

Finding optimal parameters by further tuning is important. Especially for the LiDAR, where the noise parameters can be decided by making thorough tests. The

noise in the motion model might also need adjustment and can decided through testing.

Another development is that the measurement model in the change detector (BRF model) should be modified so that it is not highly discontinuous but smoother like the Likelihood range finder model. This might be done by for each LiDAR scan, ray cast multiple rays in the map for angles close to the estimated one to see whether any of the casted ones are close to the LiDAR scan. The measurement model in the AMCL algorithm should also be improved to be able to handle short readings.

8

Conclusion

The main goal of this thesis was to implement an algorithm to detect and map changes in the surrounding environment. It was desired to investigate whether the proposed algorithm could improve the localization accuracy by having a map capable of representing a changing environment. The ambition was to keep adapting to the changing environment for longer periods of time, thus approaching what is referred to as lifelong mapping, where mobile robots navigate seamlessly without any human interference.

Assessment in environments of increasing clutteredness showed an improvement with respect to the standard AMCL algorithm. The actual pose estimates were not compared to any ground truth (high performing solution), thus no relative localization errors were obtained. Instead the performance were evaluated by comparing pose uncertainties provided by the AMCL algorithm. For each environment tested, the algorithm produced lower uncertainties and thus arguably more accurate estimates were obtained. Taking a closer view on the trajectories defining the path traveled, it does not seem to deviate any more than using the proposed algorithm. The conclusion of this specific result is that whenever the laser measurements does not match the maintained map, the system becomes uncertain. As a consequence of this, more weight is instead added to the odometry measurements. Illustrations in both [9] and [29] confirm this behaviour as it is argued that odometry provide accurate estimations up to around 4 meters without feedback from other sensors. For larger environments with un-mapped objects, the pose estimates from the standard AMCL algorithm would probably drift away causing a great difference in localization performance.

The overall results shows desirable performance in simple environments and is in its current state ready for more complex and large environments. Possible limitations has been mentioned and if future testing shows lacking performance, the actions in sec. 7.4 are recommended.

Bibliography

[1] J. Boal, A. Sánchez-Miralles, and A. Arranz. Topological simultaneous localization and mapping: A survey. *Robotica*, 32:803–821, 08 2014.

[2] S. Thrun and A. Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *AAAI/IAAI, Vol. 2*, 1996.

[3] D. Hahnel, R. Triebel, W. Burgard, and S. Thrun. Map building with mobile robots in dynamic environments. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1557–1563 vol.2, Sep. 2003.

[4] P. Biber and T. Duckett. Dynamic maps for long-term operation of mobile service robots. In S. Thrun, G. Sukhatme, and S. Schaal, editors, *Robotics: Science and Systems*, pages 17–24. The MIT Press, 2005.

[5] G.-D Tipaldi, D. Meyer-Delius, and W. Burgard. Lifelong localization in changing environments. *The International Journal of Robotics Research*, 32(14):1662–1678, 2013.

[6] K.Murphy. Bayesian map learning in dynamic environments. In *NIPS*, 1999.

[7] D. Meyer-Delius, J. Hess, G. Grisetti, and W. Burgard. Temporary maps for robust localization in semi-static environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5750–5755, Oct 2010.

[8] N. Shaik, T. Liebig, C. Kirsch, and H. Müller. Dynamic map update of non-static facility logistics environment with a multi-robot system. In G.Kern-Isberner, J. Fürnkranz, and M. Thimm, editors, *KI 2017: Advances in Artificial Intelligence*, pages 249–261. Springer International Publishing, 2017.

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

[10] H.Moravec and A. E. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 116 – 121, March 1985.

[11] J-A.Fernández-Madrigal and J.Blanco Claraco. *Simultaneous localization and mapping for mobile robots: Introduction and methods*. IGI Global, 2012.

[12] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[13] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[14] A. Birk and S. Carpin. Merging occupancy grid maps from multiple robots. *Proceedings of the IEEE*, 94(7):1384–1397, July 2006.

[15] S. Carpin. Fast and accurate map merging for multi-robot systems. *Autonomous Robots*, 25(3):305–316, Oct 2008.

[16] R. Duda and P. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15:11–15, 1972.

[17] A. Censi, L. Iocchi, and G. Grisetti. Scan matching in the Hough domain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2739–2744, Barcelona, Spain, 2005.

[18] D. M. Rosen, J. Mason, and J. J. Leonard. Towards lifelong feature-based mapping in semi-static environments. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1063–1070, May 2016.

[19] H.Kretzschmar, G.Grisetti, and C.Stachniss. Lifelong map learning for graph-based slam in static environments. *KI - Künstliche Intelligenz*, 24(3):199–206, 2010.

[20] A. Pålsson and M. Smedberg. Investigating simultaneous localization and mapping for agv systems. Master's thesis, Chalmers tekniska högskola, 2017.

[21] F.Abrate, B.Bona, M.Indri, S.Rosa, and F.Tibaldi. Map updating in dynamic environments. In *ISR/ROBOTIK*, 2010.

[22] S. Saeedi, L. Paull, M. Trentini, M. Seto, and H. Li. Map merging for multiple robots using Hough peak matching. *Robotics and Autonomous Systems*, 62(10):1408–1424, 2014.

[23] I.Andersone and A.Liekna. Robot map similarity evaluation for non-identical maps. *Engineering for Rural Development*, pages 456–461, 01 2013.

[24] I.Andersone and A.Nikitenko. Reliable multi-robot map merging of inaccurate maps. In Y.Demazeau, F.Zambonelli, J-M.Corchado, and J.Bajo, editors, *Advances in Practical Applications of Heterogeneous Multi-Agent Systems. The PAAMS Collection*, pages 13–24, Cham, 2014. Springer International Publishing.

[25] A.Birk. Learning geometric concepts with an evolutionary algorithm. In *Proceedings of The Fifth Annual Conference on Evolutionary Programming*. The MIT Press, 1996.

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

[27] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb 2007.

[28] J. Röwekämper, C. Sprunk, G. D. Tipaldi, C. Stachniss, P. Pfaff, and W. Burgard. On the position accuracy of mobile robot localization based on particle filters combined with scan matching. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3158–3164, Oct 2012.

[29] R.Siegwart, I-R. Nourbakhsh, and D.Scaramuzza. *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd edition, 2011.

A

Additional Results

A.1 Clutter level 1

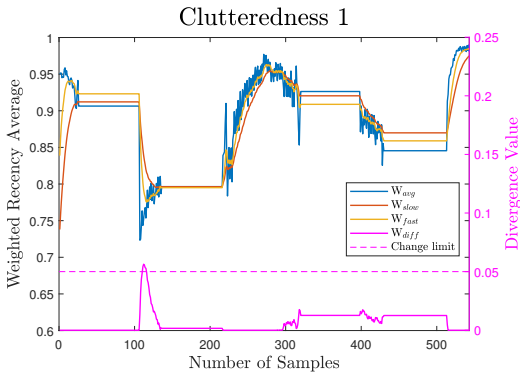
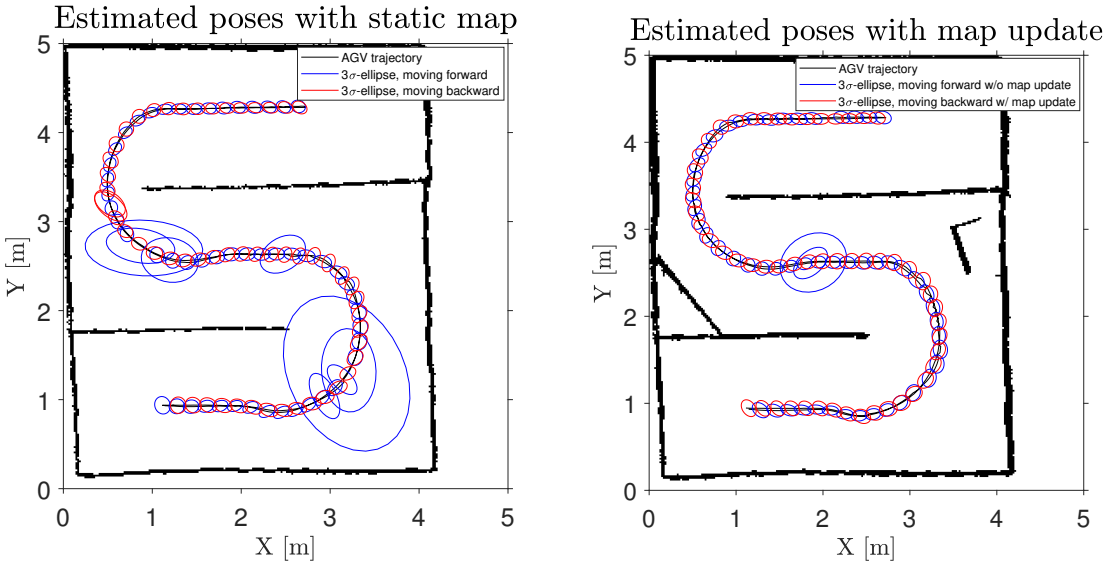
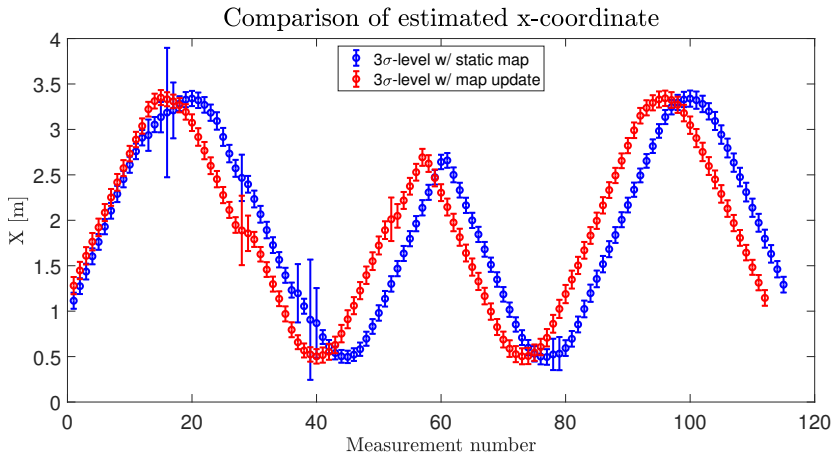


Figure A.1: Recency average weights and divergence values obtained during clutteredness level 1.

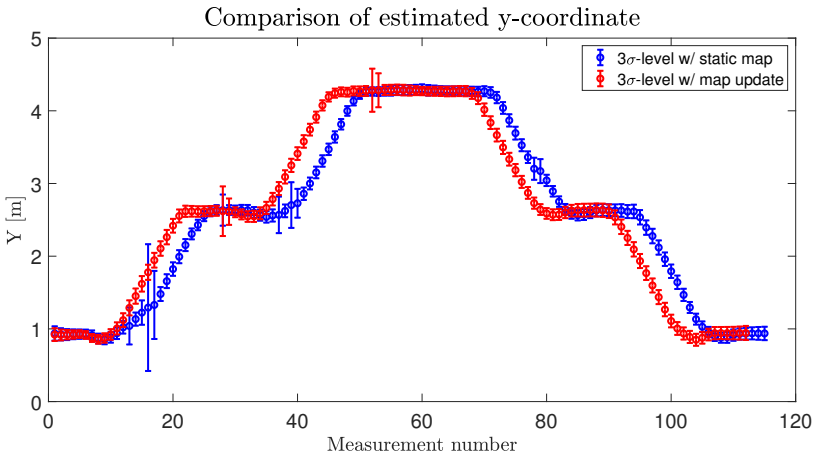


(a) AMCL **(b)** Proposed algorithm

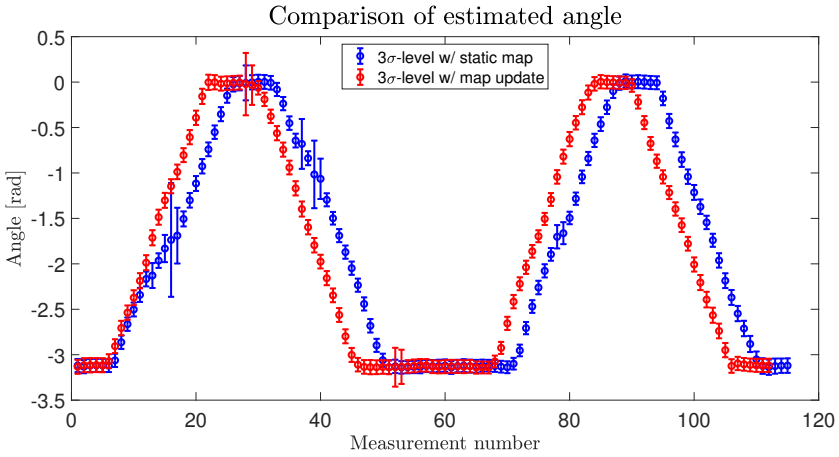
Figure A.2: The 3σ regions and path driven by the AGV in the clutteredness level 1 environment.



(a) X



(b) Y



(c) Angle

Figure A.3: 3σ deviations on pose components collected from clutteredness level 1.

A.2 Clutter level 2

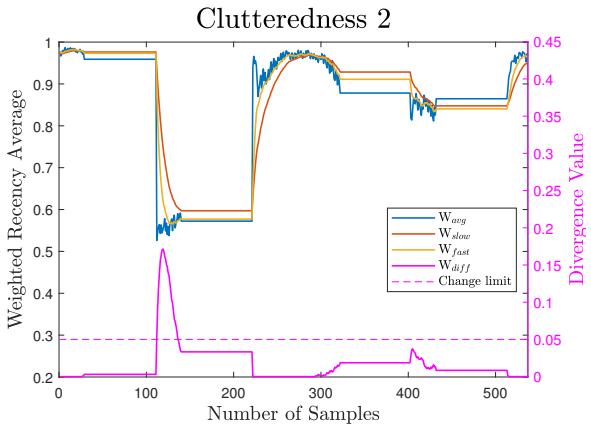
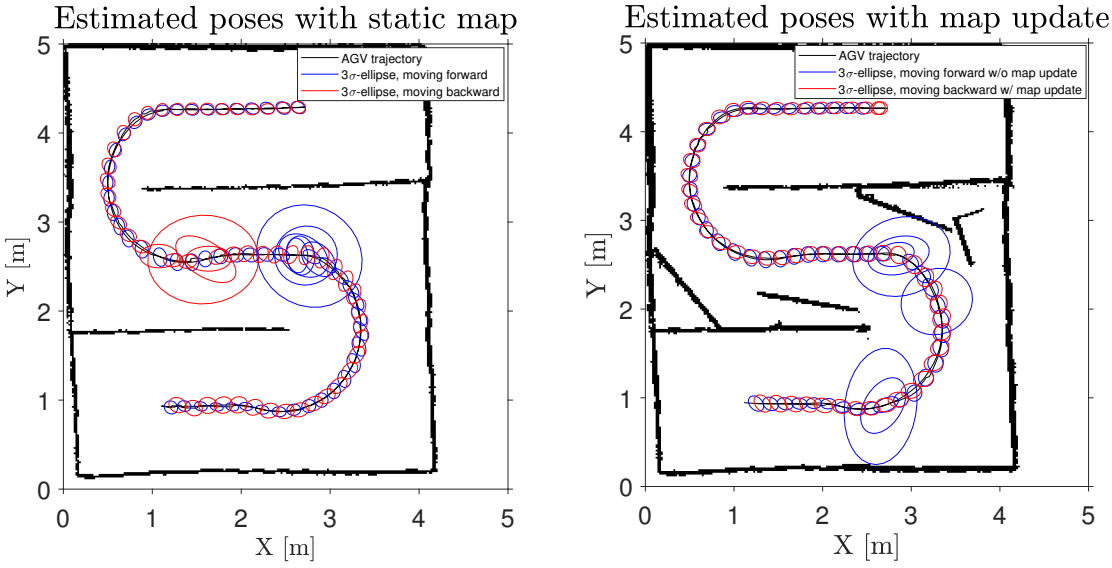
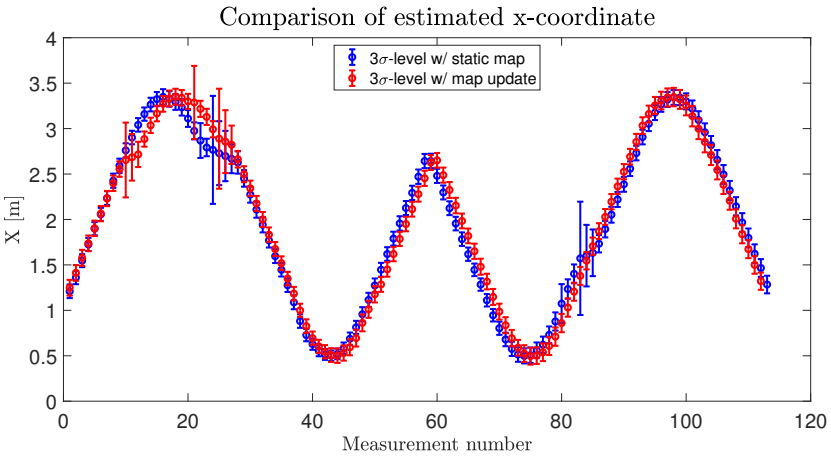


Figure A.4: Recency average weights and divergence values obtained during clutteredness level 2.

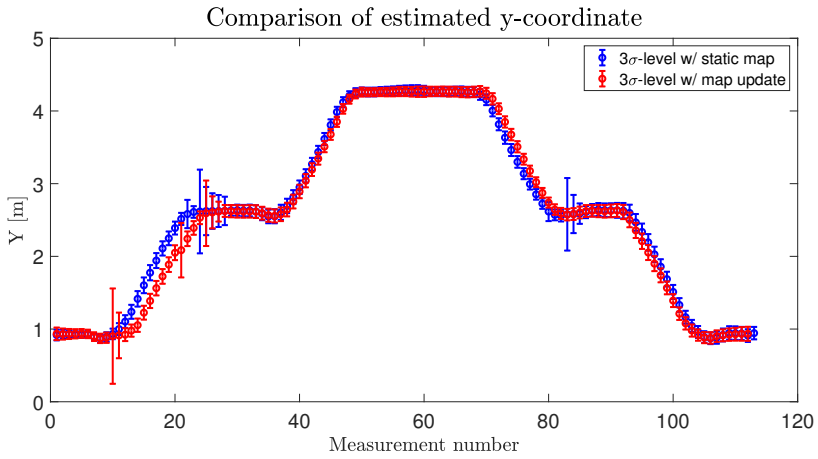


(a) AMCL **(b)** Proposed algorithm

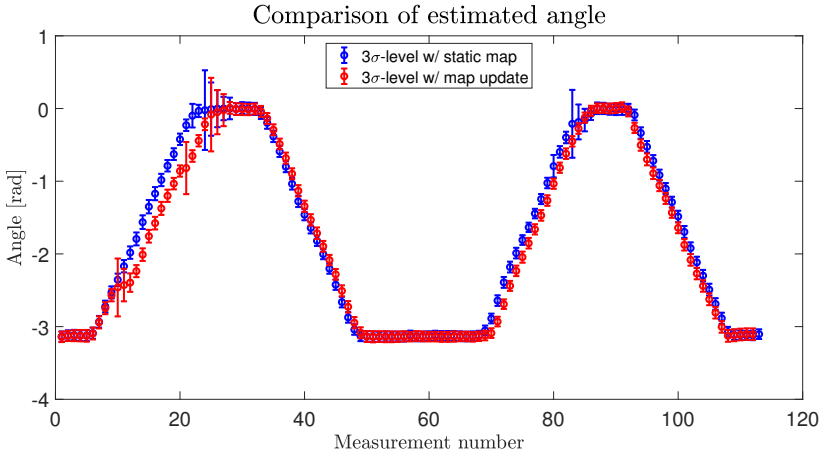
Figure A.5: The 3σ regions and path driven by the AGV in the clutteredness level 2 environment.



(a) X



(b) Y



(c) Angle

Figure A.6: 3σ deviations on pose components collected from clutteredness level 2.

B

Complete ROS Graph

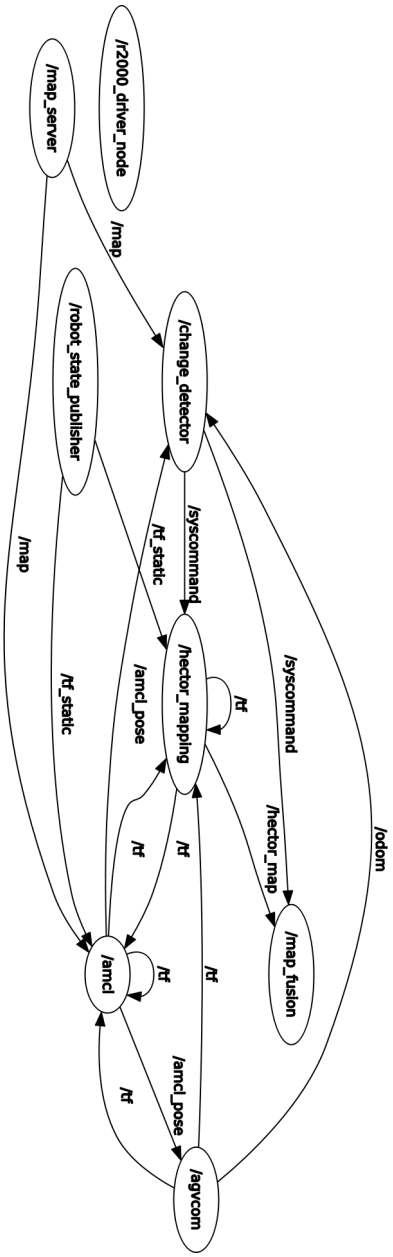


Figure B.1: Complete ROS graph of nodes, topics and their connections

C

Pepperl Fuchs R2000 Datasheet

2-D LiDAR Sensor — OBD10M-R2000-4EP-V1V17

Technical data

General specifications		
Measurement range		0.2 ... 3 m (bk 10%) 0.2 to 10 m (wh 90%) 0.2 to 30 m (reflector)
Light source		laser diode
Light type		modulated visible red light
Laser nominal ratings		
Note		LASER LIGHT , DO NOT STARE INTO BEAM
Laser class		1
Wave length		660 nm
Beam divergence		1 mrad
Pulse length		5 ns
Repetition rate		54 kHz
max. pulse energy		< 4 nJ
Measuring method		Pulse Ranging Technology (PRT)
Scan rate		10 Hz, 20 Hz, 30 Hz
Scanning angle		360°
Diameter of the light spot		< 20 mm at 10 m
Ambient light limit		> 80000 Lux
Functional safety related parameters		
$MTTF_d$		75 a
Mission Time (T_M)		20 a
Diagnostic Coverage (DC)		0 %
Indicators/operating means		
Operation indicator		LED green
Data flow indicator		LED yellow: active ethernet LED green: Ethernet link
Function indicator		LED red: fault Yellow LED: I/Q1 + I/Q2
Control elements		2 Button
Parameterization indicator		24 x 252 pixels , red
Electrical specifications		
Operating voltage	U_B	10 ... 30 V DC
Ripple		10 % within the supply tolerance
No-load supply current	I_0	≤ 400 mA / 24 V DC
Protection class		III (operating voltage 50 V)
Power consumption	P_0	< 10 W
Time delay before availability	t_v	< 40 s
Integrated application		
Application		Field monitoring
Number of fields		4
Response time		30 ms + 1 Scan duration
Detectable object shape		Almost any
Object size		> 1 mm
Linking fields		Up to 4 x 3 levels
Interface		
Interface type		4 x switching inputs/outputs (selectable)
Input/Output		
Input/output type		4 Inputs/Outputs , Independently configurable , short circuit/ reverse polarity protected
Input		
Switching threshold		low: Ue < 5 V, high: Ue > 10 V
Output		
Switching threshold		low: Ua < 1 V, high: Ua > Ub - 1 V
Switching current		100 mA per output
Measurement accuracy		
Measuring speed		54000 measurements per second
Angle resolution		0,071°; 0,15°; 0,2°
Repeat accuracy		< 12 mm
Ambient conditions		
Ambient temperature		-10 ... 50 °C (14 ... 122 °F)
Storage temperature		-20 ... 70 °C (-4 ... 158 °F)
Relative humidity		95 % , no moisture condensation
Mechanical specifications		
Housing width		106 mm
Housing height		116.5 mm
Degree of protection		IP65
Connection		4-pin, M12x1 connector, A-coded (supply) , 8-pin, M12x1 connector, A-coded (MultiPort) , 4-pin, M12x1 socket, D-coded (LAN)
Material		
Housing		ABS + PC + Aluminum

Laserlabel

Accessories

V1SD-G-2M-PUR-ABG-V45-G
Connection cable, M12 to RJ-45, PUR cable 4-pin, CAT5e

V1SD-G-5M-PUR-ABG-V45-G
Connection cable, M12 to RJ-45, PUR cable 4-pin, CAT5e

V1SD-G-ABG-PG9
Cable connector, M12, 4-pin, D-coded, shielded, non pre-wired

V1-G-2M-PUR
Female cordset, M12, 4-pin, PUR cable

V1-G-5M-PUR
Female cordset, M12, 4-pin, PUR cable

V1-G-BK5M-PUR-U
Female cordset, M12, 4-pin, PUR cable

V1-W-2M-PUR
Female cordset, M12, 4-pin, PUR cable

V1-W-BK5M-PUR-U
Female cordset, M12, 4-pin, PUR cable

V17-G-2M-PUR
Female cordset, M12, 8-pin, shielded, PUR cable

V17-G-5M-PUR
Female cordset, M12, 8-pin, shielded, PUR cable

V1S-B
Blind plug for M12 sockets

MH-R2000
Mounting aid for R2000 series, Quick clamp and adjustment system

PACTware 4.1
FDT Framework

Other suitable accessories can be found at www.pepperl-fuchs.com

Release date: 2017-09-21 14:30 Date of issue: 2017-09-21 253403_eng.xml