

# Investigating Simultaneous Localization and Mapping for AGV systems

With open-source modules available in ROS

Master's thesis in Computer Systems and Networks

ALBIN PÅLSSON  
MARKUS SMEDBERG



MASTER'S THESIS 2017

# Investigating Simultaneous Localization and Mapping for AGV systems

With open-source modules available in ROS

ALBIN PÅLSSON

MARKUS SMEDBERG



Department of Computer Science and Engineering  
CHALMERS UNIVERSITY OF TECHNOLOGY  
UNIVERSITY OF GOTHENBURG  
Gothenburg, Sweden 2017

Investigating Simultaneous Localization and Mapping for AGV systems  
With open-source modules available in ROS  
ALBIN PÅLSSON  
MARKUS SMEDBERG

© ALBIN PÅLSSON, 2017.  
© MARKUS SMEDBERG, 2017.

Supervisor: THOMAS PETIG, Computer Science and Engineering  
Advisor: MIKAEL BJÖRN, Kollmorgen Automation AB  
Examiner: OLAF LANDSIEDEL, Computer Science and Engineering

Master's Thesis 2017  
Department of Computer Science and Engineering  
Chalmers University of Technology and University of Gothenburg  
SE-412 96 Gothenburg  
Telephone +46 31 772 1000

Cover: Visualization of SLAM generated map.

Typeset in L<sup>A</sup>T<sub>E</sub>X  
Gothenburg, Sweden 2017

Investigating Simultaneous Localization and Mapping for AGV systems

With open-source modules available in ROS

ALBIN PÅLSSON

MARKUS SMEDBERG

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

## Abstract

The purpose of this project is to investigate solutions for Simultaneous Localization and Mapping (SLAM) in the context of Automated Guided Vehicles (AGVs). This thesis presents implementation details of a prototype system for AGVs, which was developed with the intention of allowing application-specific testing of SLAM.

Three of the most prominent open-source SLAM algorithms, available in ROS, have been evaluated and critically compared. Furthermore, basic background and explanation of the critical problems of SLAM are presented. The SLAM algorithms have been evaluated based on resulting map quality as well as resource requirements. Map quality is tested based on both visual comparison with a ground truth and the correctness of estimated distances in the map. In addition to this, results are presented from tests which have been conducted in order to test the SLAM-generated maps in AGV application areas. This includes tests where critical tasks, such as the robot's precision while navigating with the use of a SLAM-generated map, has been assessed.

The presented prototype system is based on the Robot Operating System (ROS), which includes state-of-the-art libraries and tools for robotic navigation. The prototype enables testing of navigation and mapping software available in ROS by publishing sensor data, from a physical AGV. The implemented software publishes readings from a laser range scanner, wheel encoders and a gyroscope. These sensor readings are published in a standardized way, making them accessible to applications in the ROS framework. The presented results from the tests answers the question of whether or not SLAM is suitable in the intended environment.

Keywords: AGV, Navigation, Robot Operating System, ROS, Automation, Simultaneous Localization and Mapping, SLAM, NDT



# Acknowledgements

This Master thesis has been carried out under the Department of Computer Science and Engineering at Chalmers University of Technology. We want to express our deep gratitude towards our academical supervisor Thomas Petig for his support, proposed ideas and feedback on technical writing.

We thank Kollmorgen Automation AB for providing the thesis topic, a workplace and AGVs for testing. Furthermore, we would like to express gratitude toward all employees at Kollmorgen for a friendly and welcoming atmosphere. We would also like to give special thanks to Mikael Björn for his technical guidance, willingness to help and enthusiasm towards the thesis topic.

Markus Smedberg and Albin Pålsson, Gothenburg, June 2017



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Related work . . . . .	2
1.2	Our contribution . . . . .	3
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Robot Operating System . . . . .	5
2.2	Simultaneous Localization and Mapping . . . . .	8
2.2.1	Problem definition . . . . .	8
2.2.2	Bayes filter . . . . .	9
2.2.3	Basic SLAM paradigms . . . . .	10
2.3	The Normal Distribution Transform . . . . .	10
<b>3</b>	<b>Implementation</b>	<b>13</b>
3.1	System background . . . . .	13
3.1.1	Hardware . . . . .	14
3.1.2	Software . . . . .	15
3.2	Software design . . . . .	16
3.2.1	ROS SLAM node . . . . .	18
3.3	Map conversion . . . . .	20
<b>4</b>	<b>Evaluation</b>	<b>23</b>
4.1	Mapping . . . . .	23
4.1.1	Map alignment . . . . .	24
4.1.2	Manual measurements . . . . .	25
4.1.3	Performance and system requirements . . . . .	26
4.2	Navigation . . . . .	28
4.2.1	Repeatability in positioning estimation . . . . .	29
4.2.2	Repeatability in automatic driving . . . . .	32
4.2.3	Navigation level . . . . .	33
<b>5</b>	<b>Discussion</b>	<b>37</b>
5.1	Software implementation . . . . .	37
5.1.1	Sensor publisher node . . . . .	37
5.1.2	Map converter . . . . .	38
5.2	Test results . . . . .	41
5.2.1	Mapping . . . . .	41
5.2.2	Navigation . . . . .	45

## Contents

---

<b>6 Conclusion</b>	<b>47</b>
<b>Bibliography</b>	<b>48</b>
<b>A Appendix 1</b>	<b>I</b>

# 1

## Introduction

An Automated Guided Vehicle (AGV) is a robot which navigates using techniques such as laser scanning or markers on the floor. Today's AGVs are most commonly used in environments such as warehouses, since they provide an effective and cost efficient way to move objects around. The application areas for AGVs are expanding which means that the need for new features and better performance is increasing.

The use of autonomous vehicles in industrial environments has been increasing in recent years since it potentially boosts efficiency and also reduces the amount of manual work required. These vehicles need to be able to position themselves reliably in order for this to work efficiently, which makes positioning a key feature for AGV systems. The most common way to achieve this is using additional infrastructure in the existing environment. The installation of such a setup is however time consuming and additional manual work is required if the environment changes. Research within the area of alternative map-based localization methods is therefore crucial for future development. By using a technique called Simultaneous Localization and Mapping (SLAM) it is possible to automate the process of surveying the environment and significantly reduce the time and cost of mapping processes [13].

SLAM is the computational problem of mapping an unknown environment while at the same time finding and keeping track of the operators location within this map. SLAM in two dimensions has been frequently used in research for several years but have not yet been widely accepted in industrial systems [27].

Robot Operating System (ROS) [22] is an open-source framework useful when building software for robot research. It contains libraries and tools which simplify the process of creating and connecting complex, but modular distributed robot systems. The ROS framework has proven to be powerful when developing applications for automated vehicles, specially prototype work requiring simulations and debugging [12]. ROS provides an opportunity for companies working with e.g., AGVs to open up their robots as a platform for research. This allows for research on new algorithms or applications for the existing systems with minimal knowledge of the inside of the system. It also allows integration of external software which have been created by experts outside of the company. This can simplify development and testing of new approaches and features.

The purpose of this project is to evaluate SLAM by developing and presenting a proof of concept system, based on ROS, with hardware and use cases similar to those used for AGVs in the industry. The implemented software of this project will be an extension to an already existing AGV system where ROS will serve as a foundation for the implementation. The performance of this setup will then

be evaluated in order to provide a starting point for further investigation whether SLAM is a feasible solution for tasks related to AGVs or not. Limitations found will contribute to direct further research. The focus of this project is mapping and navigation, but the project also indirectly includes a small-scale evaluation of ROS focusing on some of the existing stacks and tools, as an environment for testing and research.

With a system set up, using primarily a range scanner as input, several different SLAM algorithms will be tested and evaluated. These algorithms are evaluated first and foremost by the quality of the resulting map. The goal is to distinguish which SLAM-techniques give the best performance, but also to look at whether the results are promising for industry use or not. The tests include visual inspection as well as manual distance measurements. Furthermore, CPU load and memory usage are important parameters to consider in addition to the quality of results of the algorithms. The intent is to also further present which techniques for mapping and navigation that show most promise in order to direct future research. Although the tested SLAM algorithms share similarities they still have some differences which are worth analyzing.

As previously mentioned, positioning is one of the key features of AGV systems. In order to evaluate the suitability of SLAM in this environment, additional tests have therefore been conducted to determine the precision of positioning using SLAM generated maps. The tests are meant to determine if SLAM solutions meet the demands of real world use cases. These demands include low computational costs while providing near real-time execution and high precision. The most important metric when it comes to positioning is repeatability, which is the ability to position the vehicle at the same real-world coordinate over and over again. Further tests have been performed, using existing positioning applications, which measure repeatability with constructed movement patterns and environments which resembles real world use cases.

### 1.1 Related work

Indoor localization is an important topic in today's research since it can be used in a wide variety of systems, such as AGVs. There exist solutions which can be considered infrastructure-free, for example those using existing WiFi or GSM [17] for localization. Other solutions, using techniques such as Ultra Wideband (UWB) radio or RFID, require beacons or special infrastructure but are generally more accurate. For this thesis we will mainly consider the use of Light detection and ranging (LIDAR), where a scanner gathers distance measurements of the surrounding objects by sending laser beams. LIDAR measurements can be used for localization by comparing the scans to a predefined map. This technique is commonly used for indoor localization for AGVs since it provides high precision measurements without being too costly. In this project the LIDAR will also be used for SLAM.

There are various variants of SLAM techniques and implementations [29]. SLAM algorithms are often developed with a specific goal in mind, where metrics like accuracy, robustness or processing time are prioritized at various extents. The paper "A flexible and scalable SLAM system with full 3D motion estimation" [15] presents

an algorithm which was developed with the aim of allowing sufficient mapping and localization while keeping computational costs low. Meanwhile, the recently released paper "Real-time loop closure in 2D LIDAR SLAM" [13] presents a method for reducing the computational costs of computing loop closures. This potentially allows for mapping of large areas. The need to analyze strengths and weaknesses of SLAM algorithms is obvious.

While there exist research on evaluation of different algorithms, comparisons between algorithms are rare. There exist a few different techniques for evaluating SLAM algorithms. The paper "An evaluation of 2D SLAM techniques available in Robot Operating System" [25] compares the most popular SLAM algorithms available in ROS as an overview of strengths and weaknesses to define guidelines for other users. The quality of the maps are compared by looking at the error between the generated map and the ground truth. Sometimes, simple visual comparison of the result is considered to be sufficient while another common technique is based on calculating the error for the estimated trajectory during SLAM [16].

Tests on SLAM algorithms are commonly performed both in simulated environments and with physical robots. Furthermore, most existing SLAM evaluations are conducted on standard data sets, while focus will be at evaluating SLAM specifically for AGV use in the project, with more specific test cases and environments in mind.

SLAM research targets a variety of application areas. There exist research which is primarily focused on the AGV industry such as the paper called "Graph SLAM based mapping for AGV localization in large-scale warehouses" [2] where a SLAM solution for warehouse AGV's is implemented. A solution based on using mounted reflectors in addition to range measurements is presented in this paper, which differs from what has been done in our thesis.

## 1.2 Our contribution

This thesis presents an implementation of a ROS-based mapping and navigation system for an embedded vehicle controller. The project is done in collaboration with Kollmorgen Automation, a company developing AGV systems. Kollmorgen provided the vehicle controller, as well as a test vehicle which has been used in this project. The design choices and configurations of this implementation are detailed. The implementation is added on top of existing vehicle software in order to allow for extensive testing. Furthermore the resulting setup is evaluated.

The most prominent SLAM algorithms existing for ROS are evaluated and compared, both to each other and to a ground truth. The algorithms are compared based on resulting output as well as CPU and memory usage. The comparisons in this project will include the Cartographer algorithm [13] which has shown promising results but which has not yet, to our knowledge, been compared to the most prominent SLAM algorithms available in ROS.

The SLAM-generated maps are also tested together with existing navigation software in order to evaluate their usefulness in AGV systems. These tests are constructed to test the vehicles precision in navigation and position estimation.

## 1. Introduction

---

In order to use existing software, a map converter was developed. This converter enables SLAM-generated maps to be used with existing navigation tools.

The project is limited to testing existing SLAM and navigation-algorithms available in ROS. Related settings and configurations will be explained but no development of algorithms for these tasks has been done.

# 2

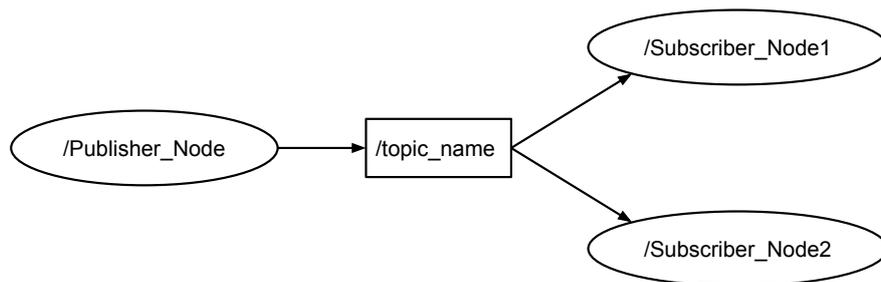
## Background

This chapter provides background knowledge which helps the reader understand the work presented in this thesis. The knowledge required for this project includes information about the ROS framework as well as details regarding the SLAM problem.

### 2.1 Robot Operating System

Robot Operating System (ROS) is an open-source framework useful when building software for various kinds of robots. It contains libraries and tools which simplify the process of creating and connecting complex robot systems [22]. ROS is specifically useful as its modular structure allows known well-debugged code to run alongside other code currently being written. The structure also allows for easy debugging of a single node.

ROS software is organized into packages. These contain modules of software such as nodes or libraries. A robot system is supposed to operate using a network of processing nodes where each node controls one smaller task of the system. Figure 2.1 shows an example of a ROS setup where nodes are displayed as ellipses.



**Figure 2.1:** Illustration of an example ROS setup with Publisher and Subscriber nodes sharing data via a topic. Both Subscriber nodes receive all data published by the Publisher node.

The nodes are connected in the network by topics or services. Topics are buses which can be used to transmit messages between nodes. Generally, nodes are not aware of who they are communicating with. Instead, a node publishes information onto a topic, where anyone interested can subscribe to all messages published to it. Topics are shown as rectangles in Figure 2.1 with a topic name

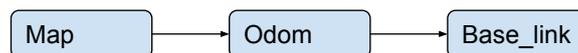
(/topic\_name). Topics are intended for streaming communication and use TCP or UDP based transport. Each topic has a message type bound to it which specifies the format of sent and received messages.

Sometimes it is more useful to let a node send a request for a message rather than subscribing to a topic. ROS Services supplies a request and reply structure for nodes. A Service is defined by one message type for requests and one for replies. A node provides a service by listening for the expected request and replying when asked. ROS Message types are used to simplify communication between nodes. Descriptions for messages are stored in ROS packages and detail what data is sent within a message.

A ROS system requires a Master, which is included in the roscore package. This master matches publishers and subscribers to topics and services. The Master provides a "lookup" which allows for nodes to find each other. Nodes can ask the master for a publisher or subscriber of a certain topic and the nodes can then communicate directly with each other.

In robotics, it is important for the robot to be aware of where it is in relation to the rest of the world. It is also crucial that the software is aware of the different locations and positions of connected sensors. All sensors use their own coordinate frames and the job for the robot is then to transform the frames in order to be able to compare data from different sensors in the same coordinate frame. TF is a ROS library which provides a way to track different coordinate frames and the transforms between those [10]. The library lets the developer use data in the desired coordinate frame without knowledge of all the frames in the system.

The transform class supports rigid body transforms. A transform is simply a message which contains a translation and rotation, providing a relation between two frames. Figure 2.2 shows a basic topology of a transform tree where the boxes represent frames and the arrows are transforms. In order for a system to work correctly, there must exist a path which connects all frames.

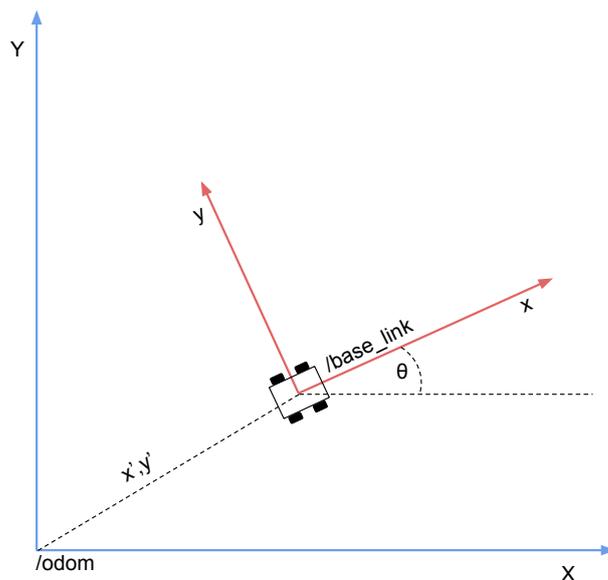


**Figure 2.2:** Basic example of TF tree structure with three frames connected by transforms.

Odom and Base\_link are commonly used frames and their relations are illustrated in Figure 2.3. The Base\_link frame is rigidly attached to the robot base and the Odom frame typically has its origin at the starting position of the robot. The transform from the Base\_link frame to the Odom frame specifies what translation and rotation that represents the robots position in the Odom frame. As an example of this, lets consider a point  $P_{base\_link}$  with coordinates in the Base\_link frame. The position of this point in the Odom frame  $P_{odom}$  can then be calculated using equation 2.2.  $T_b^o$  is the transform between Base\_link and Odom frame and can be represented using the rotation  $\theta$  and translation  $(x',y')$  as seen in equation 2.1.

$$T_b^o = \begin{pmatrix} \cos\theta & -\sin\theta & x' \\ \sin\theta & \cos\theta & y' \\ 0 & 0 & 1 \end{pmatrix} \quad (2.1)$$

$$P_{odom} = T_b^o P_{base\_link} \quad (2.2)$$



**Figure 2.3:** Example of relation between Base\_link and Odom frames.

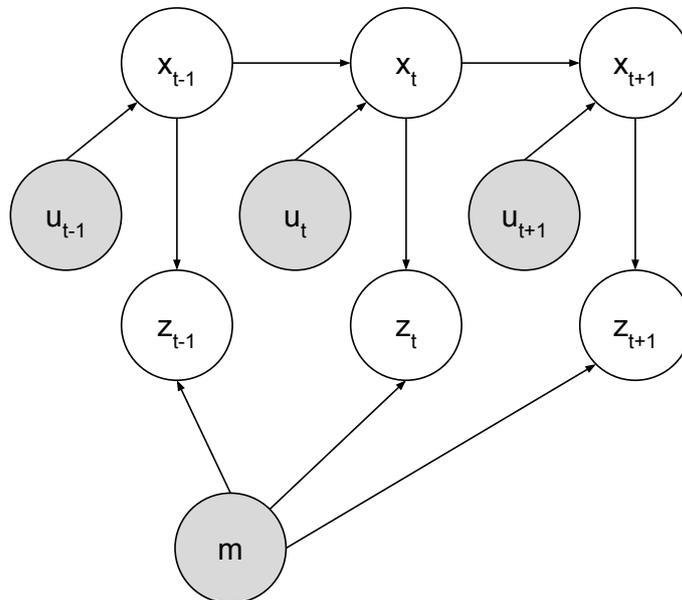
Poses in the Odom frame needs to be continuous and are often provided by a source such as wheel encoders or an inertial measurement unit. Poses in the Odom frame provides good short-term positioning but the drift makes it a bad source for long-term position references which introduces the need for other "world-fixed" frames. The Map frame is commonly used in robotics, with a transform towards the Odom frame. This transform, similar the one illustrated in Figure 2.3, specifies how the Odom frame is placed within the global Map frame. By traversing the TF tree and applying both the transform from Base\_link to Odom  $T_b^o$  and from Odom to Map  $T_o^m$  one can obtain the global Map coordinate of the Base\_link  $T_b^m$ , ie. where the robot is in the map frame;  $T_b^m = T_b^o * T_o^m$ .

## 2.2 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is the concept of letting a mobile robot generate a map of an unknown environment and at the same time use this map to calculate its position [8]. The SLAM problem exists when a robot does not have any information regarding a map nor its position and all that is given is measurements and control [28]. All information is calculated during run-time and no prior knowledge of the environment is required. To conclude, SLAM serves two different purposes: creating an accurate environment map and calculating and keeping track of the robots location over time. The SLAM problem is seen as one of the most important topics to master when developing truly autonomous mobile robots [29]. It is significantly more difficult than most robotic problems due to the fact that the map and poses have to be estimated along the way [28].

### 2.2.1 Problem definition

Figure 2.4 depicts the foundational variables of a SLAM system and the relation between them. The notion  $u_t$  represents the relative motion between time  $t-1$  and time  $t$ , which is called odometry. Odometry is stored for each time  $t$  and is often derived from the robots wheel encoders or motor control and represents a relative motion between two points in time.



**Figure 2.4:** Graphical visualization of variables and their relations in the SLAM problem.

It is necessary to have measurements of the environment when performing SLAM. A common type of measurement sensor is LIDARs, which uses laser rays to measure the distance to the closest object in all directions. These measurements form point clouds representing the environment and are stored as  $z_t$  for all times.

In SLAM it is also necessary to estimate a sequence of the robot's positions during execution. These positions are denoted as  $x_t$ , where  $t$  represents a point in time. Each position is typically represented as a three-dimensional vector which contains a two-dimensional coordinate and a rotational value to represent the robots orientation. Each position  $x_t$  has relations to the previous and the following position. The odometry measurements  $u_t$  represent the motion between times  $t - 1$  and  $t$ . Measurements  $z_t$  are associated with each position,  $x_t$ , in time as well.

The final variable,  $m$ , introduced in Figure 2.4 represents the *true* map of the environment, that is, how the environment actually looks. The SLAM problem is then defined as to estimate  $m$  and the sequence of positions  $x_t$ , only using measurements and odometry data. One especially difficult part of the SLAM problem is loop closures, which refers to the situation when the robot returns to an already visited position and re-observes landmarks.

### 2.2.2 Bayes filter

Bayes filter is the most general algorithm for estimating beliefs over states  $x$ , where a state is a collection of all information related to the robot and its environment. A belief represent the robots internal knowledge of its position, which is not the same as the true state. The general idea is to calculate a belief  $bel(x_t)$  of time  $t$  recursively based on the previous belief  $bel(x_{t-1})$  together with measurement  $z_t$  and control data  $u_t$  [28]. The general Bayes filter algorithm can be found in Algorithm 1.

The algorithm iterates over each hypothetical state  $x_t$ . The belief  $bel(x_t)$  for time  $t$  is estimated in two steps. First a prediction step is executed, seen in line 2. This step calculates a belief of  $x_t$  purely based on the previous belief  $bel(x_{t-1})$  and the control  $u_t$ . This is done by integrating over the product of two distributions. The first being the probability that control  $u_t$  moves the robot from  $x_{t-1}$  to  $x_t$  and the second the belief of state  $x_{t-1}$ . This step spreads the resulting state distribution due to noise from control.

The state distribution is tightened by using measurement data in the second step of the algorithm. This step is called the update step and is found in line 3 of Algorithm 1. This step updates the estimated belief, calculated in line 2, with the probability that measurement  $z_t$  have been observed at  $x_t$ . Furthermore the result has to be normalized, by multiplication of the normalization constant  $c$ , in order to correctly represent a probability. By providing an initial belief as a base case for the recursion, one can use Bayes filter to solve the task of localization for a robot. It is important to note that this specific algorithm is only applicable to simple estimation problems as it loops over  $x_t$ , which is continuous and can take an infinite number of values.

---

#### Algorithm 1 General Bayes filter algorithm

---

```

1: for all  $x_t$  do
2:    $bel'(x_t) = \int (p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx$ 
3:    $bel(x_t) = c p(z_t | x_t) bel'(x_t)$ 
4: end for
5: return  $bel(x_t)$ 

```

---

### 2.2.3 Basic SLAM paradigms

Solutions to the SLAM problem continuously estimates the robots state based on control, measurements and old estimates often in a similar way to the approach of the Bayes filter. Furthermore S. Thrun and JJ. Leonard [29] present three main paradigms that most SLAM algorithms follow, or are derived from. These three are Extended Kalman Filter (EKF) SLAM, Particle filter based SLAM [7] and Graph-based SLAM [18].

EKF based SLAM is historically the earliest and probably the most influential SLAM paradigm [28]. We refrain from giving an extensive background on the EKF except by saying that it is based on Bayes filter, utilizing Taylor linearization [26]. Naive implementations of this paradigm tends to be computationally costly since the computational cost grows quadratically with the number of observed landmarks [8]. However there exist variants of EKF-SLAM which can process large amount of landmarks efficiently. This SLAM solution also inherits the same problems as standard EKF solutions where one problem is that it applies linearized models to non-linear motion and observations.

The second paradigm presented by S. Thrun and JJ. Leonard is based on particle filters [19]. This was the first concept of representing a non-linear process model [8]. Particle filtering is another way of calculating probabilities of a process states with partial or noisy observations. Each particle in the system holds an estimate of the state of the system. By combination of a set of particles it is possible to achieve a good estimation of a map which realistically represents the reality. The SLAM problem makes direct implementation of particle filters impractical, as they tend to grow extremely large in numbers, and the main challenge with this approach is then to limit the number of particles. The particle filter based solution was designed to make improvements to the speed of the EKF-based algorithms [20].

The third paradigm relies on the fact that the SLAM problem can have a graph-based representation. The basic idea is that observations and robot locations can be stored as nodes. The edges represent a relation between nodes, which can be either the motion between two positions or the position at which an observation was made. The edges can be seen as constraints and by relaxing them one can retrieve the best estimate for the map and trajectory. This approach has the advantage over EKF paradigm in regards of required space and update time as inserting nodes in the graph requires constant time and the required memory grows linearly with the size of the map. [29]

## 2.3 The Normal Distribution Transform

Scan matching is a SLAM method which is based on being able to match two scans or match a scan to a map. The paper "The Normal Distributions Transform: A New Approach to Laser Scan Matching" [3] introduces the Normal Distribution Transform (NDT) and presents a new approach to scan matching.

NDT transforms a scan (2D points) into a grid  $\mathbb{Z}^2$  where every cell is associated with a normal distribution. In practice this means that each cell has a mean vector and a covariance matrix and the result of the transformation is a

continuous probability density for the 2D plane which can be used for scan matching. Creation of an NDT map works as follows: first, the modeled 2D space is divided into cells of equal size. Then, by gathering all scan points inside a cell, one can calculate the mean and covariance of the data inside that cell. This process is mathematically explained below.

- Gather all scan points  $\{x_i\}, i \in \{1, \dots, n\}$ , inside cell  $c$ .
- Calculate the mean  $q_c = \frac{1}{n} \sum_{i=1}^n x_i$ .
- Calculate the covariance matrix  $\Sigma_c = \frac{1}{n} \sum_{i=1}^n (x_i - q)(x_i - q)^t$ .

The probability of measuring a point  $x$  within cell  $c$  can then be modeled using the Normal Distribution:  $p(x) \sim N(q_c, \Sigma_c)$ . This means that a NDT grid represents the probability of measuring a point within a cell while for example an occupancy grid, which are commonly used, represents the probability of a cell being occupied. Furthermore the paper shows that this representation can be used for scan matching by creating a score, based on the distributions, and using one step of Newton's Algorithm for score optimization. Based on this, ideas of how to implement position tracking and SLAM using NDT is presented in the paper.



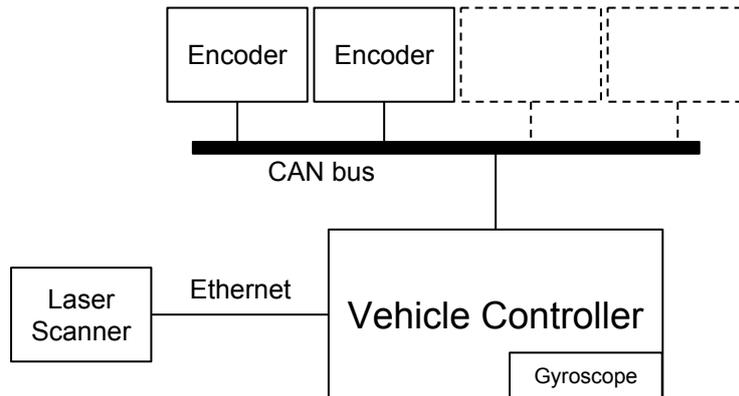


### 3.1.1 Hardware

Three types of sensors have been used for navigation. These types are Light Detection and Ranging sensors, referred to as LIDAR, wheel encoders and gyroscope. A LIDAR measures distances to closest objects in different directions by using laser rays. Wheel encoders reports how fast the wheel is rotating and what steering angle is applied to it. This information can be used for dead reckoning to calculate how far the vehicle has travelled from a starting point. A gyroscope measures rotational acceleration per axis.

The LIDAR used in the test setup is a LS2000, which is the same product Kollmorgen uses in most of their new systems. The LS2000 scanner provides an angular resolution of 1 milliradian at a rate of 20 scans per second and a range of 30 meters for wall measurements with the current setup.

Kollmorgens vehicle controller, called CVC600, consists of an embedded ARM Cortex-A8 processor running Linux, and this is where all the vehicle software is executed. The vehicle controllers main objective is to calculate the vehicles position as well as giving control commands to the vehicle based on centralized orders. Figure 3.2 depicts a system overview that illustrates how the hardware components are connected in the NDC8 system.

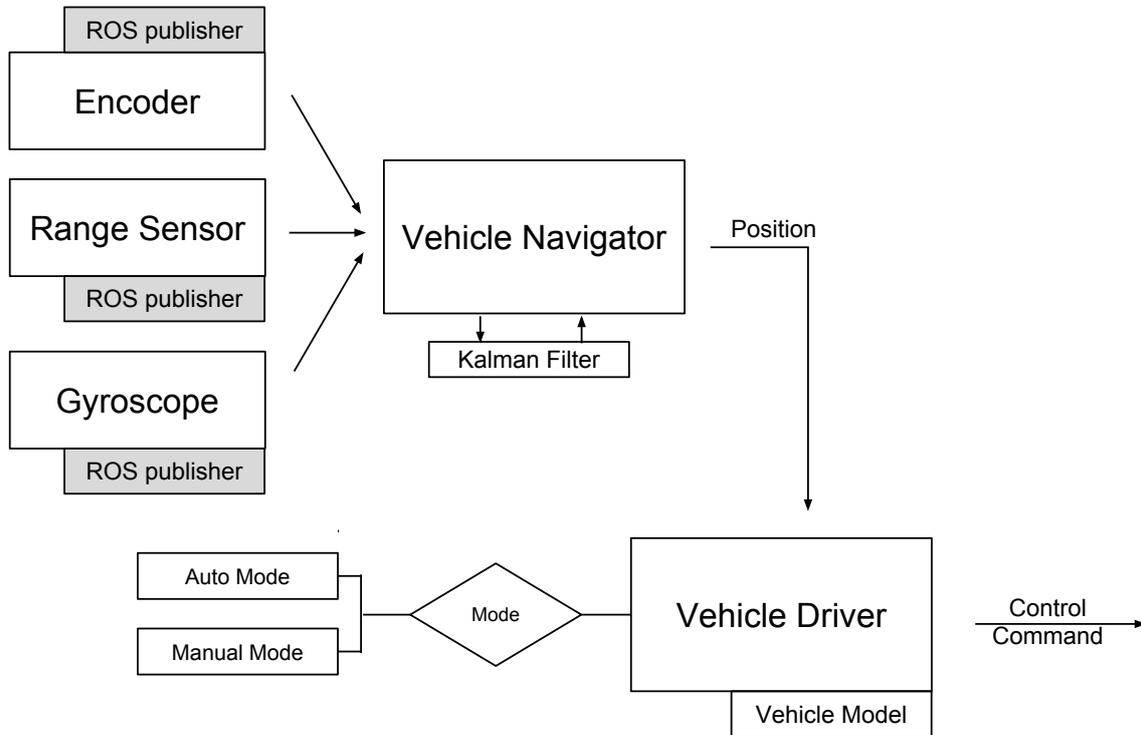


**Figure 3.2:** Hardware architecture overview. The laser scanner is connected through ethernet and the wheel encoders are connected via a CAN bus. The gyroscope is integrated in the vehicle controller.

The sensors are externally connected to the vehicle controller, with the exception of the gyroscope which is integrated in the CVC600. The wheel encoders are connected to the vehicle controller through the CAN bus and the LS2000 LIDAR is connected with ethernet and communicates using UDP packets.

### 3.1.2 Software

The NDC8 system includes a great amount of software, but only the parts regarding navigation are explained in this report. Figure 3.3 shows an overview of the vehicle controller software responsible for navigation.



**Figure 3.3:** Software architecture overview of the vehicle controller. Only the most vital parts to this thesis are included.

The information flow starts with the sensors, where we observe three software modules that handles one sensor each. Sensor measurements are collected and formatted for passing along to the rest of the system. The Vehicle Navigator includes a Kalman filter which is continuously updated with sensor reading in order to provide an estimated position. The NDC8 system supports multiple modes for position estimation. Two of these modes, Reflector navigation and Natural navigation, have been used in this project. Reflector navigation relies on mounted reflectors on the walls and provides high precision. These reflectors have known positions and as the vehicle observes multiple reflectors it can calculate its position by triangulation. The other supported navigation mode, Natural navigation, relies on a NDT map representation and performs a probabilistic scan matching of laser range measurement and NDT cells in the map. This navigation method utilizes existing physical obstacles, like walls and pillars, in the environment and does not require any installation of reflectors.

The estimated position is later passed to the Vehicle Driver module which, if in Auto Mode, calculates the control command necessary to drive to the desired destination. The vehicle model is taken into account as the Vehicle Driver calculates

the correct control command it takes. This enables the Vehicle Driver to handle different types of vehicles. In its current configuration, the vehicle controller software loop is driven at a rate of 16 Hz, with access to LIDAR, encoder and gyroscope values at each iteration. This rate provides some level of real-time reassurance and will be used to publish ROS messages. For this project, the software has been extended and does, in addition to previous functionality, publish all sensor readings to the ROS framework which is explained in Section 3.2.

The ROS installation is primarily targeting the Linux Ubuntu platform. Cross-compilation was required in order to be able to execute it on the embedded Cortex-A8. When building software for the NDC8 system, Kollmorgen Automation utilizes the build tool Yocto [24]. The Yocto Project is an open source collaboration project that provides templates, tools and methods to ease creation of custom Linux-based systems for embedded products regardless of the hardware architecture. There exist an open source Yocto layer, called meta-ros<sup>1</sup>, which conveniently enables integration of common ROS packages into the Yocto build.

Yocto recipes (.bb files) are fundamental components in the Yocto Project environment. Each software component requires a recipe to define the component in order to build. For the basic system setup, four ROS recipes were included. The *roslaunch* recipe was included to provide basic functionality such as running a roscore and using standard message types. Furthermore, the recipes *nav-msgs* and *sensor-msgs* are included to bring the message types required for the navigation stack. Last, a recipe for the *tf* package was included, providing access to transform function calls. The ROS libraries used in this project require 11.9 MB of disk space on the vehicle controller.

## 3.2 Software design

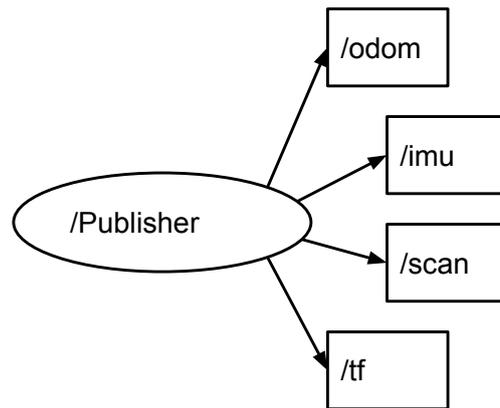
The implementation of ROS for this project was done following the conventions of the Navigation stack<sup>2</sup>. This stack contains the most common ROS packages for navigation tasks such as localization and mapping. It is also compatible with most ROS SLAM implementations. The stack is based on taking information from odometry and sensor streams and outputting velocity commands for the robot. The requirements for the use of the navigation stack includes setting up a correct TF tree and publish data using standardized message types.

The publishing of odometry, laser scan and gyroscope data has been embedded into the NDC8 code as explained in Section 3.1.2. The ROS Publisher software packages are represented in a single ROS node, *Publisher*, responsible for the transmission of all sensor measurements from the vehicle. Figure 3.4 shows the ROS *Publisher* node that publishes LIDAR data on the `/scan` topic, odometry on `/odom` and gyroscope measurements on `/imu`. The corresponding transforms for all three sensor types are published on the `/tf` topic.

---

<sup>1</sup><https://github.com/bmwcarit/meta-ros>

<sup>2</sup><http://wiki.ros.org/navigation>



**Figure 3.4:** Plot showing the implemented ROS node and published topics.

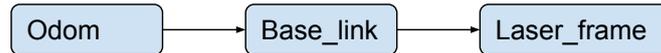
The encoder values, describing wheel motion, are accessible from the software and are used as a source of odometry data in this system. The navigation stack requires that odometry messages are published containing velocity information. It also requires that a transform between the odom frame and a fixed world frame is published, as seen in Figure 3.5. This transform indirectly contains the robots position in this fixed world frame calculated from the odometry data. The current vehicle velocity is calculated using the angles and rotation speeds from the wheel encoders. The velocity x-, y- and rotational velocity is directly published in the *odom.twist* message. The velocity is further used to calculate dead reckoning position required for the odometry transform. The rotation is calculated by accumulating the rotation velocities multiplied by the held time intervals. The translation is calculated in a basic and typical way using the current rotation together with the current x- and y-velocities and adding this value to previously calculated position.

The Imu ROS message type is the one most commonly used for gyroscope values and is therefore used in this project. At each software iteration a mean value of velocities around the z-axis, measured since last reading, is calculated. This mean value is published as the *angular\_velocity.z* value in the ROS Imu message. The standard deviation of the gyroscope values is estimated to be 0.05 rad/s and the variance published is then 0.025 rad/s. Since the z-rotation is the only IMU value to be used in this implementation, the IMU message needs to be constructed with some caution. The linear acceleration in the z direction is set to  $9.8m/s^2$  in order to represent gravity. Furthermore the variance for the unused parameters is set to a very high number in order to ensure that they are not used by any subscribing source.

The data produced by the LIDAR scanners fit the LaserScan ROS message well and this type is therefore used. To allow extensive testing, code was written to publish scan data with different parameters. The LS2000 LIDAR scanner provides an angular resolution up to 1 mrad and a frequency of 20 rotations per second. The developed software supports producing a reduced resolution and frequency to best fit any purpose. The transform from the laser scan frame to the *base\_link* frame

is simple since it only has to contain the static translation from the position of the scanner to the chosen `base_link`. However it is an important detail in order to get correct measurements from ROS applications.

The ROS navigation stack further relies on REP 105<sup>3</sup> for conventions regarding coordinate frames. This document specifies how to name frames, the preferred relationships between them and what authorities that should publish transforms. The TF tree used for this project is shown in Figure 3.5.



**Figure 3.5:** Illustration of the basic TF tree used in ROS implementation.

The physical position of the sensors in relation to the other parts of the robot is of great importance and is often expressed with the use of transforms. With the test vehicle used for this project, the gyroscope is placed in the center of the controller, which is considered the middle and `base_frame`. The wheels are located at a constant distance from the base frame and the exact position was measured. These positions are used in the velocity calculations where the result is a velocity for the base of the robot. The transform between `odom` and `base_frame` is therefore simply the dead reckoning position, as previously explained. The position of the scanner is however important to include in the TF tree. This distance was measured and is included as a static transform between the `base_link` and `laser_frame`, seen in Figure 3.5.

ROS allows for a distributed computing environment where the nodes communicate over a network. The ROS packages often use the system clock as the time source. ROS is not a real-time system and to ensure that correct transforms are used there are often constraints on the timestamps used. When running a ROS setup on several machines where the clock differ this causes trouble. The most common solution is to run a NTP server on each node. However the system used in this project does not support NTP. During testing, this problem was solved by manually synchronizing clocks over SSH as a quick fix.

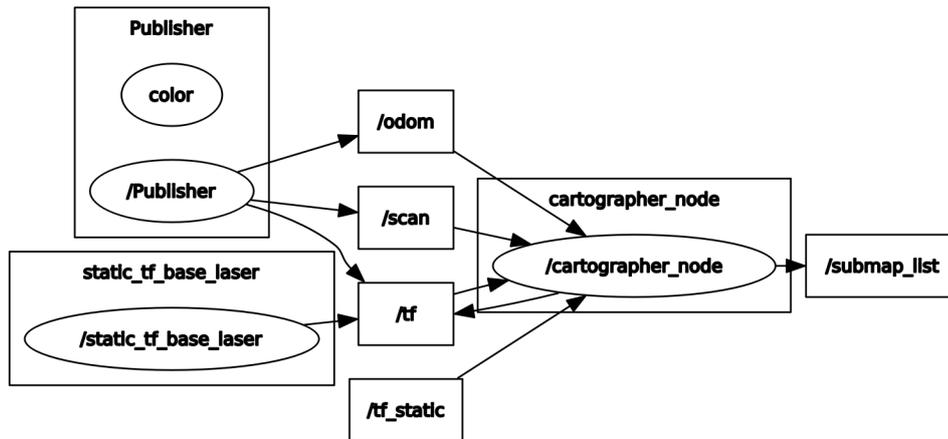
#### 3.2.1 ROS SLAM node

This section describes the system setup that was used to execute the different SLAM ROS nodes. The three algorithms that have been used in this project are Cartographer, Hector SLAM and GMapping. The sensor data used when performing SLAM is published as described in Section 3.2. Namely LIDAR measurements on the `/scan` topic and odometry on the `/odom` topic, together with required transforms on `/tf`.

Figure 3.6 depicts the system setup with the SLAM node included, in this case the Cartographer. We observe that the SLAM node subscribes to all topics that is published by the *Publisher* node. Based on the data provided on the three topics the SLAM node is able to generate a map and publish it on the `/map` topic

---

<sup>3</sup><http://www.ros.org/reps/rep-0105.html>



**Figure 3.6:** Illustration of ROS nodes and topics setup during Cartographer execution. Note that the setup is similar for all tested SLAM algorithms.

formatted as an OccupancyGrid. The occupancy grid ROS message type is used to represent a 2D map. In this format, the mapped area is divided into a grid where every cell has a value, between 0 and 100, defining the probability of that cell being occupied.

It is important to note that some of the SLAM modules may be used in various setups, utilizing different sensor data, calculating at different pace and using different coordinate frames. GMapping and Hector SLAM was executed with the default setup that is provided by the package itself. Whereas the Cartographer require some configuration. In order to run the Cartographer, a configuration file must be provided that specifies a number of parameters, such as tracking frames, frequencies and what sensor data is provided.



**Figure 3.7:** Illustration of TF tree when running SLAM node.

With a SLAM node connected to the system the transform tree has to be extended as seen in Figure 3.7. The relationships are mostly straightforward, with the exception of the relations between odom and map. The intuition would be to have both odom and map attached to base\_link, however since only one parent frame is allowed this is the recommended structure of the TF tree. Transformations from laser\_frame to odom is equal to earlier, as they are provided by the *Publisher* node. The TF tree has been extended with one transform, namely the transform from odom to map frame. This transform is provided by the SLAM algorithms and translates positions in the odom frame to corresponding position in the frame of the generated map. This transform represents as a correction of accumulated errors or

drift from the odometry source, where corrections have been calculated with the use of scan matching.

IMU data is used for a few tests. Imu is then used instead of the Odometry data and even though the sources do not provide the same type of data they can be interchanged in Figure 3.7 and Figure 3.6.

## 3.3 Map conversion

Kollmorgen's software tools and Natural navigation support maps with NDT cells and not occupancy grids, where the latter is the format supported by most ROS packages. Although there exist experiments of NDT SLAM algorithms, such as presented in "Normal Distributions Transform Occupancy Map fusion: Simultaneous mapping and tracking in large scale dynamic environments" [27] and "Generic 2D/3D SLAM with NDT maps for lifelong application" [9], these are not widely recognized and not implemented for ROS. One potential solution could be to use an existing occupancy grid slam algorithm and subsequently access the pose graph, which contains poses with corresponding scans, and building a NDT grid based on these. However this solution requires a lot of work and instead a map converter was created to enable quick and simple testing of SLAM maps.

The original idea for creating NDT grids, presented by Biber and W. Strasser [3], is to use all the gathered scan points to calculate the distribution within a grid cell in order to create the NDT map. The resulting occupancy grid from the ROS SLAM algorithms contains a probability of each cell being occupied but no data regarding the scan points used is available. This conversion is therefore by design flawed. However, the most important part of the NDT cell grid is that it represents real world objects and that kind of information is possible to translate from the occupancy grid to a NDT grid with a basic conversion.

The NDT cell grids most commonly used by Kollmorgen have a cell resolution of 30 cm since this is determined to be sufficient. The default setting for SLAM algorithms in ROS is to produce an occupancy grid with 5 cm resolution. This is something that was taken advantage of in the conversion by sampling the occupancy grid cells to create NDT cells. The basic idea of the conversion is to gather all 36 (6x6) occupancy grid cells which represent the same area as one NDT cell. The next step is to calculate the mean and covariance matrix of the occupied occupancy grid cells within that NDT cell. The occupancy grid cells are used in a similar way to how scan points are used in the original paper [3]. An occupancy grid cell contain a single value which represents the probability of that cell being occupied. The trick is then to chose which cells that are "occupied enough" to be considered to contribute to the mean and covariance of the corresponding NDT cell. Each cell has a value between 0 and 100 which represents its certainty of being occupied. The threshold for the conversion algorithm was set to 65% since this is the default value for the ROS module which was used to save maps. This means that a occupancy grid cell is contributing to a NDT cell only if it has a value greater than 65.

However, by only using the middle position of each occupancy grid cell as a point, the NDT cells might get somewhat flat. The map conversion algorithm was

extended to also use a point from each corner of a cell, if that point is not already included. This addition ensures that a single line of occupancy grid cells become a NDT cell which representing something with a depth. Below (Algorithm 2) is the basic pseudo code for the conversion.

---

**Algorithm 2** Pseudo code for map conversion algorithm.

---

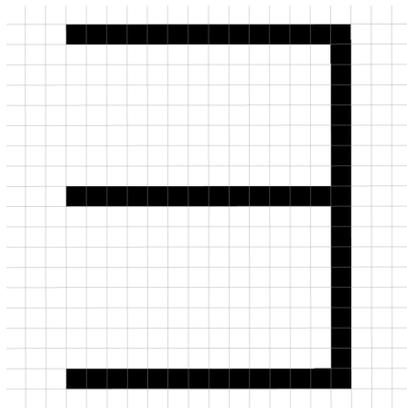
```

1: for all  $ndtCell_{x,y}$  in  $ndtGrid$  do
2:    $occupancyGridCells := getAssociatedCells(ndtCell_{x,y})$ 
3:   for all occupied  $occCell_{x',y'}$  in  $occupancyGridCells$  do
4:     add  $occCell_{x',y'}$  to set  $C$ 
5:   end for
6:    $mean_{x,y} := calculateMean(C)$ 
7:    $covariance_{x,y} := calculateCovariance(C, mean_{x,y})$ 
8: end for

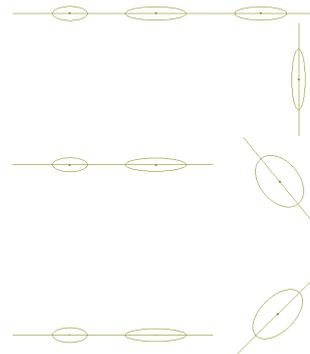
```

---

The conversion algorithm was implemented as a Python script. The script works as a ROS node which requests an occupancy grid map via the service provided by the map\_server ROS module in order to access stored maps. The occupancy grid ROS message contains meta data for the map as well as a simple array containing the map data in row-major order. The received map is first extended to have a sufficient number of cells for sampling. The occupancy grid cells are then converted as previously explained and the resulting NDT cells are written to a file with XML syntax. Figure 3.9 shows the resulting NDT cells using input as seen in Figure 3.8. The NDT cells are displayed using a visual tool showing the covariance as an ellipse around the mean position. A large scale NDT map created using this converter can be found in Figure A.8 in the appendix.



**Figure 3.8:** Mockup occupancy grid with 5 cm resolution.



**Figure 3.9:** Map in Figure 3.8 converted to NDT cells with 30 cm resolution.

### 3. Implementation

---

# 4

## Evaluation

In order to validate the performance of ROS SLAM and navigation, specific tests were planned. These tests are divided into two sections. In each section, tests are described and the results of those tests are presented. Section 4.1 presents tests regarding the map quality. These tests are mainly intended to test how well the SLAM generated maps represent reality but also to compare SLAM algorithms to each other as well as to test the resource requirements. Section 4.2 includes tests for positioning and navigation. These tests provide results regarding the usefulness of the SLAM generated maps specific to AGV industry use cases. This separation enables simpler testing and clearer results while conclusions still can be drawn regarding a potential use case where navigation is run simultaneous with SLAM. The separation also serves to produce results interesting to a separate use case, where SLAM would not be run at every vehicle but instead only be used to automate the creation of a static environment map.

During development, early tests were conducted to ensure correctness of individual sub functions. By using RViz<sup>1</sup>, a visualization tool, it was established that the odometry calculations based on encoder feedback were not perfect. Starting the vehicle from a static position creates a jump in odometry that is not correctly representative of the actual movement. Furthermore, turning the vehicle 360 degrees does result in a slight skew of scan angle perception. This behavior was however expected since encoder values and calculations are not believed to be perfect.

### 4.1 Mapping

In order to verify if the quality of maps, obtained by available SLAM algorithms, meets the requirements of today's industry, tests have been conducted to compare the maps. In these tests three different SLAM implementations, referred to as Cartographer, Gmapping and Hector Slam, were compared to each other.

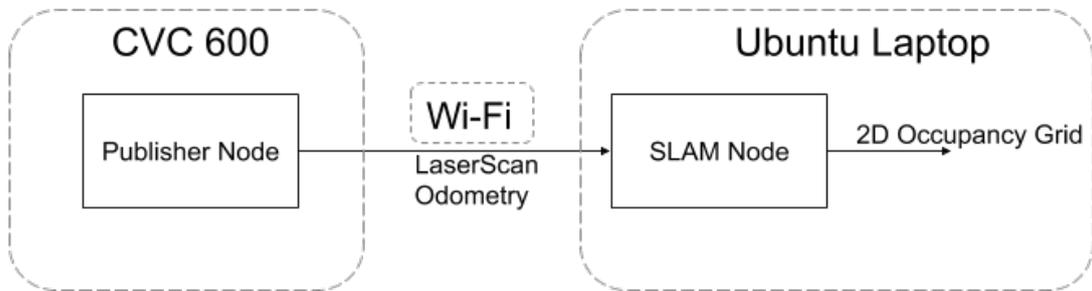
The algorithms were executed with the same input feed consisting of laser scan, odometry and gyroscope measurements. The three different sensor types were, however, used in different combinations by different iterations of the SLAM algorithms. The Cartographer and Gmapping were first run with LIDAR and odometry data. Then, for the sake of comparison, the Cartographer was run a second time using gyroscope instead of odometry data. Hector SLAM was also executed with LIDAR and gyrocope data as input.

---

<sup>1</sup><http://wiki.ros.org/rviz>

All SLAM algorithms were configured to use a grid cell resolution of 0.05 m and a laser cutoff range of 24 m. The test vehicle was run at a max speed of 0.5m/s when gathering data. The tests evaluate the precision of the resulting maps as well as algorithm performance.

The system setup used for obtaining the evaluated maps in this section is depicted in Figure 3.6. However, the different parts of the software are not deployed on the same hardware. The *Publisher* node is executed on the CVC 600 embedded system, publishing all sensor measurements over the ROS framework. The SLAM node on the other hand is deployed on an external system, in this case a laptop running Ubuntu Linux. Figure 4.1 illustrates the hardware that facilitates the two ROS nodes.



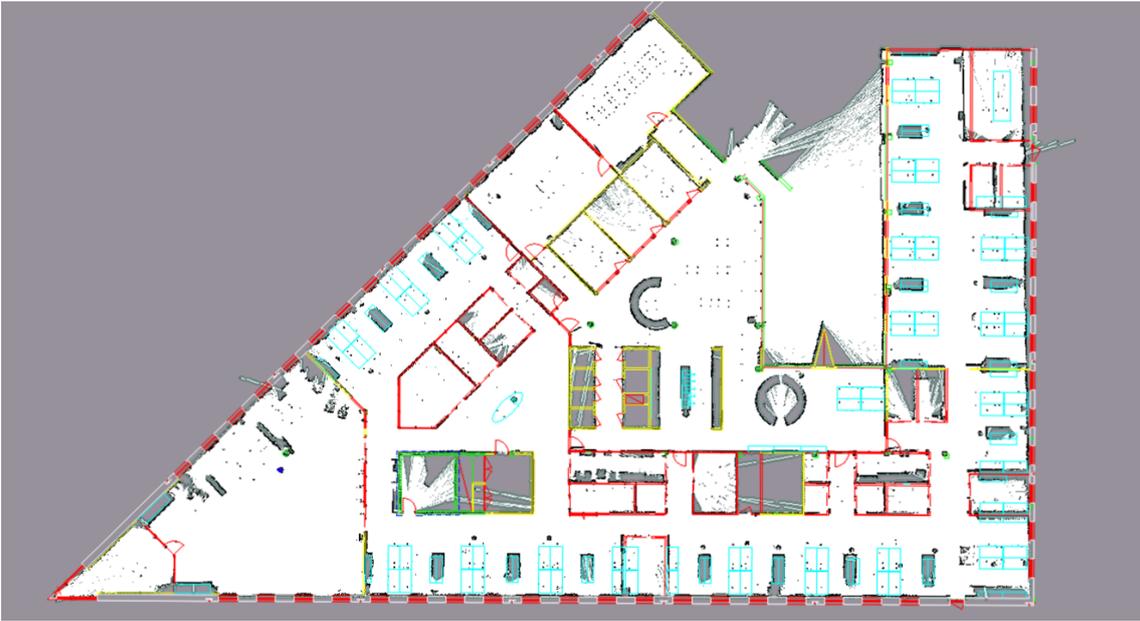
**Figure 4.1:** Illustration of the system setup used while executing SLAM.

The SLAM nodes all outputs a 2D Occupancy Grid on a ROS topic. Instead of subscribing to the topic and performing the tests on that data the maps were stored to make sure that the maps were static throughout the test process. Storage of the obtained maps published on a ROS topic is achieved with the *map\_server* module. The obtained map is binarized and stored as a row-major-order array in a .pgm file.

#### 4.1.1 Map alignment

First off, the generated maps were aligned with a Computer Aided Design (CAD) floor drawing for a visual comparison. Below, in Figure 4.2, the result of this alignment for the map created by Gmapping is shown. The results of the other algorithms can be found in the appendix in Figures A.1, A.2 and A.3.

Both maps created by the Cartographer fit the CAD drawing well. However in the map created by Gmapping, seen in Figure 4.2, it can be noticed that the right most part deviates from the reference drawing. The same kind of error can be observed in the map created by Hector SLAM as well, although not as significant.



**Figure 4.2:** Map from Gmapping aligned with CAD drawing. Areas in white and black are from the occupancy grid. The only contribution to the figure from the CAD drawing are the coloured walls.

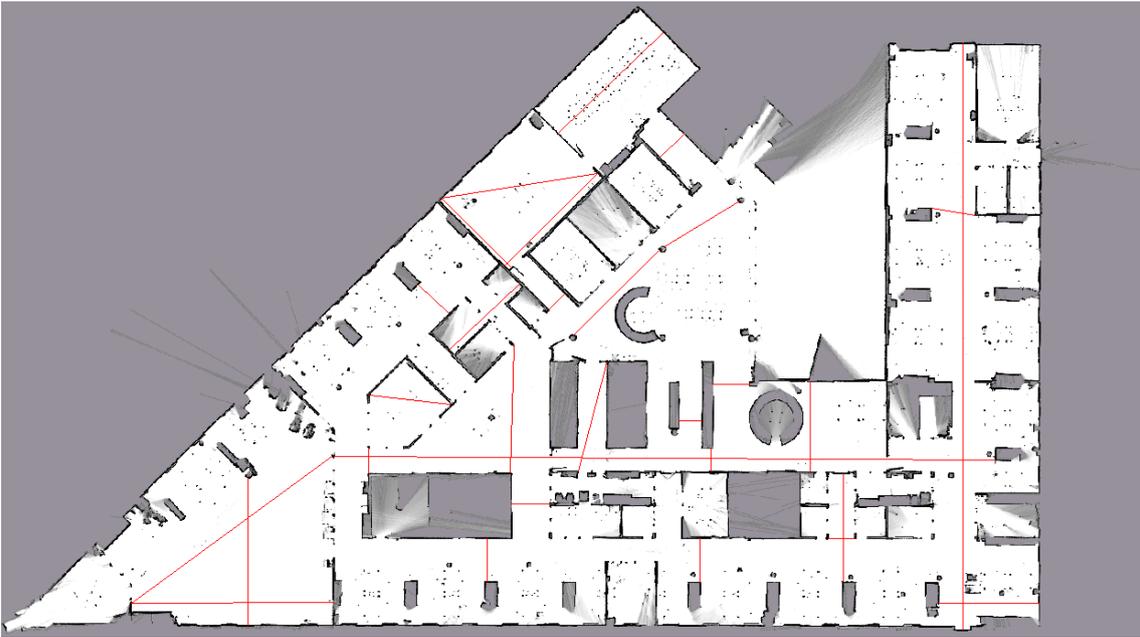
### 4.1.2 Manual measurements

A second test with aim to compare the obtained maps to the reality was performed. A combination of manual measurements and distance calculations in the map were used to calculate errors. The visualization software RViz provides tools for measuring distances between points in a map. With the use of a hand-held laser rangefinder, the Leica DISTO D210, the distance was measured in reality between the same reference points.

To achieve as exact results as possible the points of reference must be chosen wisely. The points used should preferably be unambiguously identified both in the occupancy grid as well as in reality. In our tests reference points have been chosen as distances between two straight walls, distance between corners and between clear landmarks such as pillars.

Figure 4.3 shows the points of measurements. Table 4.1 includes the results from the distance measurements comparisons between the real world and produced SLAM maps. All measurements used can be found in Table A.1.

The mean error shows the overall performance while the max error measurement indicates if an algorithm produced a serious error at some location. The Cartographer, both with gyroscope and odometry, has the best results in all metrics. Hector SLAM and Gmapping show worse results, even though the differences are small.



**Figure 4.3:** Map with marked (red) reference points used for manual measurements when evaluating the correctness of the maps. The red lines shows what distances that were measured.

**Table 4.1:** Results from map distance measurements for all tested SLAM algorithms. The errors are computed with respect to the manual measurements performed between the same reference points.

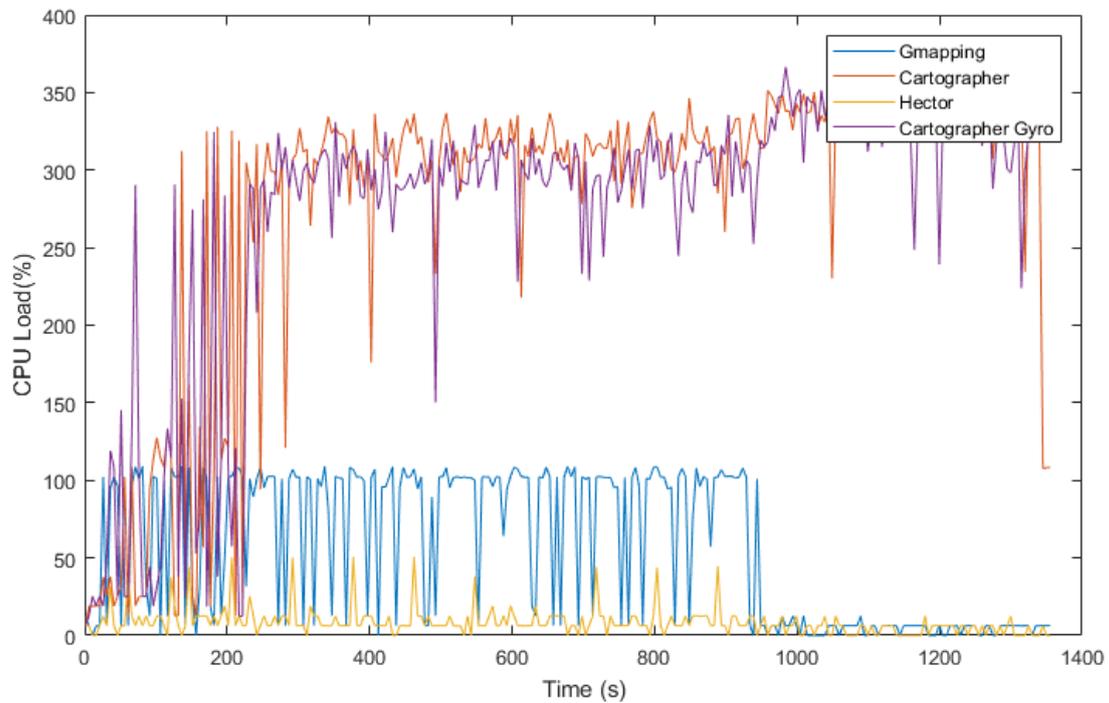
Algorithm	mean error (m)	max error (m)	SD (m)
Hector SLAM(gyro)	0.058	0.141	0.036
Gmapping(encoders)	0.049	0.183	0.039
Cartographer(encoders)	0.035	0.104	0.029
Cartographer(gyro)	0.036	0.110	0.023

### 4.1.3 Performance and system requirements

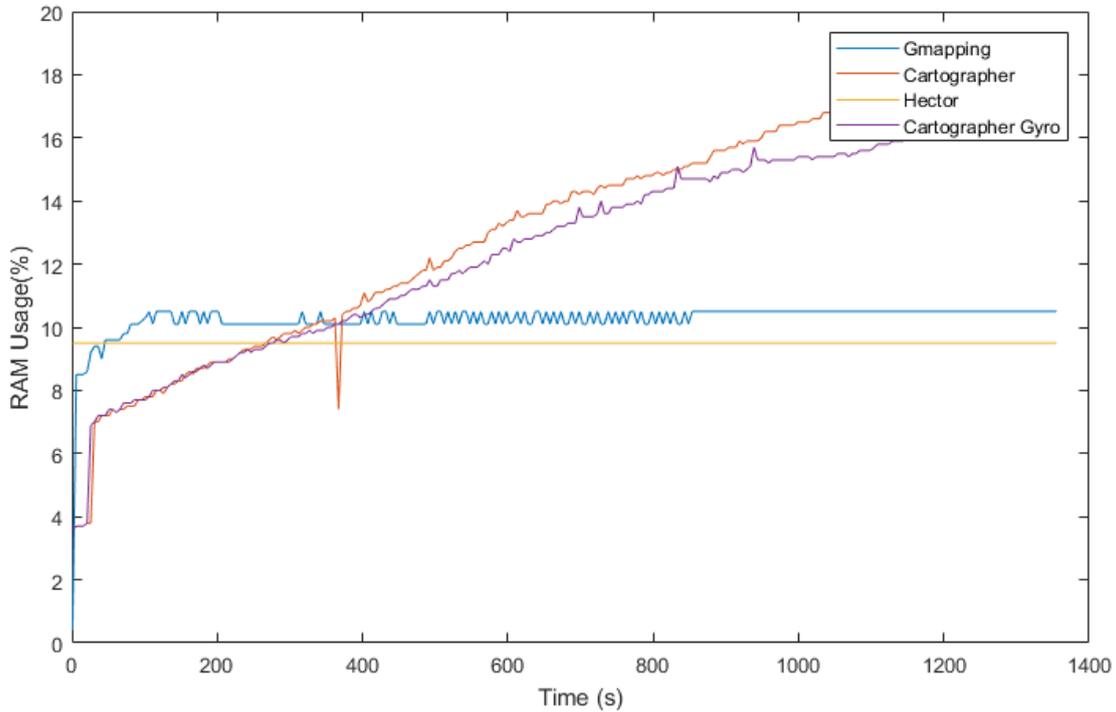
The computer used in all tests is a laptop running Linux Ubuntu 14.04 on a Intel Core i5-540M with 4 GB of RAM. During execution of every SLAM algorithm the CPU load and memory usage was logged in order to draw conclusions about performance and system requirements.

A script which logs the CPU and memory usage was executed when the maps, seen in Section 4.1.1, were built. Figure 4.4 shows the CPU load during SLAM. It should be noted that the recording of measurements only lasted for about 900 seconds. The results show that Cartographer uses significantly more CPU power than the other algorithms and that it continues to optimize even after all sensor data is received. It is also clear that Gmapping is single threaded and hits the maximum CPU load at several occasions. Hector SLAM is only using a small portion of the available CPU power.

The memory usage during SLAM is shown in Figure 4.5. Hector uses a static amount of memory which is approximately equal to the amount used by Gmapping. The Cartographer has continuously increasing memory usage and keeps increasing even after it stops receiving sensor data.



**Figure 4.4:** The plot shows the CPU load logged running the different SLAM algorithms on the same input data. Note that sensor readings stopped after approximately 900 seconds. The Cartographer shows the greatest CPU load, with an average of approximately 300%. Gmapping and Hector is measured to approximately 100% and 10% CPU load respectively.



**Figure 4.5:** The plot shows the memory usage logged running the different SLAM algorithms on the same input data. Note that sensor readings stopped after approximately 900 seconds. The Cartographer shows a continuous growth in memory usage, whereas Gmapping shows an increased usage during the first 100 s followed by almost static utilization. Hector SLAM shows a completely static memory consumption.

Table 4.2 includes mean, median and standard deviation for the CPU load of each algorithm while mapping.

**Table 4.2:** CPU loads measured during SLAM.

Algorithm	mean (CPU %)	median (CPU %)	SD (CPU %)
Cartographer	277	317	99
Gmapping	56	94	48
Hector SLAM	8	6	9
Cartographer(gyro)	272	303	91

## 4.2 Navigation

This section presents tests related to positioning and navigation. The tests are based on maps generated by previously presented SLAM setup and the results are compared to the demands of industrial use.

The tests are performed at a test location environment set up to model real use cases to the extent possible with limited time and resources. Turning the vehicle just before heading into a station is recognized to be one of the toughest tasks for navigation and this scenario will be used in several of our tests. The tests require a map compatible with the Kollmorgen NDC8 system which is what the map converter, explained in Section 3.3, is used for.

### 4.2.1 Repeatability in positioning estimation

By using mounted reflectors on the walls for navigation, the vehicle controller is able to get millimeter precision. A test was conducted by driving the test vehicle along a pre-defined path with a stop at the end running positioning estimation. Both NDC8 Reflector navigation and AMCL was run simultaneously. The Reflector navigation is used as a reference of performance and AMCL is run with a SLAM-generated map of the area. The purpose of this test is to look at the performance of AMCL in conjunction with the SLAM map. It's important to note that the reflector navigation is run at the vehicle controller while AMCL runs on a PC. Initially the AMCL package was cross-compiled and run on the vehicle controller, however the results were substandard due to the CPU load exceeding reasonable quantities. The results were gathered using a script that, on command, reads the latest AMCL ROS message and matches it with a reflector position estimate with a corresponding time stamp from the system logs.



**Figure 4.6:** Path driven during repeatability tests. This pattern is recognized to be a tough, yet common scenario in warehouse AGV localization.

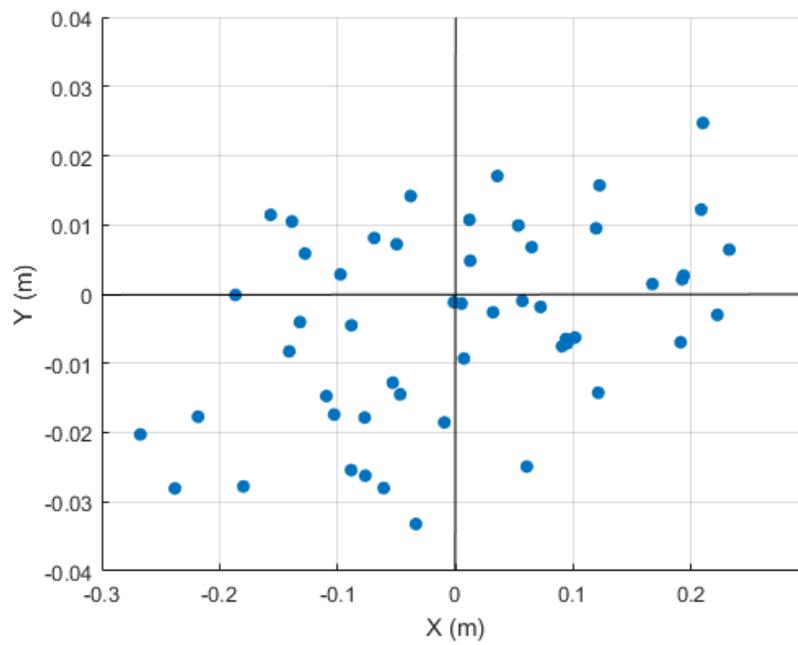
The path on which the test vehicle drove during this test is illustrated in Figure 4.6. This path is inspired by what is recognized to be one of the toughest situations in AGV navigation; the path is designed as a straight line, followed by a sharp turn and finally a shorter straight path towards the stopping point. A common situation for docking or going straight through corridors and taking sharp turns to pick up a load.

Figure 4.7 shows the measured positions at the stop point of AMCL and Figure 4.8 shows the position estimates from the reflector navigation. The tests were conducted with different coordinate systems but the mean value has been moved to (0,0) for increased readability. Table 4.3 shows the resulting standard deviations and root square mean distances from the test. More comprehensive test data can be found in Appendix A.

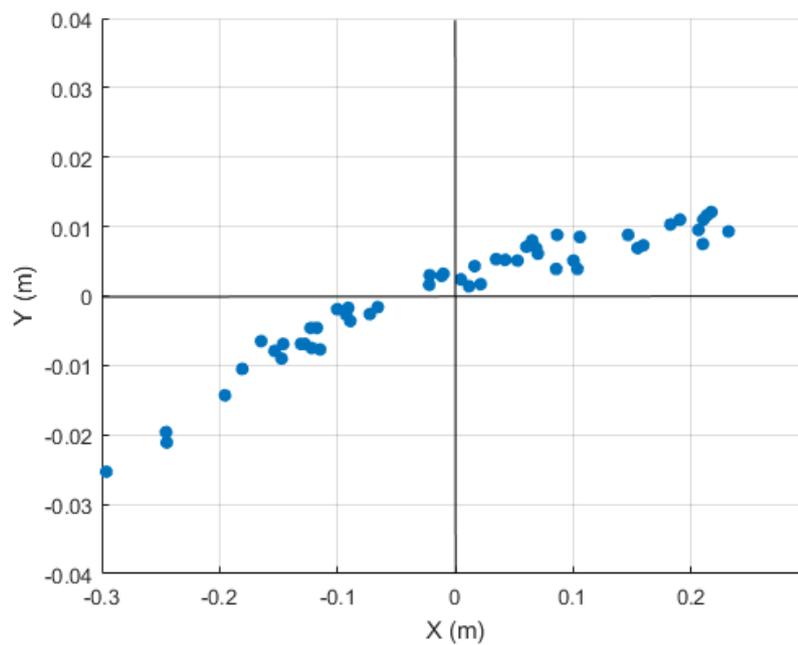
The figures shows a greater dispersion of the points using AMCL compared to using the reflector navigation. This results is further supported by the higher value for standard deviation for the y-axis for AMCL.

**Table 4.3:** Standard Deviations and Root Mean Square values for the position estimates from the repeatability test.

Technique	SD-y (m)	SD-x (m)	RMS (m)
Reflector	0.00872	0.13981	0.13873
AMCL	0.01375	0.12911	0.12859



**Figure 4.7:** Position estimates at each step during repeatability test by AMCL using a SLAM-generated map.

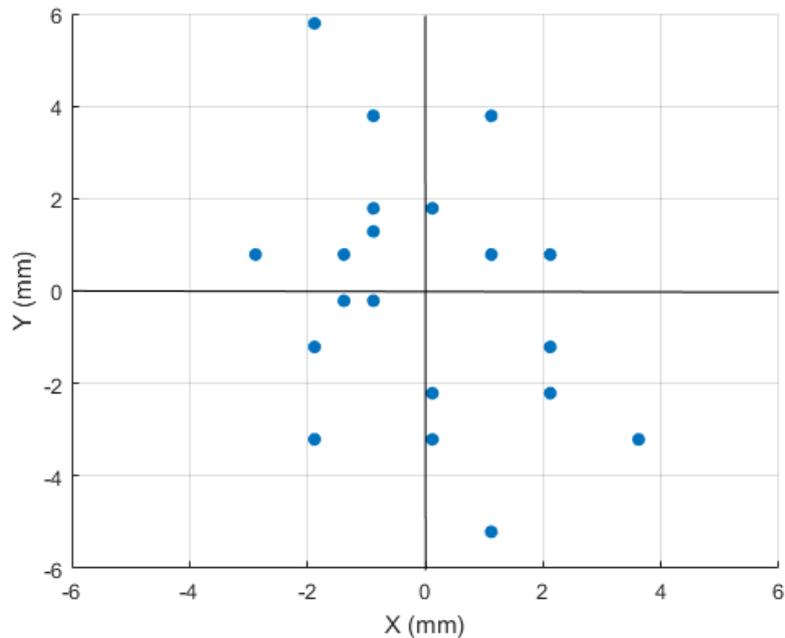


**Figure 4.8:** Estimated reflector navigation positions at each step during repeatability test.

### 4.2.2 Repeatability in automatic driving

In order to evaluate the quality that can be achieved of automatic driving with a SLAM-generated map, another test was conducted. This test was performed using Kollmorgens Natural navigation together with a SLAM generated map. The test was performed by drawing driving segments for the vehicle, using Kollmorgen software tools, and then letting the vehicle navigate according to these with a SLAM-generated map. The vehicle was ordered to repeatedly drive according to the drawn path and stop at a station. At each stop at the station, the position of the vehicle was marked. By measuring the marked real world position at each stop, the variance in real world positioning could be determined. This presents a metric for evaluation of repeatability in positioning.

Figure 4.9 shows the positions measured during testing, within a local coordinate system. The standard deviation is calculated to be 1.6949 mm in the X direction and 2.6859 mm in Y direction. Further, the largest distance between two points was measured to be 11.4 mm and the root square mean of distances 3.099 mm.

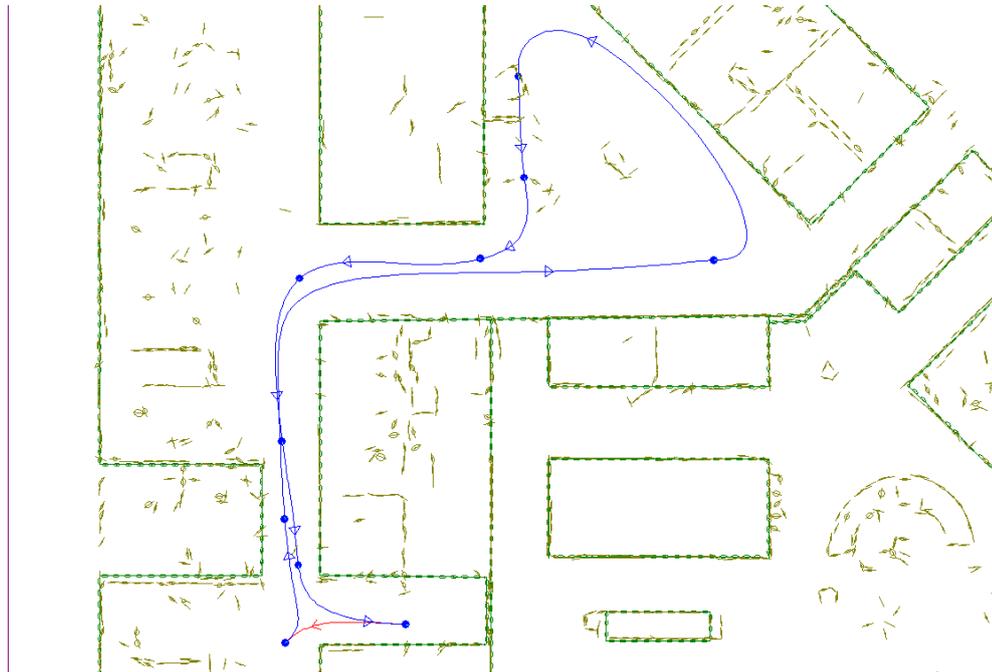


**Figure 4.9:** Visualization of real-world variance when testing repeatability in automated driving with SLAM-generated map.

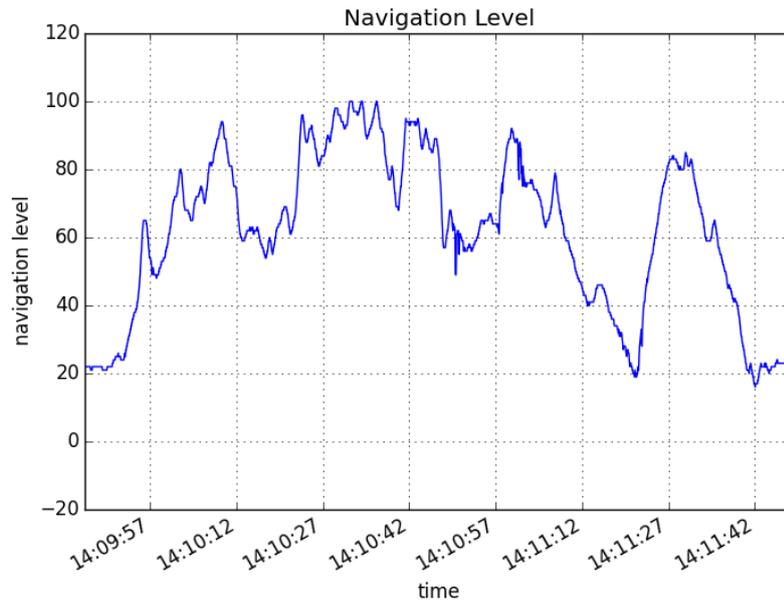
### 4.2.3 Navigation level

NDC8 includes tools for reading the navigation level of the vehicle in real time. The navigation level is a metric used to represent the certainty of the vehicles navigation ability where 0% means that the vehicle is lost.

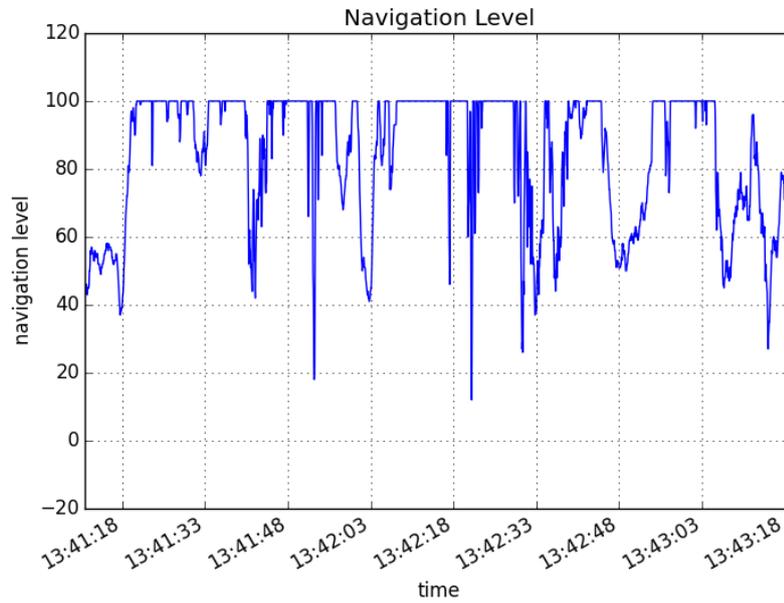
In order to further investigate the quality of the SLAM-generated maps, the test vehicle was driven, using Natural navigation, in a track which is showed in Figure 4.10. The same path was used with both a SLAM generated map and a manually created one. The navigation level during these trips were logged and is displayed in Figure 4.11 and Figure 4.12. The corresponding plots of the match ratio can be found in Figures A.5 and A.4.



**Figure 4.10:** NDT grid used to log the navigation level. The path driven by the robot is marked by solid lines with arrows.



**Figure 4.11:** The navigation level during test run with SLAM-generated map.



**Figure 4.12:** The navigation level during test run with manually drawn map.

The navigation level is constructed by a combination of three other values which are match ratio, correction and reliability. The match ratio is simply a percentage value for how well the scans match the NDT cells with the current position estimate. The corresponding plots of the match ratio can be found in Figures A.5 and Figure A.4. The reliability metric is used as a way to check that scans are

matched in all directions. Figures A.7 and A.6 displays the Reliability level during the test. Reliability is calculated by accumulating the covariances of all matched cells and calculating the ratio between the values for the different directions. As an example, this means that solely matching with one straight wall will lead to a low reliability value.

The third and last part of the navigation level metric is the accumulated correction. The key reason of having this metric is to find out when the vehicle has lost itself by matching on a erroneous location. In practice, jumps in position estimates are shown in the accumulated correction metric, which are a sign of uncertainty in positioning. These three metrics are all percentage values which are multiplied to get the navigation level.

The results for the navigation level test show that the manually created map has a higher average of certainty as well as a greater amount of accumulated time with values at 100% than the SLAM-generated one. The match ratio as well as accumulated correction have high values and are therefore almost negligible and it is mostly the reliability that is represented in the navigation level plot. The vehicle was able to navigate through the track as both plots are above 0% at all times.



# 5

## Discussion

This chapter presents discussions regarding elements of this thesis worthy of note. The implications of the results are examined and points of improvements are pointed out. Discussion around problems that occurred during the project and material for future work is included as well.

### 5.1 Software implementation

The following section includes discussions regarding design choices and technical limitations of the presented prototype system developed in this thesis work. Potential drawbacks are highlighted and a discussion of their impact is included. Some suggestions for improvements are proposed as well.

#### 5.1.1 Sensor publisher node

As previously mentioned the software, of which the ROS publishers were built upon, limits the publishing frequency to 16 Hz. This is considered to be enough, based on the moving speed of the used AGVs. The Gyroscope produces measurements with a higher rate than this and the average of these values are used when publishing the IMU message. Furthermore, the LS2000 provides a resolution of 6300 values per scan which is higher than what is used in most applications. Different ratios of angular resolution were tested, where high numbers resulted in duplicated walls in SLAM and also caused Wi-Fi problems due to the increased amount of transmitted data. Also, the developers of GMapping has limited the input size to 2048 values per measurement. The published scan messages was therefore scaled to 630 values per scan in order to minimize the stress on the system while still providing good results. It should however be noted that the main benefit from having higher angular resolution is that it is possible to better estimate objects far away, which is useful in large spaces.

The laser scanner limits the range of measurements to about 25 m. If it would be possible to increase the angular resolution, then it would also be interesting to be able to utilize a higher range for measurements. Supposedly, the probability of loop closures failing in SLAM algorithms can be reduced by having a longer range for scan measurements [2]. Furthermore, the LS2000 also provides intensity measurements which is based on the return strength of a laser beam. Since most applications today have no use for these they are not published in the implementation. There does however exist interesting research, such as by R. A. Hewitt

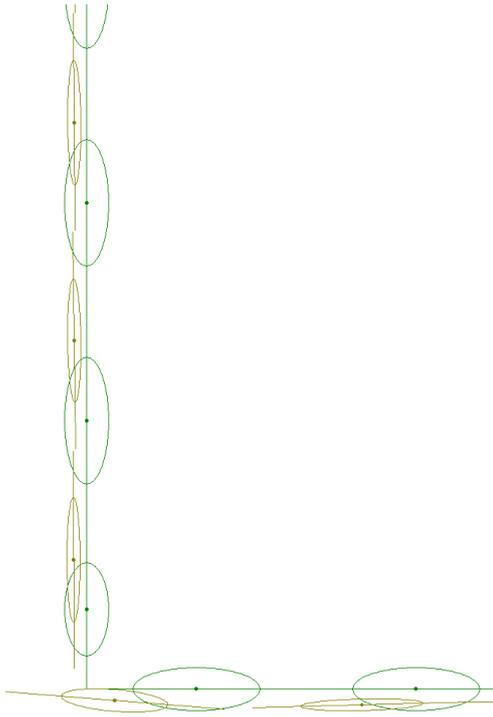
and J. A. Marshall [14], where the intensity of scans is used to improve the SLAM performance. The intensity could be used as an additional constraint to the robot pose and landmark estimates. It is also interesting to consider using intensity to include reflectivity and surface normals to the map.

### 5.1.2 Map converter

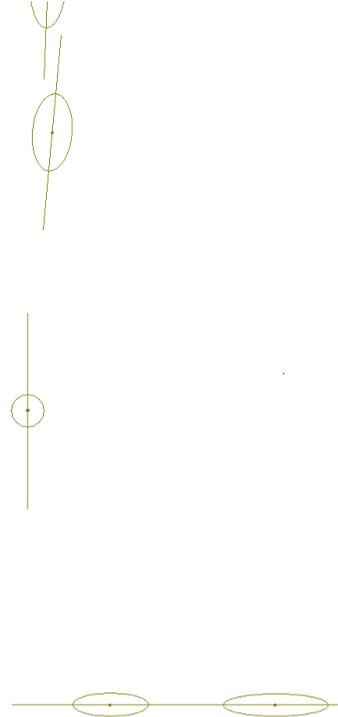
The map converter developed in this project has some obvious flaws or shortcomings. The foundation of which to build a NDT grid [3] is the scan points measured within each cell. As mentioned in Section 3.3, the occupancy grid does not have explicit information regarding the measured scan points causing the need for an alternative approach. These problems are known, however it is still important to point out the shortcomings of the algorithm as this limits the results.

The map converter uses several sampling points from one occupancy grid cell in order to solve the problem of thin walls, as discussed in Section 3.3. However, it is possible that the result would be better using something like a quincunx sample scheme that gives higher weight to the middle coordinate of the occupancy grid cell. Since the relation between occupancy grid cells and NDT cells is non-trivial in its nature, finding an optimal sampling scheme would require extensive testing and we settled for a simple solution which still provides sufficient results.

The difference in navigation level between the SLAM-generated and the manually created map during the test presented in Section 4.2.3 is mostly explained by the difference in reliability. As explained, this metric is considering the direction of matching NDT cells. The map converter used is far from perfect, and this could very well be the reason for the lower reliability level. A drop in reliability could be the result of a low amount of matching scan points. However, when the match ratio is considered separately the results show no distinct difference between the manually created map and the SLAM-generated one. Figure 5.1 and Figure 5.2 show a zoomed in version of the maps used when logging the navigation levels seen in Section 4.2.3. It is clear that the SLAM generated map is more sparse.

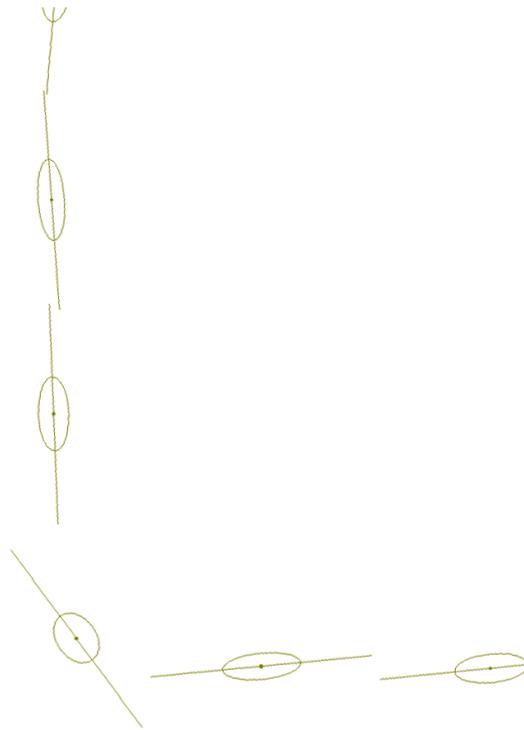


**Figure 5.1:** Zoomed in manually created NDT map used in navigation level test. The ellipses represents the covariance around the mean.



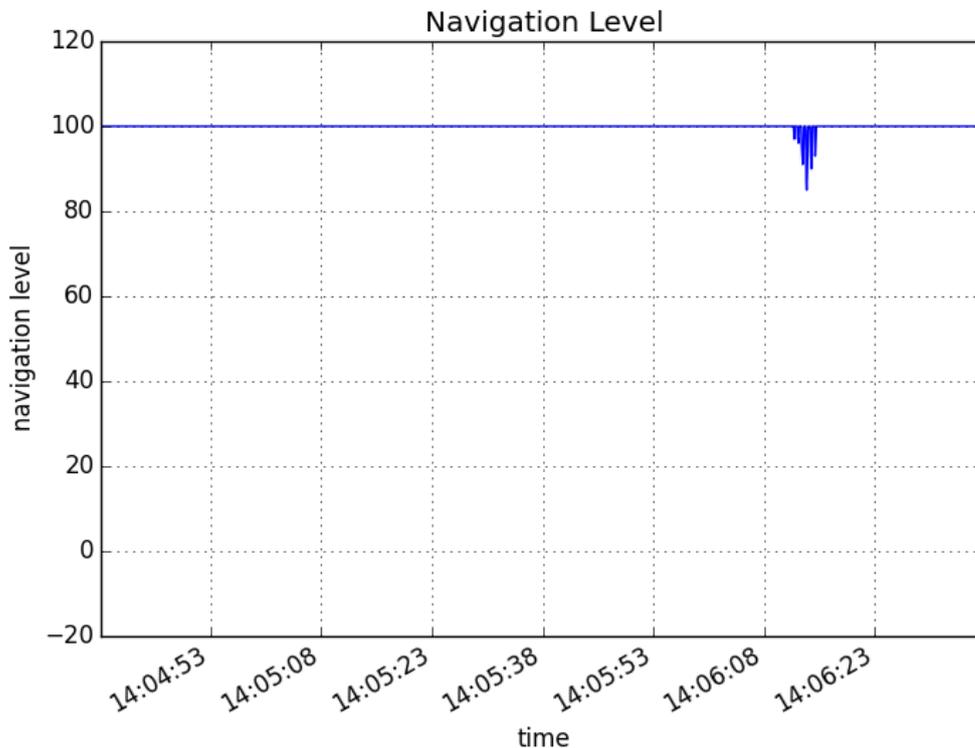
**Figure 5.2:** Zoomed in picture of SLAM-generated map converted to NDT cells. The ellipses represents the covariance around the mean.

In the version of the map converter used to generate Figure 5.2, only occupancy grid cells with a probability level over 65% was registered. This threshold was chosen as it is the default value of the ROS module which was used to save maps. The map conversion algorithm was further modified by lowering this threshold to 55%, in order to experiment with a more dense NDT map. In addition, weights were added for the cells in order to transfer the certainty level from the occupancy grid cell to the NDT cells in a reasonable way. A occupancy grid cell with a high probability of being occupied gets a high weight. The weight of a occupancy grid cell determines its contribution to the mean and covariance for the resulting NDT cell. Figure 5.3 displays the NDT cells of the same corner area as seen in Figure 5.2, but these have been created with the modified map converter. As intended, the NDT map is now more dense.



**Figure 5.3:** Zoomed in picture of SLAM-generated map converted to NDT cells with the modified map converter. This version includes occupancy grid cells with lower probability of being occupied than the one used for tests in Chapter 4. This results in a more dense NDT map compared to Figure 5.2.

The navigation level test was re-run with the more dense SLAM-generated map shown in Figure 5.3. The resulting navigation level is shown in Figure 5.4. It should be noted that the three metrics which are combined to create the navigation level each have their own maximum threshold. This means, for example, that when the Reliability is higher than 30% it is considered to be 100% when combining with the other metrics to create the navigation level. Plots of the individual values for the Reliability and Match ratio from the test with the updated map converter can be found in Figures A.9 and A.10 and they show higher values than those from the manually created map. It should also be noted that the navigation level and its different sub components have limited usefulness when making comparisons since they were created for debugging-specific purposes. It is however possible to conclude that the navigation level does not have to be worse with a SLAM-generated map than with a manually created map. The test also show one of the strengths of using a SLAM solution, namely the ability to keep a map updated with the current environment which is believed to be the reason for these exceptional results.



**Figure 5.4:** Navigation level when running the same track as presented in Section 4.2.3 but with an updated version of the map converter. This version includes occupancy grid cells with lower probability of being occupied than the one used for tests in Chapter 4.

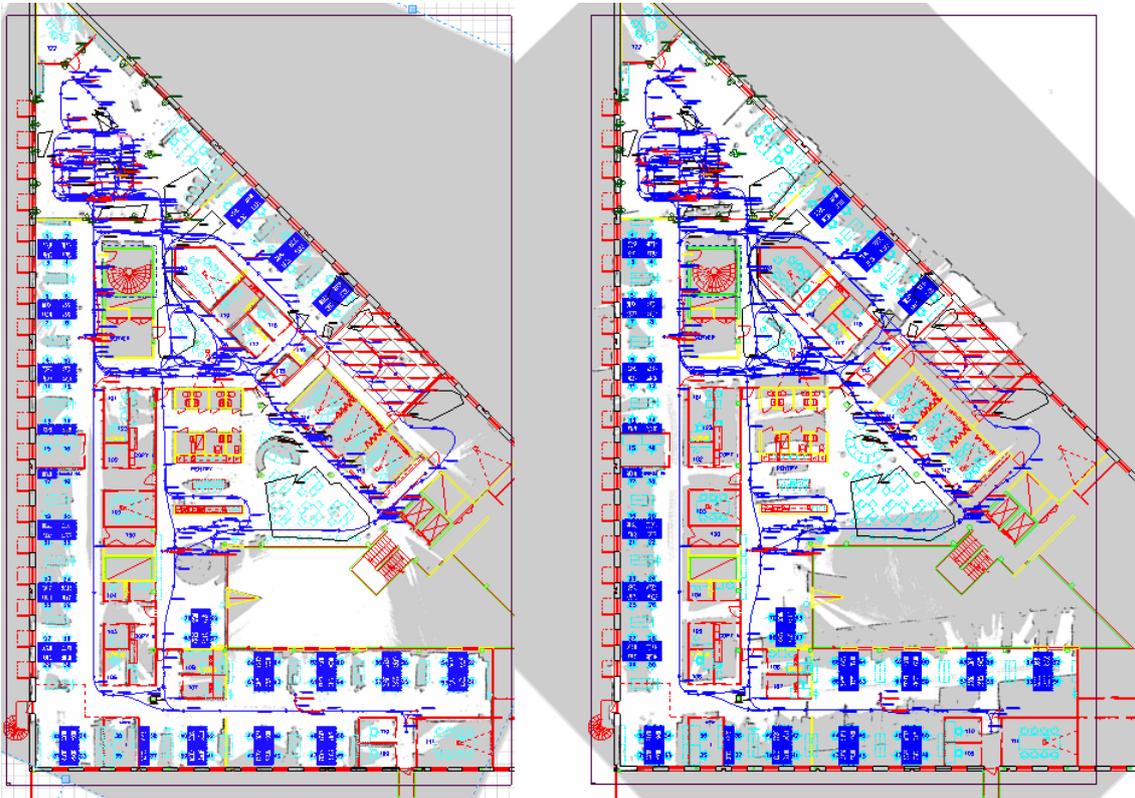
## 5.2 Test results

This section presents our thoughts around the implications of the results presented in Chapter 4 as well as discussions regarding the methods used. Furthermore some related work is presented with the purpose of putting the results into a bigger context and comparing approaches.

### 5.2.1 Mapping

The alignment of SLAM-generated maps to a CAD floor plan showed interesting results. Based on visual comparison it is determined that the cartographer algorithm produced a pretty much flawless result whereas Hector SLAM and Gmapping showed errors in the output. It is important to note that since the alignment is made manually, there is room for error causing misleading results. The skew that can be seen in the resulting maps of Gmapping and Hector SLAM is a typical error for SLAM algorithms. Figure 5.5 showcases this problem, where the map to the left is constructed using LIDAR and encoder measurements while the map to the right was created with LIDAR and gyroscope measurements. This mapping (Figure 5.5)

with gyroscope was made using a much higher vehicle speed, compared to the SLAM map built with gyroscope in the mapping test, in order to highlight the problems that occur when the algorithms "fails". As can be seen, the right-most picture shows distinct errors in the lower part of the map. One should also be aware of the fact that the CAD drawing might not represent reality perfectly.



**Figure 5.5:** SLAM generated maps with floor plan overlay. The picture demonstrates the phenomenon where some parts of a map gets skewed. The white parts are from the occupancy grid produced by SLAM.

Manual measurements were made to further emphasize the results. This test also showed that the Cartographer produced the best results among the tested algorithms. The measurements in the SLAM-generated maps were made manually with the RViz visualization tool. The occupancy grids have a resolution of 5cm, which means that each cell represents an area of  $5cm^2$ . The resolution implies that errors, in the tests, in the range of 5-10 cm is reasonable. The average error for all algorithms are satisfying when the measurement error is taken into account. The measured maximum error shows potential problems in the maps, however not very big ones for any of the algorithms.

Hector SLAM was run with rotational velocity from a gyroscope as input instead of odometry. This decision was made since Hector SLAM is developed with this purpose in mind, allowing for better results when odometry is unreliable for example due to non-planar environments [15]. While one can assume that AGVs in warehouses have good conditions for odometry, there are other reasons for excluding odometry. Examples of such reasons would be that sensors such as wheel encoders

are expensive to mount and that with the use of only IMU measurements one would not require an actual vehicle for mapping. This makes it interesting that Hector SLAM produces as good results as Gmapping in the tested environment. Furthermore, results also show that Cartographer basically provides the same result with a gyroscope as with encoders. Odometry would however most likely show better results than using the gyroscope if higher moving speed would have been used since it provides a better measurement of translation.

Evaluation of SLAM performance is most often found in papers presenting new algorithms in order to prove validity. Sometimes the authors are satisfied with using simple visual inspection while some papers [13] [27] include evaluations based on some metric similar to the one presented in "On measuring the accuracy of slam algorithms" [16], which measures the errors between the calculated trajectory and a ground truth. That is, how well the estimated poses match reality. This kind of evaluation requires access to more information, such as ground truth trajectory, than the ones used in this thesis. The errors in trajectory estimation that can be found using this type of test are also found in the resulting map and therefore indirectly represented in the results presented in this report. In this thesis, simply testing the quality of the resulting map and its performance used in navigation was determined to be sufficient.

The paper "An evaluation of 2D SLAM techniques available in Robot Operating System" [25] compares the most popular SLAM algorithms available in ROS. As the Cartographer had not been developed at that time, it is missing from that evaluation. The authors proposes an interesting comparison method for maps. The concept of the test is to align the obtained map narrowly with a ground truth map and calculate the error of each occupied cell. The error for each occupied cell in the ground truth can be calculated as follows: a k-Nearest Neighbour (KNN) [1] search finds the k nearest neighbours. As we are only interested in the distance of the closest occupied cell we would let K equal to one. The array of distances provided by the KNN search for each occupied cell in the ground truth map should then be summarized and divided by the length of the array. Summation of the errors of each cell and division by number of cells gives a total error and standardized metric of the obtained map is retrieved.

A test, similar to the one presented by Santos, Portugal and Rocha [25], was planned for this thesis and the ground truth based on CAD drawings of the mapped environment was retrieved and converted to an Occupancy Grid. Furthermore, the ground truth was then supposed to be aligned with the slam generated maps. However, the tested underlying alignment techniques (openCV and Matlab image processing toolbox) which are based on matching pair of points to find a fit, were unable to align the maps correctly. The visual inspection used in this thesis was successfully able to differentiate the algorithms and the error based alignment test was therefore abolished.

When examining the CPU load and memory usage during mapping, some interesting results were found. First off, it is clear from Figure 4.4 that GMapping is restricted to a single core while Cartographer is written for multi-threading. A distinct difference in CPU load between Hector SLAM and the other algorithms can be observed as well, proving the intention of Hector SLAM, namely to run on

low power and low cost processors [15]. The most interesting finding is that the Cartographer does not "finish" but instead keeps optimizing the map even after all measurements are received. This is seen as the CPU load does not drop and the memory usage keeps increasing after the 900 seconds of sensor measurements that was recorded.

As previously stated, loop closure is one of the most important components of SLAM. One failed loop closure will most likely cause the map to diverge from reality [2]. GMapping is often referred to as state of the art for particle filter-based SLAM. The algorithm is supposed to provide good results with small loops however struggle with larger areas which is probably what is expressed in our tests.

Graph-based SLAM algorithms has the advantage of having a better basis for dealing with loop closures explicitly. The Cartographer, which is graph-based, is implemented using submaps and global optimization. Scans are inserted into the submaps at estimated positions and when a submap is finished, it is further considered for loop closure with all other submaps and scans [13]. The Cartographer has previously shown promising results and these are validated in our tests showing convincing results compared to other state of the art algorithms. A problem of the graph-based SLAM algorithms is that memory usage grows with the amount of scans while GMapping has almost a fix sized map [2], which is also seen in our tests.

Hector SLAM is designed to be a lightweight fast SLAM algorithm and runs without explicit techniques for loop closure [15]. The tests presented in this thesis shows that the algorithm does just that. The produced map shows good quality even though some errors can be found. CPU and memory usage of Hector SLAM is significantly lower than the other tested algorithms. These result contradict those found in "An evaluation of 2D SLAM techniques available in Robot Operating System" [25] where GMapping and Hector SLAM expresses the same CPU load. The algorithm settings used in this paper is supposed to be the default configuration, just like in our case. The difference in the results could potentially be derived from the difference in scan update frequencies, where 10 Hz is used in that paper and 16 Hz is used in this thesis. Hector SLAM is presented to better take advantage of a higher update rate [15].

The idea of running a SLAM algorithm, directly on the vehicle controller, was experimented with during this project however this resulted in immediate software crashes, due to the restricted CPU capacity. The fact that most areas of use are related to small-scale embedded systems makes development of computationally cheap SLAM algorithms, like Hector SLAM, an important research topic. Attempts exist where the aim is to optimize SLAM together with the hardware to achieve this, such as presented in the paper "Design and evaluation of an embedded system based SLAM applications" [30]. The paper presents a solution to the SLAM problem where hardware architecture is designed to fit with a feature detector and SLAM algorithm. A related idea is presented in "Synchronized Visual-Inertial Sensor System with FPGA Pre-Processing for Accurate Real-Time SLAM" [21] where a inertial sensor unit is presented which is supposed to aid the deployment of a system with SLAM capabilities.

## 5.2.2 Navigation

The test presented in Section 4.2.1 showed that the reflector positioning performed better than the AMCL SLAM map positioning. The fact that there is a "line" of estimates(dots) in the results can be explained by two different theories. The results may have been gathered in the same line which is seen in the plot, due to some delay in communication and logging. The other explanation would be that the direction of which results of both techniques are spread wide is particularly hard to estimate. The test was however conducted a second time on a second location, in order to rule out any obvious flaw with the environment, with the same result. The main purpose of the test was to verify the precision of the SLAM-generated map as well as the AMCL ROS package, where the reflector navigation acts as a guideline for the "optimal performance". The dispersion seen in the results are at a very small scale with results only departing with a few millimeters on the y-axis. This indicates that positioning estimation with a SLAM-generated map has the potential of performing at the level required in industry settings.

Furthermore, it should be noted that the Reflector navigation is executed on the vehicle controller while AMCL had to be executed on an external PC due to the limited system resources. The AMCL ROS package is somewhat demanding, in terms of CPU load and memory. AMCL is an implementation of the adaptive Monte Carlo localization approach, which uses particle filters for position tracking [11].

In order to directly test a SLAM-generated map for industrial use cases the repeatability in automatic driving was tested. The most basic, and important, use case is the simple task of driving onto the position of a loading dock or vehicle station. These stations are often placed in a corridor where the vehicle has to follow the movement pattern shown in Figure 4.6. The test shows that with a SLAM-generated map, the vehicle is able to navigate to the same position over and over again with very small deviation. The vehicle could perhaps be ordered to pick up a pallet, and the resulting worst case position deviation of roughly 1 cm is then a satisfying result.

The navigation level test shows that navigation with a SLAM-generated map can achieve the same performance as with a manually created map. There are of course advantages to either solution. As an example, a SLAM algorithm will include all existing objects in the environment and there is no simple solution to manually exclude objects whereas it is problematic to take advantage of small objects with manually created maps.

The two navigation level curves follow each other and share similar drops and gains. This shows that the most critical problems encountered are not because of the map but instead are caused by a difficult area or vehicle movement. Furthermore an important result to note from this test is that the vehicle, with only a modest setup time, performed as well as what is achieved using a manually created map. The only setup work used was a quick run with a SLAM algorithm together with a simple and flawed map converter.

It should be noted that none of these tests look into any particularly hard cases. One such example could be when there are a lot of moving objects in a warehouse which occludes the sight. These kind of situations are not specific to the use of SLAM-generated maps and the solution is often some manual fix. The

relevance of these kind of tests was therefore determined to be small.

Finding new solutions for localization is always an interesting topic. With the increased availability of 3D sensors, one might want to consider using more environment data such as full 3D scans for AGV localization. The 2D localization solutions might then not be extendable into 3D and other techniques might be required, which is one point where NDT might prove more useful [23]. While the most common localization technique used for AGVs is range scanners, researching other new indoor localization techniques is interesting to reduce cost or increase precision. The paper "A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned" [17] includes an extensive look at various localization technologies. One of the most promising techniques would be UWB radio [5] which is based on a time-of-arrival method for measuring the time of flight for UWB pulses. The results of this paper shows that the best UWB solution proved an average localization error of 0.72 m while the best wifi-based solution had a resulting accuracy of 1.6 m.

Many attempts to recreate a GPS-like solution for indoor environment exists, however the precision of such solutions is not enough for problems presented to AGV systems. High precision is required in these systems to avoid collision with other vehicles, and to get through narrow spaces for example. To further put this in perspective, impressive indoor localization accuracy, using UWB, can be found at around 0.5 m [6]. The LS200 laser scanner used in this thesis has a accuracy of 25 mm <sup>1</sup> and the results presented in this report shows vehicle position accuracy somewhere around 0.01 m.

---

<sup>1</sup>[https://files.pepperl-fuchs.com/webcat/navi/productInfo/doct/tdoct3124d\\_eng.pdf?v=20160204155755](https://files.pepperl-fuchs.com/webcat/navi/productInfo/doct/tdoct3124d_eng.pdf?v=20160204155755)

# 6

## Conclusion

The aim of this thesis was to investigate automated mapping solutions for AGV systems. This was evaluated by developing a prototype system, based on the Robot Operating System and conducting tests on the performance.

This report presents details for a ROS implementation compliant to the navigation stack. It is no doubt that ROS is powerful when developing applications or prototypes for automated vehicles. The framework includes state of the art navigation packages and enables testing in a simple way. The potential weaknesses are related to real-time constraints and message transmission time guarantees [12], however this has not shown to be a problem during this thesis project.

The evaluation of existing SLAM algorithms shows that there exist algorithms able to produce high quality environment maps without excessive resource use. The Hector SLAM algorithm shows a lot of promise in smaller areas where it requires minimalistic amounts of resources. The environment used in testing for this thesis is of moderate size, and even though the Cartographer showed exceptional performance, the algorithm might require further testing in larger areas with larger loop closures.

A map converter was created, creating a bridge between the NDT and occupancy grid map representations. The converter successfully enabled further application-specific tests. With focus on repeatability, additional tests were conducted to evaluate the navigational performance when using static SLAM generated maps. By comparison with existing techniques, which have proven solid performance, the presented tests have been put in context. The results from the tests presented in this thesis shows that the overall performance achieved with SLAM-generated maps is satisfying. In addition to providing accurate maps, the technique reduces the setup time and the amount of manual work required for map creation significantly. In the algorithms current condition, SLAM might require putting a more powerful computer on the robot for increased computational power.

Having a decentralized version of SLAM, where an environment map can be continuously updated and shared between several vehicles is an interesting topic for future work. Assuming offline SLAM algorithms provide acceptable performance, the next natural step is to run it online, in a decentralized way, at the same time as the AGVs carries out orders in a warehouse. The decentralization introduces many new problems, such as when and how to share or fuse data between vehicles [4], but this could potentially further reduce the maintenance work required in industry use.



# Bibliography

- [1] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
- [2] P. Beinschob and C. Reinke. Graph slam based mapping for agv localization in large-scale warehouses. In *2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 245–248, Sept 2015.
- [3] P. Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 3, pages 2743–2748 vol.3, Oct 2003.
- [4] G. Bresson, R. Aufrère, and R. Chapuis. Real-time decentralized monocular slam. In *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*, pages 1018–1023, Dec 2012.
- [5] K. C. Cheok, M. Radovnikovich, P. Vempaty, G. R. Hudas, J. L. Overholt, and P. Fleck. Uwb tracking of mobile robots. In *21st Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 2615–2620, Sept 2010.
- [6] B. Denis, L. Ouvry, B. Uguen, and F. Tchoffo-Talom. Advanced bayesian filtering techniques for uwb tracking systems in indoor environments. In *2005 IEEE International Conference on Ultra-Wideband*, pages 6 pp.–, Sept 2005.
- [7] Arnaud Doucet, Nando de Freitas, Kevin P. Murphy, and Stuart J. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, UAI '00*, pages 176–183, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
- [8] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.
- [9] E. Einhorn and H. M. Gross. Generic 2d/3d slam with ndt maps for lifelong application. In *2013 European Conference on Mobile Robots*, pages 240–247, Sept 2013.
- [10] T. Foote. tf: The transform library. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6, April 2013.
- [11] Dieter Fox. Kld-sampling: Adaptive particle filters. In *NIPS*, volume 14, pages 713–720, 2001.
- [12] A. M. Hellmund, S. Wirges, Ö. Ş. Taş, C. Bandera, and N. O. Salscheider. Robot operating system: A modular software framework for automated driving. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1564–1570, Nov 2016.

- [13] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, May 2016.
- [14] R. A. Hewitt and J. A. Marshall. Towards intensity-augmented slam with lidar and tof sensors. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1956–1961, Sept 2015.
- [15] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160, Nov 2011.
- [16] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss, and Alexander Kleiner. On measuring the accuracy of slam algorithms. *Autonomous Robots*, 27(4):387, 2009.
- [17] Filip Lemic, Jasper Buesch, Zhiping Jiang, Han Zou, Hao Jiang, Chi Zhang, Ashwin Ashok, Chenren Xu, Patrick Lazik, Niranjini Rajagopal, Anthony Rowe, Avik Ghose, Nasim Ahmed, Zhuoling Xiao, Hongkai Wen, Traian E. Abrudan, Andrew Markham, Thomas Schmid, Daniel Lee, Martin Klepal, Christian Beder, Maciej Nikodem, Szymon Szymczak, Pawel Hoffmann, Leo Selavo, Domenico Giustiniano, Vincent Lenders, Maurizio Rea, Andreas Maccalenti, Christos Laoudias, Demetrios Zeinalipour-Yazti, Yu-Kuen Tsai, Arne Bestmann, Ronne Reimann, Liqun Li, Chunshui Zhao, Stephan Adler, Simon Schmitt, Vincenzo Dentamaro, Domenico Colucci, Pasquale Ambrosini, Andre Ferraz, Lucas Martins, Pedro Bello, Alan Alvino, Vladica Sark, Gerald Pirkel, Peter Hevesi, Xue Yang, Romit Roy Choudhury, Vlado Handziski, Souvik Sen, Dimitrios Lymberopoulos, and Jie Liu. A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned. In *The 14th ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN '15)*. ACM – Association for Computing Machinery, April 2015.
- [18] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Auton. Robots*, 4(4):333–349, October 1997.
- [19] Nicholas Metropolis and Stanislaw M. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247):335–341, September 1949.
- [20] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Eighteenth National Conference on Artificial Intelligence*, pages 593–598, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
- [21] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 431–437, May 2014.
- [22] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [23] J. Saarinen, H. Andreasson, T. Stoyanov, J. Ala-Luhtala, and A. J. Lilienthal. Normal distributions transform occupancy maps: Application to large-scale

- online 3d mapping. In *2013 IEEE International Conference on Robotics and Automation*, pages 2233–2238, May 2013.
- [24] Otavio Salvador and Daiane Angolini. *Embedded Linux Development with Yocto Project*. Packt Publishing Ltd, 2014.
- [25] J. M. Santos, D. Portugal, and R. P. Rocha. An evaluation of 2d slam techniques available in robot operating system. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–6, Oct 2013.
- [26] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *Int. J. Rob. Res.*, 5(4):56–68, December 1986.
- [27] T. Stoyanov, J. Saarinen, H. Andreasson, and A. J. Lilienthal. Normal distributions transform occupancy map fusion: Simultaneous mapping and tracking in large scale dynamic environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4702–4708, Nov 2013.
- [28] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- [29] Sebastian Thrun and John J. Leonard. *Simultaneous Localization and Mapping*, pages 871–889. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
- [30] B. Vincke, A. Elouardi, and A. Lambert. Design and evaluation of an embedded system based slam applications. In *2010 IEEE/SICE International Symposium on System Integration*, pages 224–229, Dec 2010.



# A

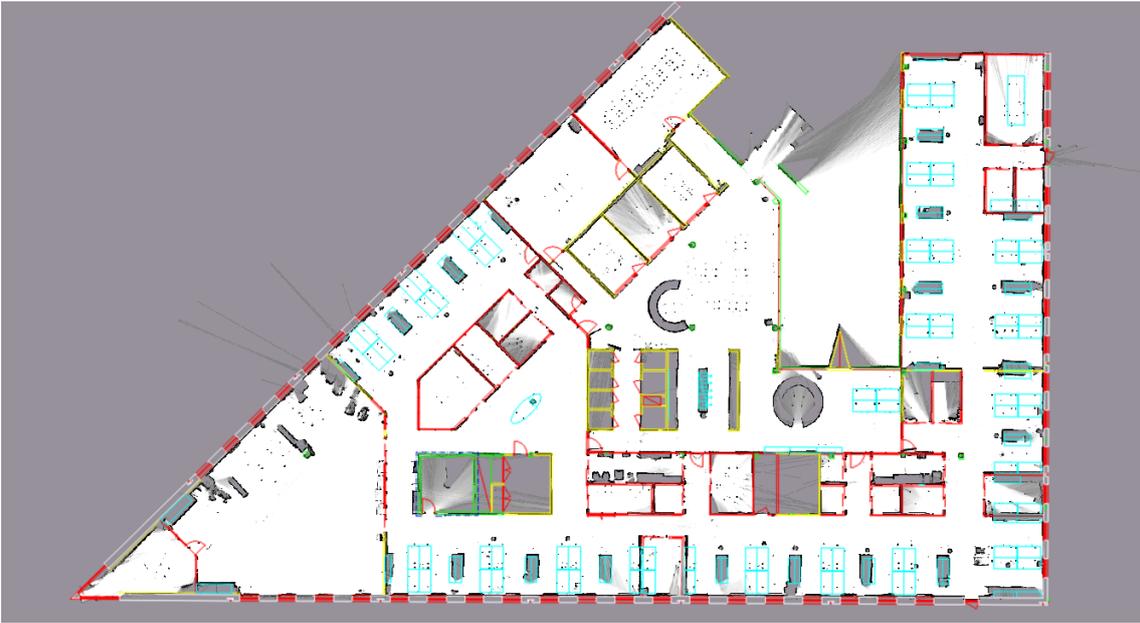
## Appendix 1

The appendix includes test results that were determined to be superfluous and therefore not included in the report.

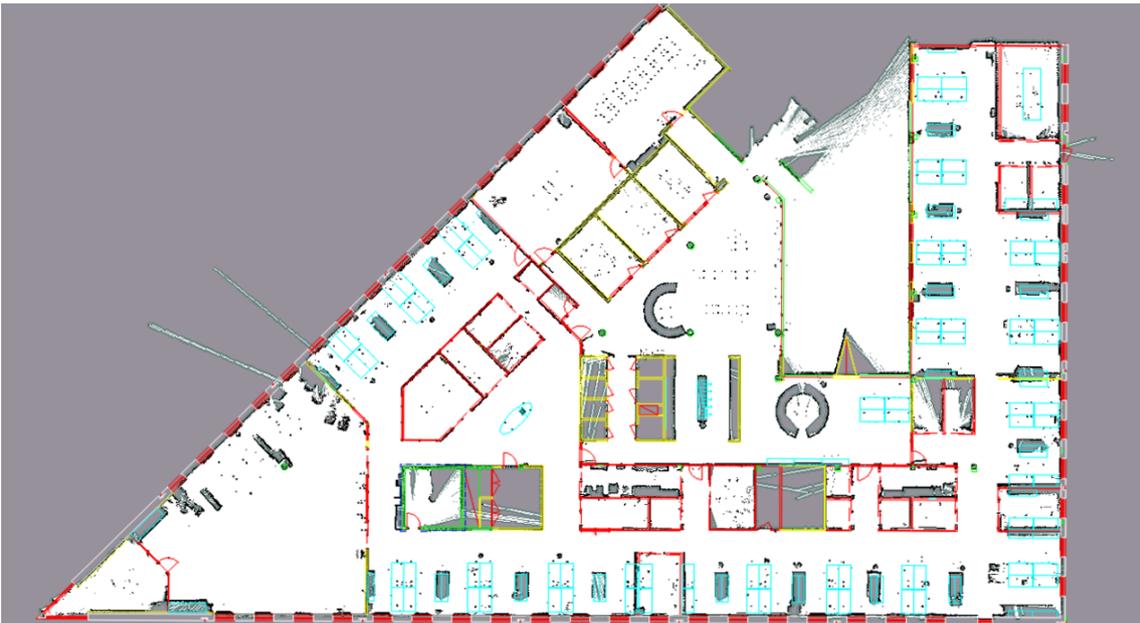
Figures A.1, A.2 and A.3 displays alignment of the CAD drawing on top of the maps generated by different SLAM algorithms.



**Figure A.1:** Map from Cartographer, using gyroscope, aligned with CAD drawing.



**Figure A.2:** Map from Cartographer, using encoders, aligned with CAD drawing



**Figure A.3:** Map from Hector SLAM aligned with CAD drawing.

Table A.1 includes the test data from the manual measurements test. The reference values measured by the handheld laser range finder are followed by the distance between the corresponding positions in the SLAM-generated maps.

**Table A.1:** Distance between reference points measured both in reality and in all four SLAM generated maps.

#Test	Reality	Cartographer	Hector	Gmapping	Cartographer(gyro)
1	13.043	13.032	12.957	13.061	13.091
2	9.6212	9.568	9.558	9.507	9.591
3	16.417	16.394	16.339	16.379	16.388
4	1.655	1.654	1.637	1.609	1.663
5	42.706	42.701	42.592	42.523	42.727
6	8.312	8.289	8.311	8.395	8.374
7	2.836	2.866	2.804	2.81	2.812
8	2.45	2.425	2.423	2.386	2.394
9	6.255	6.225	6.165	6.223	6.22
10	1.5	1.502	1.42	1.454	1.522
11	2.765	2.717	2.681	2.706	2.73
12	1.497	1.499	1.47	1.482	1.43
13	1.5	1.571	1.483	1.475	1.537
14	2.326	2.324	2.271	2.29	2.291
15	5.877	5.864	5.785	5.879	5.987
16	2.85	2.954	2.787	2.79	2.9
17	1.681	1.725	1.658	1.636	1.737
18	7.092	7.002	6.951	6.997	7.031
18	6.5	6.448	6.434	6.457	6.447
19	38.188	38.258	38.221	38.16	38.183
20	1.53	1.483	1.496	1.529	1.541
21	2.875	2.859	2.836	2.895	2.885
22	7.245	7.247	7.235	7.243	7.249
23	7.787	7.765	7.714	7.699	7.815
24	5.833	5.798	5.802	5.768	5.874
25	2.178	2.11	2.152	2.132	2.143
26	8.519	8.504	8.468	8.47	8.503
27	6.357	6.33	6.264	6.348	6.37
28	10.56	10.482	10.441	10.482	10.53

Table A.2 includes all measurements gathered when comparing position estimation with SLAM and AMCL to reflector positioning. Each row includes the unix time at which the measurements were logged as well as what position was estimated. Both positioning techniques were run simultaneously but two separate coordinate systems were used.

**Table A.2:** Position estimation - full data for the test presented in Section 4.2.1.

Test	Reflector			AMCL		
	Time	X	Y	Time	X	Y
1	1492758659.98	-0.1328	-0.0085	1492758659.90	31.7294	149.7073
2	1492758706.18	-0.4028	-0.0274	1492758706.10	31.4167	149.6921
3	1492758750.13	-0.2465	-0.0001	1492758750.14	31.6304	149.7071
4	1492758795.52	-0.3769	-0.0278	1492758795.50	31.4809	149.6964
5	1492758841.32	-0.4798	-0.0378	1492758841.34	31.4053	149.6935
6	1492758888.63	-0.3334	-0.0432	1492758888.62	31.6756	149.7085
7	1492758933.51	-0.1075	-0.0079	1492758933.50	31.7606	149.7110
8	1492758975.38	-0.3467	-0.0245	1492758975.33	31.6228	149.7051
9	1492759020.80	-0.3607	-0.0380	1492759020.74	31.4975	149.6974
10	1492759066.45	-0.2394	-0.0349	1492759066.40	31.6124	149.7052
11	1492759108.64	-0.4566	0.0014	1492759108.64	31.4240	149.6931
12	1492759163.86	-0.2926	-0.0193	1492759163.84	31.7166	149.7088
13	1492759208.98	-0.5671	-0.0303	1492759208.90	31.2740	149.6747
14	1492759257.43	-0.2643	0.0071	1492759257.44	31.6047	149.7053
15	1492759303.22	-0.2049	-0.0171	1492759303.22	31.6736	149.7039
16	1492759346.19	-0.4409	-0.0183	1492759346.12	31.4792	149.6983
17	1492759389.81	-0.3093	-0.0285	1492759389.81	31.5483	149.7030
18	1492759478.96	-0.1983	-0.0163	1492759478.90	31.6400	149.7061
19	1492759601.44	-0.3011	-0.0112	1492759601.42	31.5599	149.7032
20	1492759648.88	-0.0776	-0.0130	1492759648.82	31.8018	149.7093
21	1492759689.99	-0.3497	-0.0028	1492759689.92	31.4775	149.6973
22	1492759736.81	-0.1787	-0.0242	1492759736.85	31.7834	149.7116
23	1492759830.67	-0.2947	-0.0114	1492759830.62	31.5747	149.7024
24	1492759874.57	-0.1805	-0.0005	1492759874.54	31.6702	149.7051
25	1492759971.36	-0.4093	-0.0247	1492759971.33	31.4391	149.6931
26	1492760016.20	-0.2352	-0.0032	1492760016.26	31.6564	149.7088
27	1492760065.49	-0.1777	0.0057	1492760065.40	31.7247	149.7069
28	1492760110.55	-0.3974	-0.0072	1492760110.53	31.4525	149.6954
29	1492760155.54	-0.3379	0.0042	1492760155.52	31.5479	149.7016
30	1492760201.59	-0.5182	-0.0277	1492760201.54	31.3252	149.6789
31	1492760258.38	-0.4318	-0.0140	1492760258.30	31.3893	149.6895
32	1492760349.22	-0.2094	-0.0175	1492760349.20	31.6556	149.7039
33	1492760392.72	-0.4275	-0.0041	1492760392.70	31.4473	149.6954
34	1492760437.46	-0.2061	-0.0165	1492760437.40	31.6386	149.7069
35	1492760480.76	-0.4385	0.0005	1492760480.72	31.4226	149.6910
36	1492760524.91	-0.5379	-0.0381	1492760524.94	31.3246	149.6804

---

37	1492760570.90	-0.3530	-0.0228	1492760570.90	31.5043	149.6984
38	1492760617.40	-0.1062	-0.0073	1492760617.34	31.7526	149.7103
39	1492760659.59	-0.2431	-0.0110	1492760659.52	31.5915	149.7017
40	1492760705.69	-0.8316	-0.0701	1492760705.60	31.0062	149.6086
41	1492760752.78	-0.2879	0.0007	1492760752.70	31.5584	149.7029
42	1492760798.13	-0.0913	0.0022	1492760798.13	31.7763	149.7095
43	1492760840.23	-0.2276	-0.0119	1492760840.24	31.6352	149.7080
44	1492760884.27	-0.3884	-0.0354	1492760884.22	31.4482	149.6925
45	1492760926.42	-0.4865	-0.0101	1492760926.40	31.3746	149.6857
46	1492760970.66	-0.2680	-0.0126	1492760970.62	31.5864	149.7043
47	1492761016.66	-0.0899	0.0147	1492761016.64	31.7805	149.7110
48	1492761060.40	-1.1073	-0.2232	1492761060.45	30.7828	149.4924
49	1492761106.75	-0.3763	-0.0362	1492761106.70	31.4698	149.6981
50	1492761153.35	-0.0675	-0.0036	1492761153.33	31.7872	149.7121
51	1492761219.94	-0.1089	-0.0170	1492761219.98	31.7801	149.7075
52	1492761263.54	-0.2874	-0.0052	1492761263.54	31.5817	149.7014
53	1492761315.58	-0.3688	-0.0019	1492761315.50	31.4553	149.6923
54	1492761359.18	-0.3881	-0.0145	1492761359.12	31.4423	149.6931

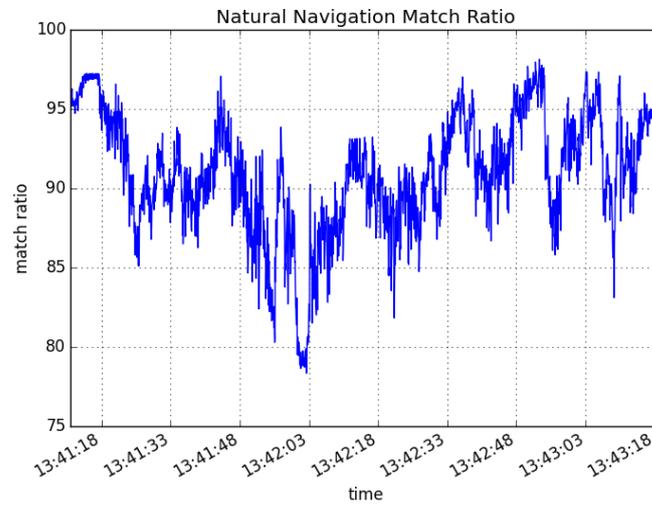
---

Table A.3 holds the stopping positions measured when ordering the vehicle to drive, with a SLAM-generated map, to the same reference position over and over again. A local coordinate system was used and the tests are not presented in the same order that they were conducted.

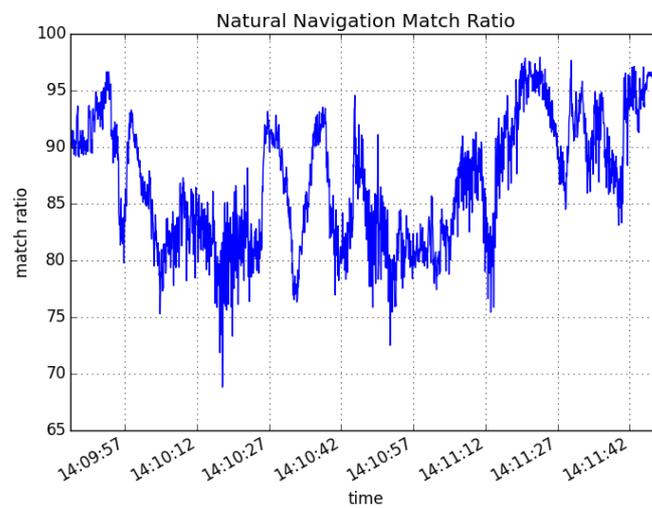
**Table A.3:** Navigation precision - measured stopping position at each test.

Test	x-pos	y-pos
1	0.70	1.80
2	0.80	1.40
3	1.00	1.60
4	0.90	1.40
5	0.80	1.35
6	0.70	1.10
7	1.00	1.30
8	1.10	1.30
9	1.10	1.00
10	0.80	1.60
11	1.00	1.30
12	1.10	1.10
13	0.90	1.00
14	1.25	0.90
15	1.00	0.70
16	0.90	0.90
17	0.70	0.90
18	0.80	1.20
19	0.75	1.20
20	0.75	1.30
21	0.60	1.30

Figure A.4 and Figure A.5 show the match ratio corresponding to the Navigation levels presented in Section 4.2.3.

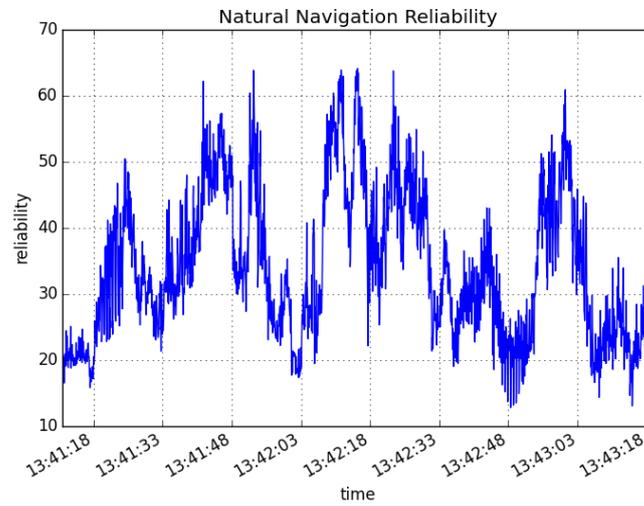


**Figure A.4:** Match ratio with handmade map used in navigation test.

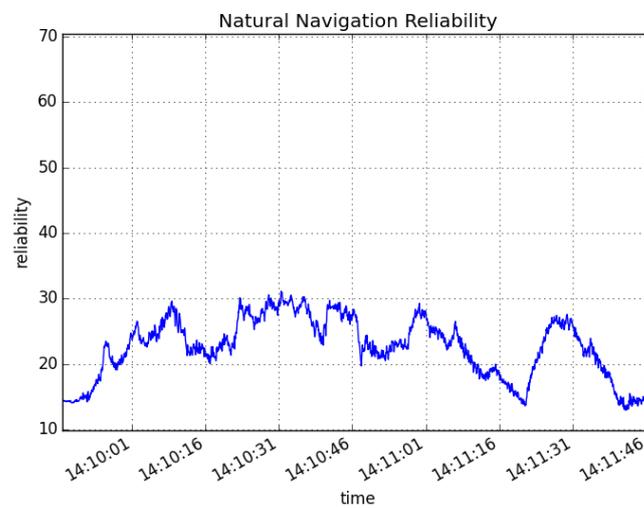


**Figure A.5:** Match ratio with SLAM-generated map used in navigation test.

Figure A.6 and Figure A.7 show the reliability corresponding to the Navigation levels presented in Section 4.2.3.

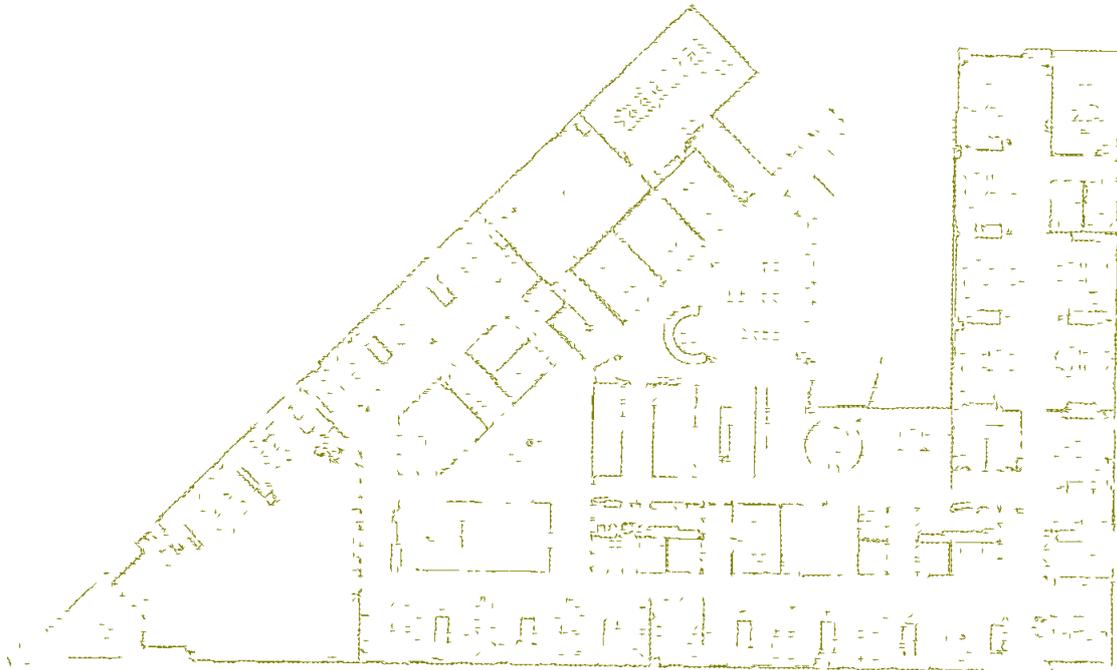


**Figure A.6:** Reliability with handmade map used in navigation test.



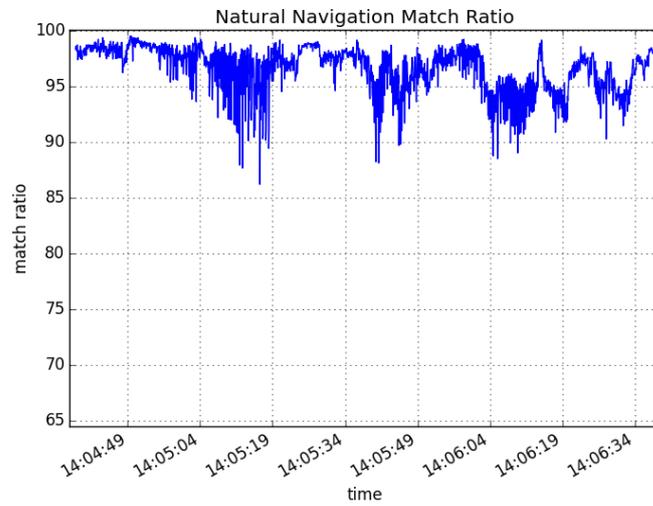
**Figure A.7:** Reliability with SLAM-generated map used in navigation test.

Figure A.8 shows the result of converting the cartographer map, of the full environment, to NDT cells.

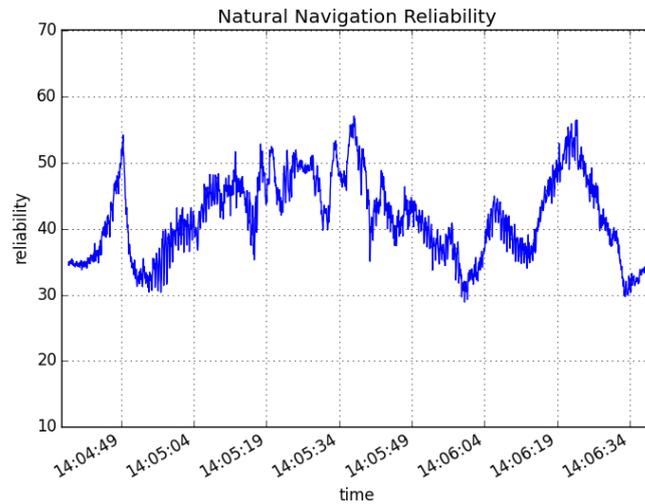


**Figure A.8:** Map, created by the Cartographer, converted to NDT cells.

Figures A.9 and A.10 display the result from the navigation level test when run with a modified version of the map converter. This version is explained in Section 5.1.2.



**Figure A.9:** Match ratio when running the same track as presented in 4.2.3 but with a updated version of the map converter. The features of this version is presented in Section 5.1.2.



**Figure A.10:** Reliability when running the same track as presented in 4.2.3 but with a updated version of the map converter. The features of this version is presented in Section 5.1.2.