CHALMERS
UNIVERSITY OF TECHNOLOGY

Automated guided vehicle navigation in unmapped semi-structured environments

Master's thesis in Systems, Control and Mechatronics

Sverre Bergdahl
Daniel Palmqvist

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

Master's thesis 2019

Automated guided vehicle navigation in unmapped semi-structured environments

SVERRE BERGDAHL
DANIEL PALMQVIST

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Automated guided vehicle navigation in unmapped semi-structured environments

SVERRE BERGDAHL
DANIEL PALMQVIST

Supervisor: Mikael Björn, Kollmorgen Automation AB
Examiner: Knut Åkesson, Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Automated guided vehicle navigation in unmapped semi-structured environments
SVERRE BERGDAHL
DANIEL PALMQVIST
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Since the invention of the automated guided vehicle (AGV), a portable robot employing different types of navigation, in 1953, the technology has seen big improvements. From using bumpers and emergency arrest handles as sensors to detect collisions to today's technology with a wider arrangement of available sensors. At Kollmorgen Automation AB, the AGVs use a LIDAR scanner to localize itself in an area by matching seen reflectors against known reflector positions, which is called *Reflector* localization. The AGV can also locate itself by matching seen objects to a map of known objects, which is called *Natural* localization. For some areas such as warehouses and pallet racks where the environment can change from day to day, *Natural* localization run into the problem that the map might be different than the area. A solution to this, used today, is putting up reflectors or using dead reckoning to traverse these areas. However, adding reflectors to an area demand an environment that allows for their installation and that they are in the view of the AGV at most times. Dead reckoning, however, work well for shorter distances but quickly accumulates error over distance travelled. An interesting solution to these problems is using Simultaneous Localization and Mapping (SLAM), a method used by a robot to create a map while simultaneously localizing itself in said map, to aid in dead reckoning in while traversing unmapped areas.
This master thesis project which was conducted in cooperation with Kollmorgen aims to evaluate the accuracy of single plane LIDAR SLAM generated trajectories in industrial environments. The SLAM algorithms evaluated were chosen to be GMapping and HectorSLAM. Tests were conducted in cooperation with Kollmorgen at their facilities using a setup that tried to replicate industrial environments. Test one and two had the AGV drive in a U-shape along a wall with the difference being that test two had boxes placed along the outside of the trajectory. Test three had the AGV drive in a line with boxes evenly placed along the trajectory to replicate pillars. The fourth test had the AGV drive in zig-zag between rows of boxes to replicate rows of pallet racks in a warehouse. In these tests it was found that for moderate speeds and in object rich environments SLAM generated trajectories proved to be both highly accurate and repeatable.

Keywords: AGV, Automated Guided Vehicles, Navigation, ROS, Robot Operating System, SLAM, Simultaneous Localization and Mapping, Dead-reckoning, Dead Reckoning.

Acknowledgements

This masters thesis has been carried out under the Department of Electrical Engineering at Chalmers University of Technology. We want to thank Kollmorgen Automation for the research topic, workplace and AGVs for testing. We would also like to extend our thanks to our co-workers at Kollmorgen Automation for their welcoming and helping attitude during our project, and especially thank our supervisor Mikael Björn for his knowledge and help during the project.

Sverre Bergdahl and Daniel Palmqvist, Gothenburg, February 2019

Contents

List of Figures

List of Tables

Glossary

ROS	Robot Operating System.
SLAM	Simultaneous Localization and Mapping.
LIDAR	Light Detection And Ranging, Also known as LADAR or LiDAR.
AGV	Autonomous Guided Vehicle.
EKF	Extended Kalman Filter.
Kollmorgen	Kollmorgen Automation AB, the company at which this masters thesis project is conducted.
NDC8	Vehicle system used and developed by Kollmorgen Automation AB.
TF	ROS module used to calculate transforms between coordinate frames.
ICP	Iterative Closest Point
ICL	Iterative Closest Line

1
Introduction

1.1 Background

Since thier invention in 1953, Automated Guided Vehicles (AGVs) have been getting more and more common in factories and warehouses. Technologically, the first AGVs employed simple track-guided systems with primitive "sensors" such as bumpers and emergency arrest handles utilizing mechanical switches. The idea behind this was to replace the drivers of tractor trailers using automation and was implemented by Barrett-Cravens located in Northbrook, Illinois. While Barrett-Cravens was the first to implement an automated vehicle, the first installation was at the Motor Freight Company in Columbia, South Carolina in 1954 and was an automated tractor-trailer used for long-distance consignment shipping [1].

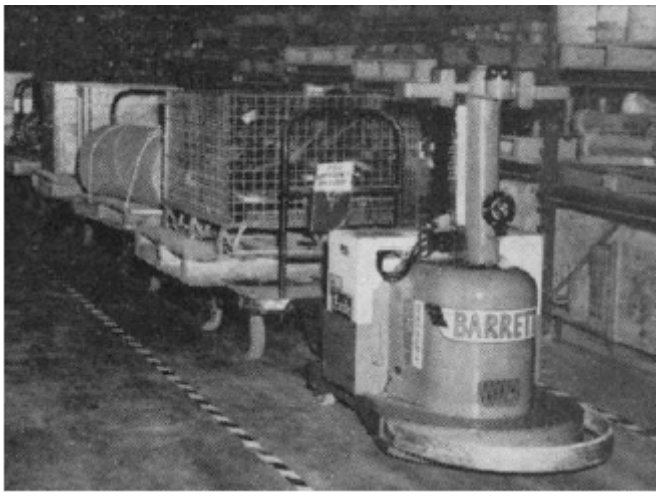

Figure 1.1: One of the first AGVs built by Barrett-Cravens in 1954. (*Source* Barrett-Cravens/Savant Automation (1958) according to [1]).

While the earliest AGVs travelled with the help of tracks, development was done to allow for following electrically conductive strips mounted to the floor, a principle now known as "inductive track guidance". This allowed the vehicle to orient itself along the induced magnetic field while driving [1].
As the development of modern electronics began to speed up, so too did the development of AGVs. With the availability of on-board computers and higher market demand, the modern AGVs [1] began to take form. For these modern AGVs there are a number of different methods of navigation used. Common methods include magnetic; which uses magnetic lines or markers in the floor, LIDAR in combination

with reflectors; which utilizes a laser scanner to detect and triangulate position with regards to a set of reflectors. More modern systems can use LIDAR scans to match the AGV position against a known map of the environment which has the benefit of not needing reflectors or other artificial markers to function. It is also possible for an AGV to use multiple of these navigation methods in combination. The primary navigation method used by the AGVs at Kollmorgen utilizes single plane LIDAR scanners to navigate with and without reflectors, which is called *Reflector* and *Natural* navigation respectively. A mix between these two can also be used.

1.2 Problem description

While the development of AGVs has come far, there are still some areas that are problematic to navigate through with only *Reflector* or *Natural* navigation. These problematic environments contain areas that are hard to make good maps of as they produce either noisy readings or change frequently. Typical examples of these types of environments are pallet racks which will change in appearance as pallets are moved from day to day. While these environments are static in relation to the moving AGV, the environment will change as pallet racks are emptied/refilled and boxes moved around. A consequence of this is that it is substantially harder to make a good environmental map without needing to re-calibrate the maps when significant changes in the environment has occured. While there are multiple ways to solve this problem, such as using artificial landmarks (reflectors) or using dead reckoning, these come with their own problems. Using artificial landmarks demand an environment which allows for their installation and that they are in the AGV's view at most times. Dead reckoning, on the other hand, work well for shorter distances but accumulates error over distance travelled.
The purpose of this project is to evaluate whether or not dead reckoning can be can be reinforced with SLAM-assistance in order to navigate parts of an environment where no maps are available. As vehicles in industrial environments have high demands in regard to positioning and accurately following specified routes in a repeatable manner. The specific question is how well a SLAM based approach can keep track of its own position with regard both to deviation in distance and angle, while traversing along a predefined trajectory in an industrial environment. The results of the SLAM tests will be compared with positioning done by odometry based dead reckoning with and without gyroscope for each test case as well as a ground truth which is the reflector navigation.

1.3 Related work

Work on and evaluation of SLAM algorithms is a common research topic. However, the usual topic of interest in these research papers is whether or not the maps made by these algorithms are good enough to be used in regards to metrics such as accuracy, robustness or how taxing the algorithm is on the CPU [2].

In [3] the authors use a few test scenarios to evaluate the mapping capabilities of SLAM (GMapping) and dead reckoning algorithms. Although a majority of the test cases were irrelevant for this project, the final two scenarios compared the trajectory of SLAM and dead reckoning with a ground truth, which is useful in this thesis.

In [4], the authors utilize SLAM algorithms to map a warehouse with the intention of using that map for navigation. This paper use mounted reflector tags to aid in mapping and navigation, whereas the work being done in this thesis use reflector navigation as a ground truth when evaluating SLAM-based trajectories. There is much research concerning the quality of SLAM algorithms [5, 6, 7]. However these focus on the quality of the produced map and not on the robot trajectories.

In [8], the authors investigate the use of LIDAR odometry to aid in using visual odometry, which is a method using cameras to calculate distance travelled by comparing images. The use of cameras to calculate motion between pictures is considered a type of dead reckoning. As this thesis attempt to improve dead reckoning using SLAM, this article is interesting even though the authors use LIDAR scan-matching and not SLAM.

In [9], the authors implement an improved version of EKF-SLAM which is based on a differential model of vehicle motion and compare it to dead reckoning. Even though a different SLAM algorithm is used compared to this thesis, the results are still relevant for understanding SLAM in general and the methods used to evaluate different algorithms.

In [10] the authors implement a SLAM algorithm to help AGVs navigate in warehouse environments where landmarks might not be available. This method has fewer downsides compared to wired (high installation cost and difficulty in changing paths) and wireless (lower safety and requiring landmarks to be installed) navigation methods. As this paper discusses the usage of SLAM algorithms to navigate an AGV in environments where landmarks are unavailable, the results of this paper are of some interest.

In [11], the authors apply a method for automatic navigation in partially structured warehouse environments where changes in the floor plans are relatively frequent. This method makes the AGV more autonomous by removing the need for paths between waypoints. This paper is interesting as the semi-structured environments studied are relevant to this thesis.

When reviewing the literature it is obvious that while there has been much research into the topic of SLAM over the years, however the metric evaluated in most cases is either the quality of the produced map or CPU usage. The authors of this thesis were unable to find any good literature where the trajectory while navigating using SLAM was being investigated. There is thus a lack of data on how well robots navigating using SLAM are able to track their trajectory in real time.

1.4 Research question

This thesis investigates how well an AGV using single plane LIDAR SLAM is able to track its own trajectory in an industrial or warehouse environment. The metric investigated specifically is the deviation from a predetermined path. The exact question is defined as follows

- How accurate and repeatable are trajectories produced by SLAM using a single plane LIDAR. This is measured with respect to deviation in both meters and radians and the variance of these values over several test runs.

This question will be answered by using reflector navigation used by Kollmorgen Automation as a ground truth while recording the scan and odometry data from a test run. This data will later be used to do simulations of the run using the SLAM algorithms GMapping and HectorSLAM to calculate AGV position. The deviation from ground truth at the end position from SLAM will later be compared to deviation at the end position by odometry with and without gyroscope.

1.5 Our contribution

The results of this thesis aim to present an evaluation between dead reckoning and ROS-based SLAM algorithms with regard to accuracy of the generated trajectories, with the reflector navigation used by Kollmorgen as ground truth. This project was done in collaboration with Kollmorgen, which is a world leading provider of AGV systems. Kollmorgen provided the test vehicle, the vehicle controller software and a location for testing.
The relevant areas of use is, for example, corridors in factories where pallets move around from day to day. Due to the semi-static nature of these kinds of environments they are hard to map as landmarks can easily get blocked with the end result being the AGV getting lost. Switching from a reflector-based navigation to SLAM-assisted dead reckoning in these kinds of areas and later switching back to the AGVs normal mode of navigation when it enters a known area is an interesting solution to this problem.
The trajectory accuracy of the SLAM-algorithms HectorSLAM and GMapping will be tested and compared against traditional dead reckoning (with and without gyroscope), with reflector navigation used by Kollmorgen as ground truth. The metric evaluated will be the deviation from the ground truth both in distance and angle.
Four different test scenarios are set up to replicate different scenarios an AGV can encounter in an industrial environment, such as driving between pallet racks or having a wall visible on only one side of the trajectory. Three different speeds (0.5m/s, 0.25m/s and 0.125m/s) were used for each scenario to replicate different scan rates of the LIDAR for each of the scenarios. Each SLAM simulation was then run five times for each test scenario in order to get some measure of how noisy each algorithm is.
Testing was conducted at the Kollmorgen facilities in Mölndal, Sweden using one of their prototype AGVs, called Charmvagn, equipped with a LIDAR scanner, wheel encoders and gyroscope. The tests were designed to replicate areas that are hard to

navigate for standard navigation systems such as warehouses where reflectors might get blocked or the environment changes due to pallets moving around from day to day, making navigation based on an existing map locally impossible.

1.6 Delimitations

This thesis presents an evaluation of ROS-based SLAM algorithms as a tool for reinforcing dead reckoning navigation in areas where traditional navigation methods cannot be used. The project will be limited to using the existing SLAM algorithms GMapping and HectorSLAM and no further development on these methods will be done. As the focus on the project is to evaluate whether or not SLAM-reinforced dead reckoning can be used to aid navigation, the intended trajectory of the AGV is assumed to be unobstructed and the environment static in relation to the AGV. The mapping quality of the different methods will not be evaluated, no development of path planning will be done and the *long corridor* problem associated with SLAM will not be explored. The *long corridor* problem refers to the fact that if a robot is moving straight down a corridor with smooth walls the laser scanner will not be able to detect any movement, see fig. 2.5 for an example of this. Vehicle dynamics will not be thoroughly investigated for other vehicles than the specific model used in this project.

1.7 Outline

The thesis has chapters dedicated to theory, platform setup, methods, results, discussion and conclusions. Theory contains information about dead reckoning, scan matching, the SLAM problem and solutions to this problem used in this thesis (GMapping and HectorSLAM). Platform contains the system architecture from the AGV down to the ROS modules used. Methods describe the different test scenarios used when testing the algorithms, the experiment setups and evaluation methods used to get results. The results chapter describe the results from the different test scenarios. Discussion will consist of discussion and evaluation of the results, choice of method as well as thoughts of future development. The final chapter will present the conclusions that has been made in regard to the results from the tests. This chapter also aims to answer the research questions stated in section 1.4.

2

Theory

This chapter presents the theory behind the dead reckoning used by the AGV as well as the theory behind SLAM and the different solutions to this used in the thesis. Dead reckoning is the term used to describe positioning done by estimating movment, in this case based on odometry and/or gyroscope. SLAM is short for Simultaneous localization and mappnig and is the process when a mobile robot does localization on a map while at the same time creating the map at runtime. Both dead reckoning and the SLAM methods used in this thesis are described in detail in this chapter.

2.1 Dead Reckoning

Dead reckoning is the term that is used to describe positioning of a robot based on estimating its velocity and integrating this over time. It is a method that has both good short-term accuracy, is computationally inexpensive and can run at high sampling rates [12]. However it is based on integrating estimated movement and will thus build up large accumulated error over distance traveled and depending on how the gyroscope is handled also over time. It is used to some degree in almost all mobile robots since it can be fused with other navigation methods that provide long term accuracy. Dead reckoning can be done with a variety of sensors, for wheeled robots the most common one is wheel encoders that are often used in conjunction with a gyroscope.

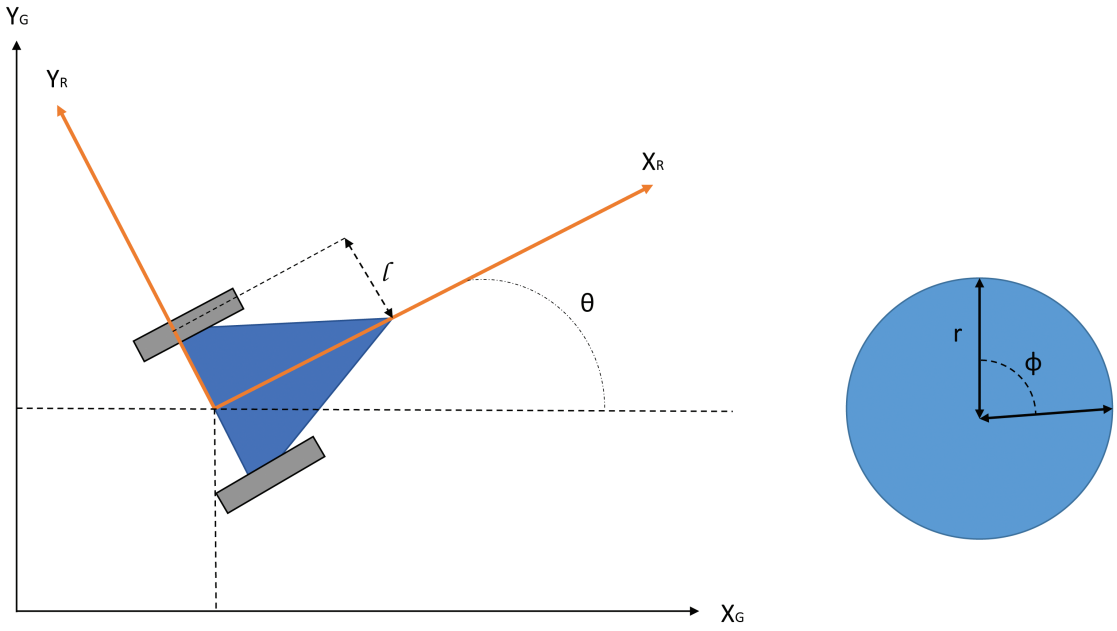

Figure 2.1: A differential drive robot and the Robot (x_R, y_R) and Global (x_G, y_G) coordinate frames. Along with the wheel of the robot with radius r and angle ϕ

The vehicle used in this project is of differential drive type and is equipped with wheel encoders and a gyroscope, the dead reckoning formulation used is explained in this section. The vehicle controller simultaneuously calculates two dead reckoning trajectories, one using only wheel encoders and one using both wheel encoders and gyroscope data. They are calculated according using eq. (2.9) and (2.10). Which are derived according to [13]:
First the forward kinematic model eq. (2.1) of the robot is defined as,

$$\dot{\xi}_R = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \dot{\phi}_1, \dot{\phi}_2). \tag{2.1}$$

Where l is the distance between the wheel and the center of rotation, r is the radius of the wheels, θ is the angle to the robot relative to the global coordinate frame and

$\dot{\phi}_{1,2}$ is the angular velocity of the right and left wheels respectively. Movement in x for each wheel is calculated using eq. (2.2) and (2.3),

$$\dot{x}_1 = \frac{1}{2} r \dot{\phi}_1, \tag{2.2}$$

$$\dot{x}_2 = \frac{1}{2} r \dot{\phi}_2. \tag{2.3}$$

Since this is a differential robot eq. (2.2) and (2.3) can simply be added to get the movement speed in x_R direction resulting in eq. (2.4)

$$\dot{x}_R = \dot{x}_{R1} + \dot{x}_{R2}. \tag{2.4}$$

Which is the $\dot{x}$ component of the robot state $\dot{\xi}_R$ as for the $\dot{y}$ component since it is a differential robot neither wheel can contribute to movement in y direction and thus this will always be zero. The subscripts $_R$ and $_G$ refer to the robot and global coordinates.
With a differential type robot the rotation can be calculated as an arc with the radius $2l$, with $\omega_{1,2}$ defined as rotational velocity for the right as left wheel,

$$\omega_1 = \frac{r\dot{\phi}_1}{2l}. \tag{2.5}$$

For the left wheel the same equation is used with the modification that since it affects the angle of the robot in opposite direction when it spins the negative angle is used instead,

$$\omega_2 = \frac{-r\dot{\phi}_2}{2l}. \tag{2.6}$$

Together equations eq. (2.4), (2.5) and (2.6) for the kinematic model for the robot:

$$\dot{\xi}_R = \begin{bmatrix} \frac{r\dot{\phi}_1}{2} + \frac{r\dot{\phi}_2}{2} \\ 0 \\ \frac{r\dot{\phi}_1}{2l} + \frac{-r\dot{\phi}_2}{2l} \end{bmatrix}. \tag{2.7}$$

Together with the rotational matrix eq. (2.8) and eq. 2.7 form the model for the estimated relative position of the robot,

$$\dot{\xi}_G = R(\theta)^{-1} \cdot \dot{\xi}_R. \tag{2.8}$$

Where $R(\theta)^{-1}$ is the transform $\dot{\xi}_R$ to $\dot{\xi}_G$

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.9}$$

2.1.1 Dead Reckoning With gyroscope

The above formula in eq. (2.8) is what is used for the dead reckoning only using wheel encoders, for the version with gyroscope the angular velocity of the robot $\dot{\phi}_R$ is measured by the gyroscope instead of being calculated using data from the wheel encoders. The equation used for gyroscope dead reckoning is thus:

$$\dot{\xi}_G = R(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\phi}_1}{2} + \frac{r\dot{\phi}_2}{2} \\ 0 \\ \dot{\phi}_R \end{bmatrix}. \tag{2.10}$$

2.1.2 Dead reckoning error sources

Dead reckoning is short term accurate but accumulates error over time, the reasons for this are outlined below. The first source of error is form the characteristics of the vehicle itself, in the equations above both the wheel radius and the distance from the wheel to the center of rotation is used. Thus any change to these will alter the result of the dead reckoning. Wheel radius can be affected by a number of factors, wear to the wheels will reduce their radius and so will heavy loads, especially on vehicles that do not have solid wheels. A big contact patch causes more slippage against the floor when a wheel is rotating around its own axis. Poor grip against the floor will also cause error to build up from wheel slipping. For the gyroscope there are two sources of error [12]. First there is drift which is that the gyroscope will slowly drift away from its initial zero, this is an inherent fault of all gyroscopes and the only real way of correcting it is to zero the gyro which means that measurements are stopped and the value from the gyro is reset. Then there is the fact that since the gyroscope measures angular velocity, one integration is needed to get the angle, thus both random noise in the measurement and the drift will be integrated and thus error will build up, illustrated by eq. 2.11 where E represents a white noise,

$$\theta = \int (\dot{\phi}_R + E). \tag{2.11}$$

In order to use gyroscopes there is thus a need to reset the values from it at regular intervals. On the vehicle used in this project this is done whenever the vehicle is stopped. Finally there is the human factor, many of the error sources described above can be reduced by careful tuning of the vehicle. This however is labour intensive and thus keeping it to a minimum is desired. As described above there can also be complicating factors like wear involved, which would require periodical re-tuning for the odometry to stay accurate. The vehicle used in this project is a best case scenario for dead reckoning, it is light, has narrow wheels and good grip factors that are important for good odometry data. In addition it has been used to test the limits of dead reckoning accuracy and thus the vehicle is very nicely tuned.

2.2 Scan Matching

Scan matching is the process of calculating the movement between a pair or sequence of laser scans based on the scans themselves. This is one of the key methods used in laser based localization. The basic idea behind it is looking at two consecutive scans and then finding the rigid transformation between them, thus finding the translation and rotation that has occurred between the scans, the most common approach to this is using ICP (iterative closest point) methods[14] or the related ICL (iterative closest line) [15] methods.

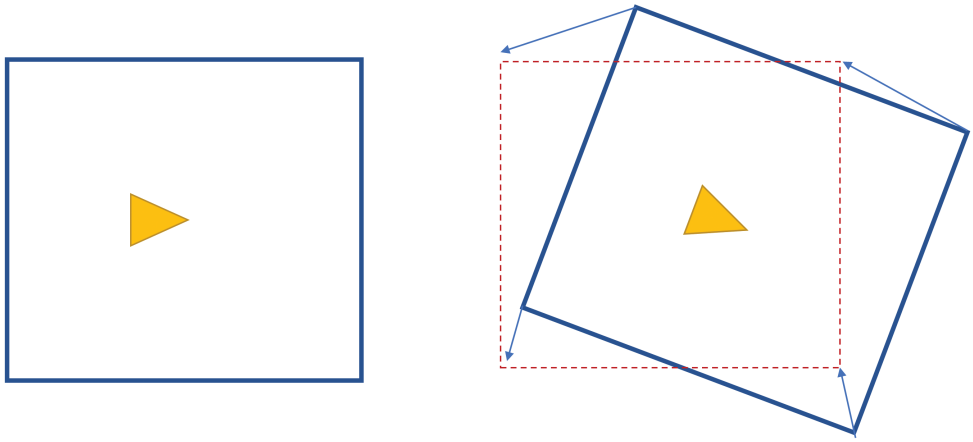

Figure 2.2: Illustration of scan matching, blue is the current scan while the red dotted rectangle is the last scan.

In figure 2.2 scan matching is illustrated, in this case the corners are identified as features. The goal of the algorithm is then to find the transformation between the last and the current scan. One thing that is important to note is that in the example above a 90 degree rotation while the robot is standing in the center of the rectangle would result in a scan that looks identical regardless of which direction the robot is rotated. Thus the result is multi-modal, this is a well known problem and ways to combat this include, higher scan frequency, more detailed scans, different choice of features and finally using more sensors in order to exclude possible modes. For details on how scan matching works and how it is used in the SLAM context refer to sections 2.3.1 and 2.3.2.

2.3 Simultaneous Localization and Mapping

Simultaneous Navigation and Mapping (SLAM) is the concept of having a robot navigate an unknown environment and both create a consistent map of it while also determining its own location on this map. This is a hard problem due to its chicken and egg nature and was for a long time regarded as the holy grail of problems in robotics [16]. Even though many parts of the problem can be considered to be solved in theory, SLAM is still one of the most important topics in the field of autonomous

robots and there are still many unsolved problems related to it [17], such as being self-tuning, fail-safe and allowing for long term autonomy. The only data available to the robot is the sensor readings while all information about the environment is calculated at runtime.

The basic SLAM problem [16] can be described as follows and is illustrated in figure 2.3.

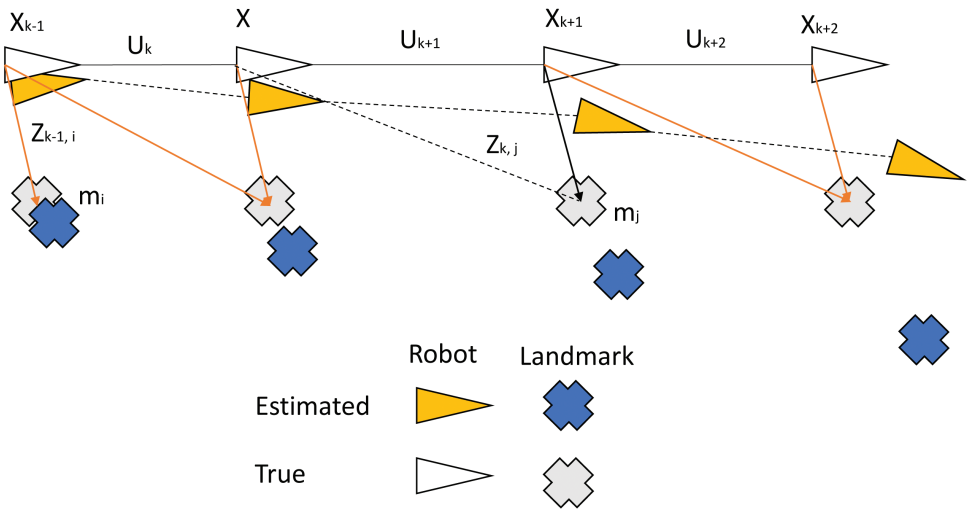

Figure 2.3: Illustration of the basic SLAM problem. This figure will be referenced when explaining the different SLAM algorithms used below. It also display a common problem in SLAM, namely that the map will become skewed in a "banana" shape.

In figure 2.3, the following parameters are

- X_k: The state vector describing the location and orientation of the robot.
- u_k: The control vector between times $k-1$ and k.
- m_i: A vector containing the location of the ith landmark, these are assumed to be time-invariant.
- z_{ik}: Observation of the ith landmark at time k.

And sets containing the full history of all these variables are also defined as X, U and M and Z respectively. The goal is to use the laser readings of the landmarks in order to establish the position of the robot on a map while simultaneously creating the map. Both map representation and what type of landmarks vary between different SLAM algorithms. This is achieved by comparing scans over time and by looking at the difference between them a trajectory through the world can be constructed with scan matching. This can be done through only scan matching or by combining other data sources such as odometry or GPS by means of sensor fusion. In sections 2.3.1 and 2.3.2 the SLAM algorithms used in this project are explained in greater detail. The basic SLAM algorithm [18] is shown in fig 2.4 and is explained in more detail below.

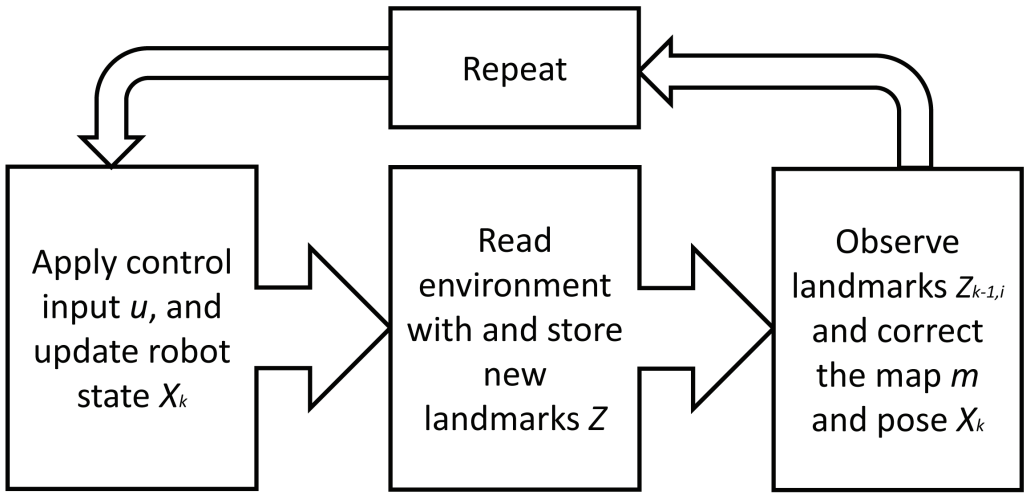

Figure 2.4: Flowchart of the basic SLAM algorithm.

1. Apply a control input u to move the robot, and update the robot state X_k.
2. Read the environment with the sensors and store new landmarks z in the map m.
3. Observe landmarks $z_{k-1,i}$ that have already been observed and use this observation to correct both the map m and the estimated pose of the robot X_k.
4. Repeat 1 through 3.

The steps above is a gross simplification of each step but the basic algorithm remains the same for all types of SLAM. One important thing to note is that in each step there is a certain uncertainty added to the system and finding ways to minimize the buildup of uncertainty in both the robot state and the map is where much of the work is being done. It also needs to be noted that 2.3 is an example of graph SLAM however the basic concept is the same for all others types of SLAM.

2.3.1 GMapping

GMapping [19] is an open source SLAM implementation based on the use of Rao-Blackwellized particle filters [20], it is based on the fastSLAM2.0 algorithm [21, 22] that has been improved and modified to allow for grid maps. FastSLAM is a so called graph SLAM algorithm and defines the maps as a series of landmarks and poses and the relationship between them, see 2.3 for an illustration of graph SLAM. This type of map representation generally contains much less information than a grid map and is less computationally taxing. As such, modifications are needed to make it work with grid maps.

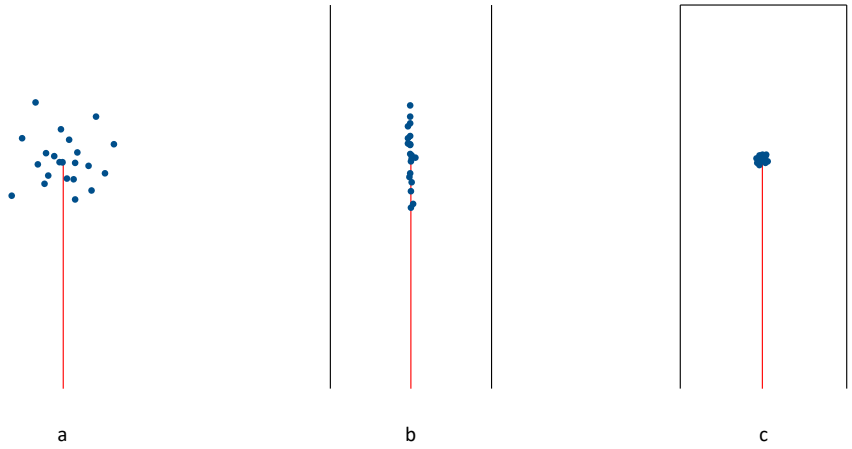

Figure 2.5: Illustration of how the laser is used by GMapping to reinforce the odometry, from left to right: a) Without any meaningful input from the laser pure odometry is used. b) With a wall visible on each side the laser can be be used to narrow down the location along the axis of movement. c) With a wall visible in three directions the position can be known with little uncertainty.

The concept behind mapping with a Rao-Blackwellised particle filter is the estimation of the joint posterior, with the map m, the trajectory $x_{1:t} = x_1, ..., x_t$, the observation data $z_{1:t} = z_1, ..., z_t$ and the odometry information $u_{1:t-1} = u_1, ..., u_{t-1}$.

$$p(x_{1:t}|z_{1_t}, u_{1:t-1}) \tag{2.12}$$

The filter makes use of the factorization

$$p(x_{1:t}|z_{1_t}, u_{1:t}) = p(m|x_{1_t}, z_{1:t}) \cdot p(x_{1:t}|z_{1_t}, u_{1:t-1}) \tag{2.13}$$

With this factorization the trajectory can be estimated and then used to compute a map given the trajectory. This is what is known as Rao-Blackwellization and eq. (2.13) can usually be calculated efficiently [23].

To estimate the posterior over all the potential trajectories a particle filter is used, each particle presents a potential vehicle trajectory that has it's own map associated to it which is built from the observations and the trajectory. Of these particles the ones that have a low likelihood of being true are then eliminated and the ones with high likelihood are kept.
A common approach in localization is to use smoothed likelihood functions, however the laser especially does not have a smooth likelihood function instead it is peaked around one or more possible modes. To solve this the last observation made by the sensor is integrated into the proposal which allows for sampling only of the meaningful regions. Eq. (2.15) is the optimal proposal distribution with respect to

the particle weights [24] while the weights w^i for each particsle then can be expressed as (2.16)

$$p(x_t|m_{t-1}^{(i)}, x_{1_t}^{(i)}, zt, u_{t-1}) = \tag{2.14}$$

$$\frac{p(z_t|m_{t-1}^{(i)}, x_t)p(x_t|x_{t-1}^{(i)}, u_{t-1})}{p(z_t|m_{t-1}^{(i)}, x_{t-1}^{(i)}, u_{t-1})}, \tag{2.15}$$

$$w_t^{(i)} = w_{t-1}^{(i)} \cdot \int p(z_t|x')p(x'|x_{t-1}^{(i)}, u_{t-1})dx'. \tag{2.16}$$

In order to efficiently draw the next generation a Gaussian approximation

$$\mathcal{N}(\mu_t^{(i)}, \Sigma_t^{(i)}). \tag{2.17}$$

is computed based on the locally estimated posterior (2.14) around the maximum likelihood function created by the scan matching. For each particle i both parameters $\mu_t^{(i)}$ and $\sum_t^{(i)}$ are calculated individually,

$$\mu_t^{(i)} = \frac{1}{\eta^{(i)}} \cdot \sum_{j=1}^{K} x_j \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}), \tag{2.18}$$

$$\Sigma_t^{(i)} = \frac{1}{\eta^{(i)}} \cdot \sum_{j=1}^{K} x_j \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}) \cdot (x_j - \mu_t^{(i)})(x_j - \mu_t^{(i)})^T. \tag{2.19}$$

where $\eta^(i)$ is a normalization factor for the Gaussian parameters,

$$\eta^{(i)} = \sum_{j=1}^{K} p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_j|x_{t-1}^{(i)}, u_{t-1}). \tag{2.20}$$

Thus a closed form approximation of the optimal proposal is obtained and using this proposal distribution the weights can be computed as eq. (2.21) instead of (2.16),

$$w_t^{(i)} = w_{t-1}^{(i)} \cdot \eta^{(i)}. \tag{2.21}$$

The value $\mathcal{N}_{eff}$ is a factor [25] that is used to estimate how well the particles represent the posterior. When this factor drops below $N/2$ where N is the number of particles resampling is performed,

$$\mathcal{N}_{eff} = \frac{1}{\sum_{i=1}^{N}(\tilde{w}^{(i)})}. \tag{2.22}$$

The authors of the GMapping algorithm [19] describe it as follows. Whenever a new set of measurements (u_{t-1}, z_t) is available the proposal is individually calculated for each particle resulting in the following algorithm:

1. A initial guess of the robot pose $x_t^{'(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$ for each particle i is obtained from the previous pose $x_{t-1}^{(i)}$ of that particle and the odometry data u_{t-1} collected since the last filter update. In this context $\oplus$ is the standard pose compounding operator [26].

2. A scan matching algorithm is executed on the map $m_{t-1}^{(i)}$ starting from the initials guess $x_t^{'(i)}$ the search of the scan marching is limited to a region around $x_t^{'(i)}$. The scan matcher used by GMapping is "vasco" from the "Carnegie Mellon Robot Navigation Toolkit"[1]. If the scan matching fails the pose is computed using only odometry data and steps 3 and 4 are skipped.
3. A set of sampling points is selected in an interval around the pose $\hat{x}_t^{(i)}$ reported by the scan matcher. Based on this the mean and covariance matrix of the proposal are computed by pointwise evaluation of the target distribution $p(z_t|m_{t-1}^{(i)}, x_j)p(x_j|x_{t-1}^{(}i), u_{t-1})$ in the sampled positions x_j. During this stage the weighting factor $\eta^{(i)}$ is computed according to (2.20).
4. The new pose $x_i^{(i)}$ of the particle i is drawn from the Gaussian approximation (2.17) of the improved proposal distribution. Fig (2.5) shows how this distribution is found intuitively.
5. Update importance weights (2.21).
6. The map $m^{(i)}$ of particle i is updated according to the drawn pose $x_t^{(i)}$ and the observation z_t.

After computation of the next iteration of samples a resampling step is carried out depending on the value of (2.22).

2.3.2 HectorSLAM

HectorSLAM (**He**terogeneous **C**ooperating **T**eam **O**f **R**obots) is part of a series of ROS modules developed at TU Darmstadt for urban search and rescue operators [27]. HectorSLAM only uses a laser scanner for the SLAMing part, however other sensors such as gyroscope, accelerometer and GPS are used by other parts of the Hector suite. This means that it is resistant to bad odometry but is instead reliant on high quality, high frequency laser scans and is especially sensitive to fast rotations of the robot. It has the property that it is able to produce high quality maps without loop closure, something that is of interest to this study. HectorSLAM [28] works by matching the laser scan to the map-so-far and this is done by using a Gauss-Newton approach inspired by computer vision. This saves computation time since there is no need to associate the scan beams with each other or doing a pose search. Aligning scans with the map-so-far also means that the scan is implicitly aligned with all previous scans.

The map in HectorSLAM is defined as a grid map which by nature is discrete, this limits the precision that can be achieved since it does not allow for computation of interpolated values or derivatives. For this reason the following scheme is employed, each continuous point on the map can be approximated using the four closest discrete points. Where M is the map and P_m is a point,

$$\begin{aligned} M(P_m) \approx & \frac{y-y_0}{y_1-y_0}\left(\frac{x-x_0}{x_1-x_0}M(P_{11}) + \frac{x_1-x}{x_1-x_0}M(P_{01})\right) \\ & + \frac{y_1-y}{y_1-y_0}\left(\frac{x-x_0}{x_1-x_0}M(P_{10}) + \frac{x_1-x}{x_1-x_0}M(P_{00})\right). \end{aligned} \tag{2.23}$$

[1] http://carmen.sourceforge.net/intro.html

And the derivatives can be approximated as:

$$\begin{aligned}\frac{\partial M}{\partial x}(P_m) \approx & \frac{y - y_0}{y_1 - y_0}(M(P_{11}) - M(P_{01})) \\ & + \frac{y_1 - y}{y_1 - y_0}(M(P_{10} - M(P_{00})),\end{aligned} \tag{2.24}$$

$$\begin{aligned}\frac{\partial M}{\partial y}(P_m) \approx & \frac{x - x_0}{x_1 - x_0}(M(P_{11}) - M(P_{10})) \\ & + \frac{x_1 - x}{x_1 - x_0}(M(P_{01}) - M(P_{00})).\end{aligned} \tag{2.25}$$

The 2D HectorSLAM algorithm [28] used in this project works as follows. Define the parameters p_x and p_y for the pose in x and y respectively and the map M and scan data $S_i(\xi)$, which contains the end point coordinates of each of the laser beams $s_i = (s_{i,x}, s_{i,y})^T$. The goal of the scan matcher is to find the rigid transformation $\xi = (p_x, p_y, \psi)^T$ that minimizes

$$\xi^* = \underset{\xi}{argmin} \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2. \tag{2.26}$$

I.e. the best alignment of that laser scan with the map. $S_i(\xi)$ is the end point coordinates in global coordinates and is found using the following expression:

$$S_i(\xi) = \begin{pmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,y} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix}. \tag{2.27}$$

Thus $M(S_i(\xi))$ returns the coordinates of the beam end points in the same frame of reference as the map. The goal is then given some starting estimate of ξ to estimate the $\Delta\xi$ that minimizes the error according to

$$\sum_{i=1}^{n} [1 - M(S_i(\xi))]^2 \to 0, \tag{2.28}$$

through Taylor expansion and solving for $\Delta\xi$ the Gauss-Newton equation for the minimization problem is acquired:

$$\Delta\xi = H^{-1} \sum_{i=1}^{n} \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]^T [1 - M(S_i(\xi))]. \tag{2.29}$$

with

$$H = \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]^T \left[\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \right]. \tag{2.30}$$

The map gradient $\nabla M(S_i(\xi))$ is provided by 2.23 which together with 2.26 forms

$$\frac{\partial S_i(\xi)}{\partial \xi} = \begin{pmatrix} 1 & 0 & -sin(\psi)s_{i,x} & -cos(\psi)s_{i,y} \\ 0 & 1 & cos(\psi)s_{i,x} & -sin(\psi)s_{i,y}. \end{pmatrix} \tag{2.31}$$

using the map gradient and 2.31 the Gauss-Newton equation 2.17 can then be evaluated. Since this works on non-smooth linear approximations of the map gradient

convergence towards a minimum is not guaranteed. In order to combat this an approach using several maps of progressively coarser resolution inspired by work in computer vision is used, where the scan alignment is started at the coarsest level first and then continued using the finer maps.

2.4 Summary

In this chapter the theory behind the methods used in this chapter have been presented. It is important to understand the relation between them. The odometry in this thesis is at its core based on calculating the trajectory of a mobile robot based on how much the electric motors move, this can then be extended by the use of a gyroscope.
Scan matching is the process of calculating movement based on two or more consecutive scans, since modern laser scanners are both very accurate and have high frequency this can be very effective. However if a robot is localizing itself on a map with the use of laser scans it will have no way to differentiate between locations that produce identical scans. For example if using scan matching in an office environment containing identical cubicles localization based on scan matching can result in "teleportation" between these cubicles. A laser scanner also produces a large amount of data, often requiring downsampling of either resolution or frequency in order to be able to work in real time.
SLAM is at a basic level an optimization problem where the distance between the map and the current view has to be minimized like eq. (2.28). The best way to solve an optimization problem effectively is to provide it with a good initial guess. This is done in different ways by the the GMapping and HectorSLAM algorithms. GMapping uses the odometry in order to provide an initial guess and try to eliminate multimodal scan results, HectorSLAM uses an approach based on keeping several maps of different resolution in memory simultaneously and optimizing with respect to each map in order starting with the lowest resolution map. The result of the low resolution optimization is then used to provide the initial guess of the next optimization problem. HectorSLAM uses the scanner at full frequency in order to eliminate multimodal results.
In the next chapter the hardware and software setup used for the experiments conducted for this thesis is laid out and explained in detail.

3

Platform

This chapter aims to describe the hardware and software setup used during the tests. The first section describes the system architecture used in the test while the second section describes the experiment setup.

3.1 Test equipment

This section aims to describe the AGV used in the tests and the software architecture used by it. The first subsection aims to describe the hardware of the AGV while the second shortly aims describe the system used by Kollmorgen. The third subsection give an introduction to the ROS framework and the final subsection aims to describe the different ROS packages used to perform and evaluate the tests. Figure 3.1 shows the setup that was used to log the data from the experiments, the AGV was controlled using the Kollmorgen NDC8 system from the Windows laptop where the dead reckoning data was also logged, while data was being logged by ROS and sent to the laptop running ROS.

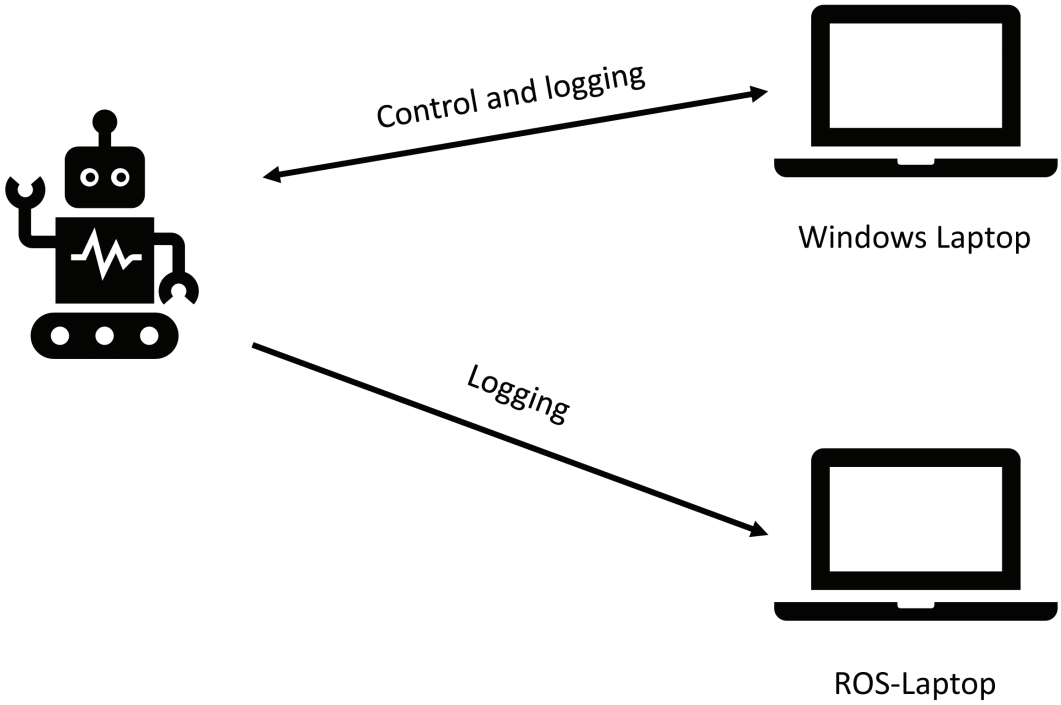

Figure 3.1: Figure describing the setup used for data logging

3.1.1 Charmvagn

In the project, a differentially steered AGV called a Charmvagn is used. The Charmvagn is a small-scale AGV used for testing and demonstration but uses the same controller and software used in real life applications. The sensors used in the tests were a Light Detection and Ranging (LIDAR) sensor, wheel encoders and a gyroscope. A LIDAR sensor uses a laser to measure the distance to objects in different directions. Wheel encoders report the angular velocity of the wheels and is used for dead reckoning when calculating the distance travelled from a starting position while the gyroscope measure the rotational acceleration per axis. The LIDAR used is a Kollmorgen LS2000[1], which is the same type of scanner used by other AGVs at Kollmorgen. This specific LIDAR has an angular resolution of 1 milliradian at a rate of 20 scans per second and a maximum range of 25 meters with an accuracy of ±25mm and a maximum sweep of 360 degrees. The LS2000 provide a resolution of 6300 values per scan, with an accuracy of ±0.87mrad, which is higher than what is used by many applications. In an earlier project [2], the total amount of data was limited to 630 values per scan to minimize stress on the system as the data is transferred over WLAN. However the whole scan sweep is unusable due to the rear mounted forks so these scans are removed by applying a filter in ROS to the scan data removing scans that are "too close" to the vehicle. While this approach can cause other problems since it also means that the vehicle will not detect other ob-

[1] `http://npm-ht.co.jp/wordpress/wp-content/uploads/2018/06/ls2000.pdf`

jects that are very close it was deemed acceptable for testing purposes. The vehicle used can be seen in figure 3.2.

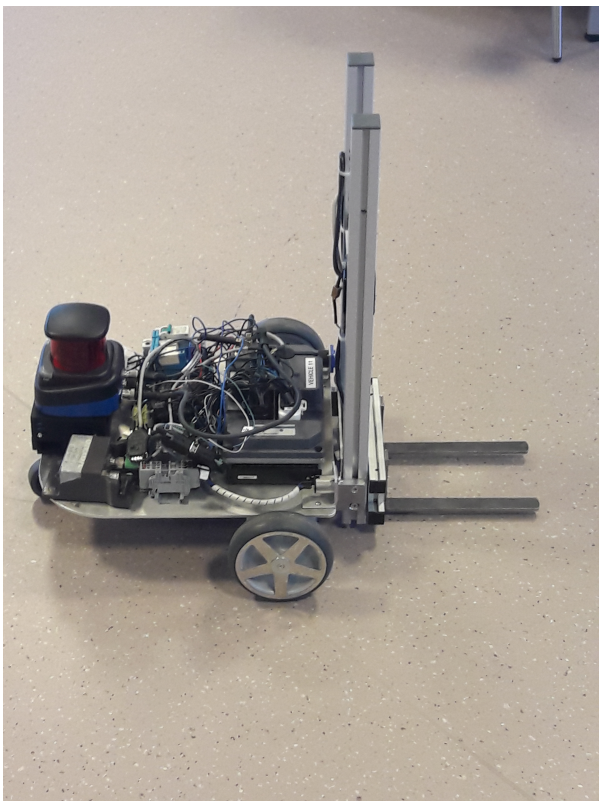

Figure 3.2: The AGV used in the project. It is equipped with a LIDAR scanner, gyroscope and wheel encoders on both wheels.

3.1.2 Kollmorgen system background

At Kollmorgen Automation the AGVs come equipped with a generic control system called NDC8 which include a vehicle controller, navigation sensors, displays, vehicle software and system software used for diagnostics and configuration. The data gathered from each run can be saved onto a computer and replayed at a later occasion.
As the Kollmorgen AGVs generally do not use ROS as part of their software the vehicle controller software has been extended to be compatible with ROS software as part of an earlier project. This has been done by adding ROS functionality on top of the existing software, allowing for the systems to be run in parallel.

3.1.3 Robot Operating System

Robot Operating System (ROS) is an open source framework for developing robot software [29]. It is made with a modular structure that is designed to allow for easy integration of different modules and running several of them at the same time.
The structure of ROS is that of a number of servers called nodes that are talking to each other with messages in a peer-to-peer structure. The nodes can be run both

locally on a single machine or networked over several units using either LAN or WLAN. This allows computationally heavy tasks to be run "offboard" the robot. **Nodes** are the part of the system which perform the actual computation in ROS and these communicate via topics. This is done by the node publishing data to a topic which will pass it on to any node that is subscribing to it. Each node may have several topics that it subscribes and publishes to and each topic may send/receive data from several nodes. Generally neither publishers and subscribers are aware of each others existence. Keeping track of these nodes is the **Master** whose role is to enable the nodes to locate each other. In this project all tasks related to mapping and navigation are run on a laptop while the AGV in essence only delivers data. This structure is designed to allow for easy debugging as existing modules can be combined with new code since there is minimal dependencies between the nodes. The ROS modules used in this thesis is the GMapping and HectorSLAM packages to perform SLAM, rosbag to record and play data and tf to calculate transforms between the coordinate frames. The simulations were visualized using RViz[2], which is a package allowing for visualization of sensor data and trajectories during simulation.

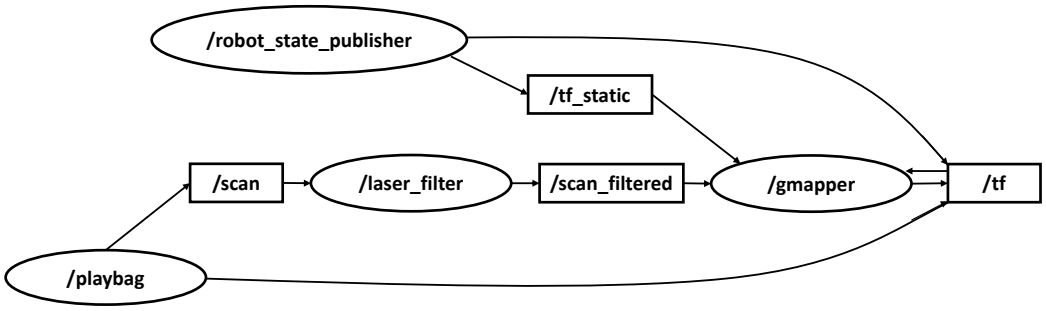

Figure 3.3: Basic flow of the ROS navigation stack while running GMapping using data from a bagfile, with some components omitted.

3.1.4 ROS packages

This section aims to shortly describe the different ROS packages and their required data used in this thesis. The main packages used that will be described here is the *tf*, *rosbag*, *GMapping* and *HectorSLAM* packages.

[2] `http://wiki.ros.org/rviz`

3.1.4.1 Tf

Tf[3] is one of the more important functions of ROS when using or simulating robots with many parts. This system keeps track of all the coordinate frames in relation to each other. This is an essential tool for keeping track of a robots many parts. As the interesting metric to be evaluated is the location of the AGV in the map, the relevant transform looked at is the transform between the *map* coordinate frame and the *base_link* coordinate frame. This package is also where things such as the location of wheels and scanners are kept track of.

3.1.4.2 Rosbag

In order to record scan and odometry data to use for the SLAM simulations, a ROS package named rosbag[4] was used. This package is able to record data from published topics and saving them in the .bag-format for later use. While recording, rosbag subscribes to the desired topics to save the data to the computer. When replaying the data, rosbag instead act as a publisher and publishes the data for other nodes to use.
The data that was recorded was chosen to be the laser scan data from the LIDAR and odometry data due to these being the most relevant when simulating GMapping and HectorSLAM.

3.1.4.3 GMapping

The GMapping[5] package for ROS is focused on doing SLAM using the OpenSLAM GMapping method for laser-scan based SLAM. The required data for this package is the scans from the laser scanner attached to the robot and the odometry information.

3.1.4.4 HectorSLAM

The HectorSLAM[6] package for ROS is a package utilizing the HectorSLAM method described in the theory chapter. This package only utilize laser scan data to function.

3.2 Experiment setup

To test the different algorithms the AGV was set to run four different scenarios chosen to represent different types of environments. The different trajectories used in these scenarios were created with *Layout Designer*, a tool made by Kollmorgen to create paths for their AGVs. As the reflector navigation need to identify several reflectors to not consider itself "lost", reflector calibration was done using *Reflector Surveyor*, designed by Kollmorgen to help creating reflector layouts, to ensure that the trajectories could be travelled without losing navigation.

[3] `http://wiki.ros.org/tf`
[4] `http://wiki.ros.org/rosbag`
[5] `http://wiki.ros.org/gmapping`
[6] `http://wiki.ros.org/hector_slam`

Reflector navigation by Kollmorgen was set to be considered ground truth as it has a very high accuracy, millimeter precision in good conditions. Good conditions in this case is low distance to the reflectors, good vision and vehicle tuning. These conditions are fullfilled in the tests done for this thesis. During simulation, the scan range of the LIDAR was limited to a maximum of four meters in order to limit the area seen by the AGV in the different test scenarios. Each test scenario was run at three different speeds in order to emulate different scan rates, as the scan data is streamed over WLAN and using the full data is not feasible for practical reasons. The chosen speeds were set to 0.5m/s, 0.25m/s and 0.125m/s. With a scan data frequency of 16Hz, this works out to 32, 68 and 128 scans per meter moved. In every test run, the AGV was set to navigate using reflector navigation while the data was logged on an external computer using rosbag for later use.

3.3 Summary

In the above chapter the hardware and software setup used to conduct the testing is explained. The tests were conducted on a Charmvagn AGV which is a small scale differentially steered AGV used by Kollmorgen as a test platform. It uses exactly the same controller and sensor platform as a full scale AGV, in this case wheel encoders, gyroscope and a single plane LIDAR.
In order to create the SLAM trajectories ROS (Robot Operating System) was used and ran a publishing node on top of the regular vehicle control software. Thus both the regular vehicle control software and ROS are able to run in parallel. Since ROS is based on Linux while the Kollmorgen system is Windows based a pair of laptops, one running Ubuntu and one running Windows was used in order to log the data from both the dead reckoning and SLAM.
In the next chapter how this setup is used to evaluate the SLAM trajectories is described in detail.

4
Methods

This chapter aims to describe the different test scenarios used in the experiments. In this thesis, four test scenarios were constructed in order to test SLAM trajectories. In the first test, the AGV was set to drive in a U-shape alongside a wall with no objects along the trajectory. The second test case had the same trajectory, but with boxes placed alongside the trajectory in order to replicate pillars. In the third test case, the AGV was set to drive in a straight line with boxes placed alongside the trajectory in order to replicate a corridor with pillars. In the final test case, the AGV drove zig-zag through several rows of boxes in order to replicate rows of pallet racks. To evaluate the SLAM algorithms, data logged to a ROS-laptop and later simulated five times for each test scenario and speed in order to ensure the same test data was used for both algorithms.

4.1 Test cases

In order to properly evaluate how well the SLAM algorithms perform in regard to localization, several scenarios were created. These were designed to replicate a number of common industrial environments that are hard to properly map and/or setup reflector navigation in, namely warehouse environments that change over time due to goods being moved around. To replicate these areas, boxes were placed around the trajectory in order to simulate evenly spaced pillars and pallet racks commonly found in these kinds of industrial environments. All testing was done at the Kollmorgen test facilities in Mölndal, Sweden.

4.1.1 Wall visible on one side

In this test, the AGV was set to drive alongside a wall in a U-shape with no other obstacles visible. This was done to replicate an industrial environment that is totally empty of goods. The trajectory layout used for, and map created from, this test can be seen in figure 4.1. The distance travelled is approximately 40 meters.

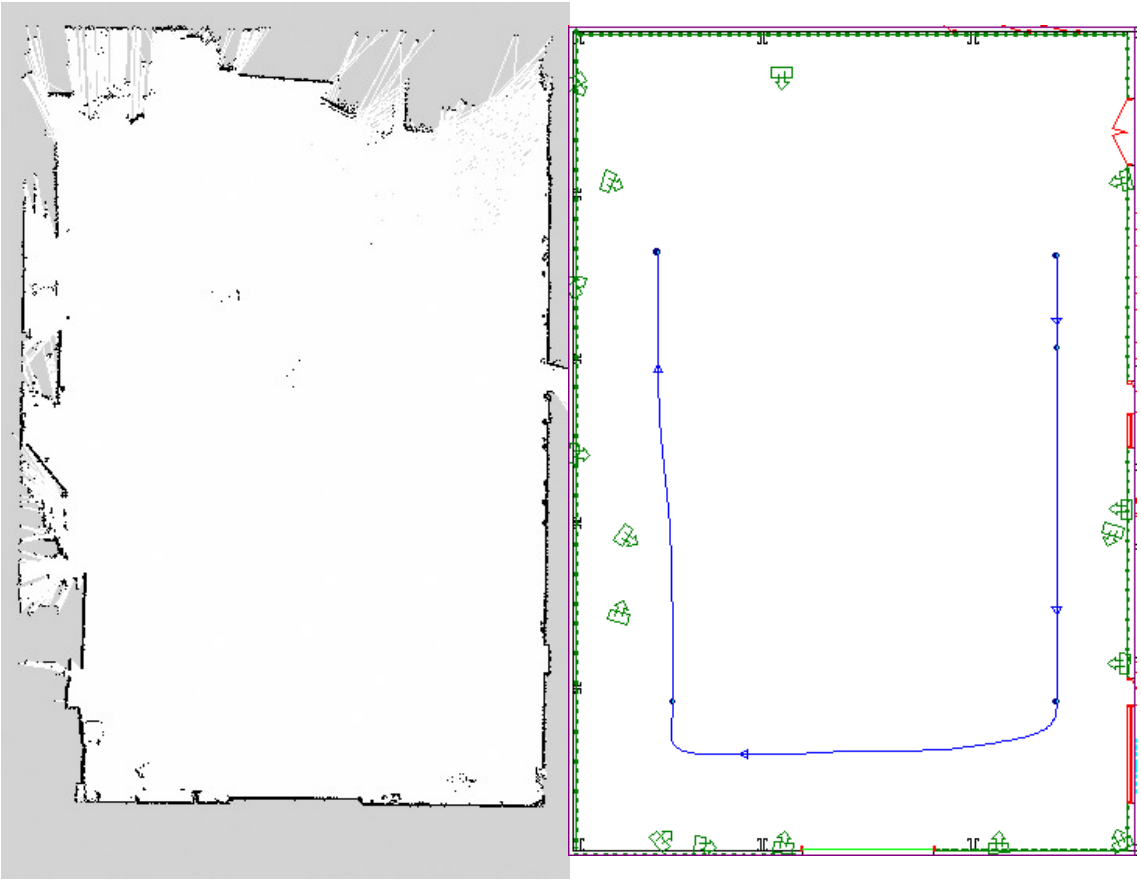

Figure 4.1: Map of the environment (made with GMapping) for the first test case (left) and trajectory layout of the test case. The AGV start at the point in the top right and travel toward the top left. The green arrows are the positioning of reflectors.

4.1.2 Wall on one side with pillars on the other

This test travelled with the same trajectory as the test in 3.3.1 with the main difference being that boxes were added to the inside of the trajectory to replicate pillars. This test was designed to replicate an AGV moving through a warehouse with the wall visible on one side and some objects visible on the other. The trajectory layout used for, and map created from, this test can be seen in figure 4.2. The total distance travelled is approximately 40 meters.

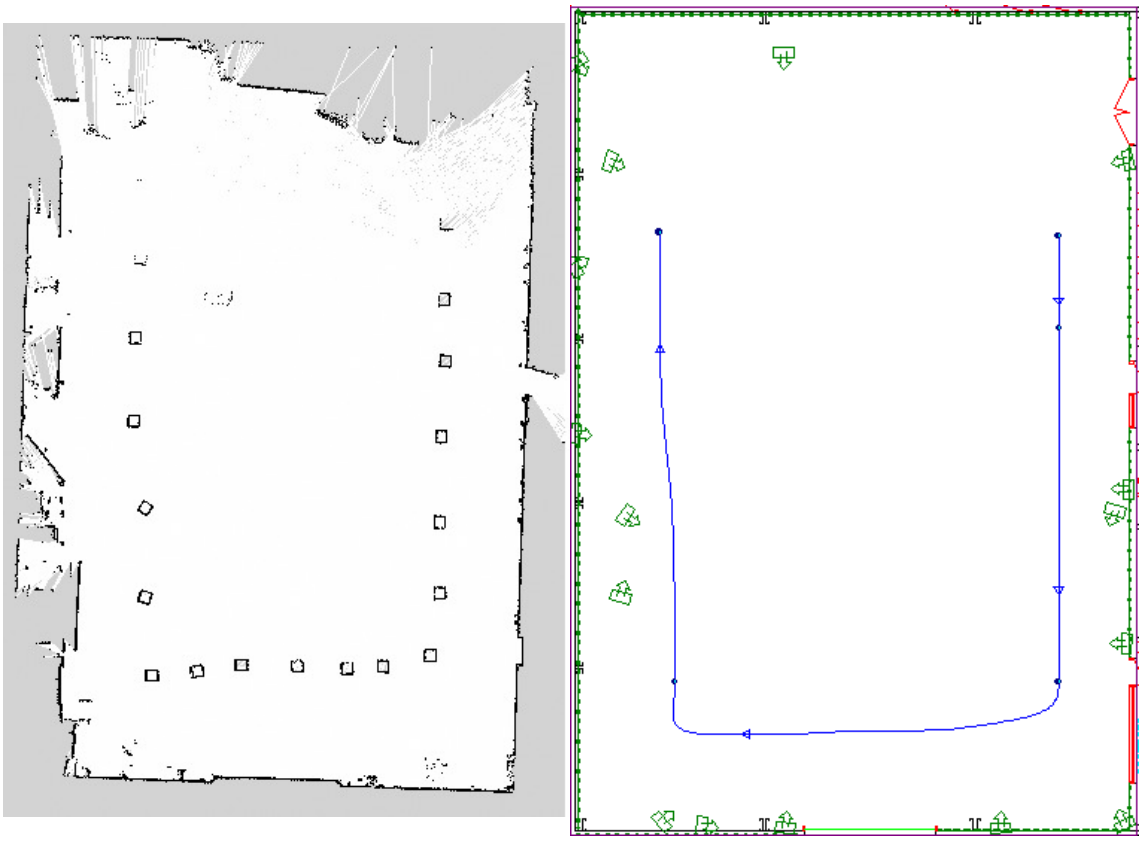

Figure 4.2: Map of the environment (made with GMapping) for the second test case (left) and trajectory layout of the test case. The AGV start at the point in the top right and travel toward the top left. The green arrows are the positioning of reflectors.

4.1.3 Corridor with pillars on both sides

In this test the AGV drove in a straight line between two rows of boxes in order to replicate an empty warehouse where no real walls are visible to the laser scanner. The boxes were placed evenly spaced along both sides of the AGV trajectory to replicate pillars. The trajectory used, and map created from, can be seen in figure 4.3. The distance travelled in this test is approximately 16 meters.

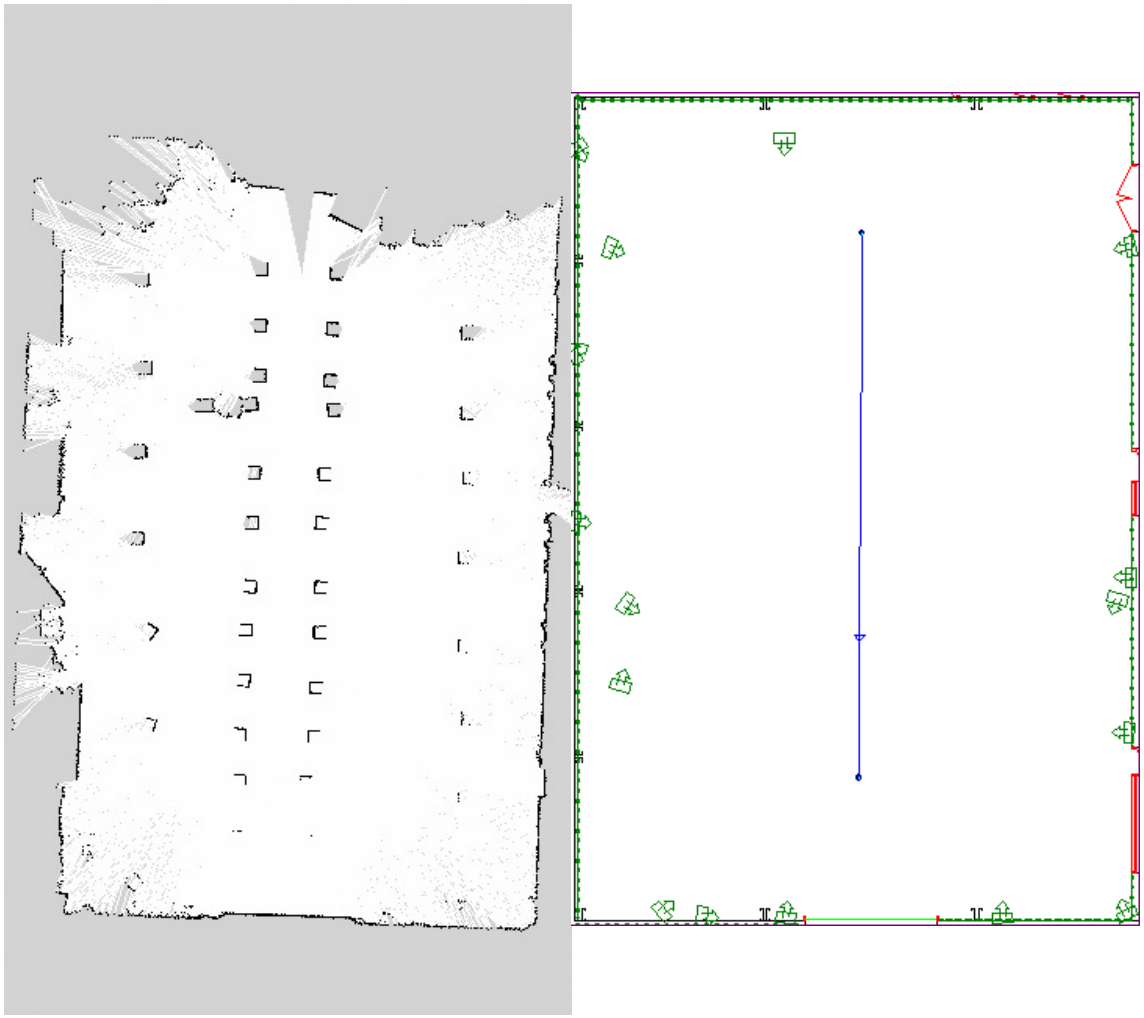

Figure 4.3: Map of the environment (made with GMapping) for the third test case (left) and trajectory layout of the test case. The AGV starts at the top and moves downward. The green arrows are the positioning of reflectors.

4.1.4 Zig-zag through corridors

In this test the AGV was driven in zig-zag between a series of rows of boxes which were placed in between the trajectories. The goal was to emulate a scenario where an AGV is navigating through rows of pallet racks in a warehouse. The boxes were placed parallel to the traversed trajectory and were placed so that the AGV still had room to be able to drive freely into the next row without any collisions. The map created and layout used can be seen in 4.4. The distance travelled in this test is approximately 75 meters.

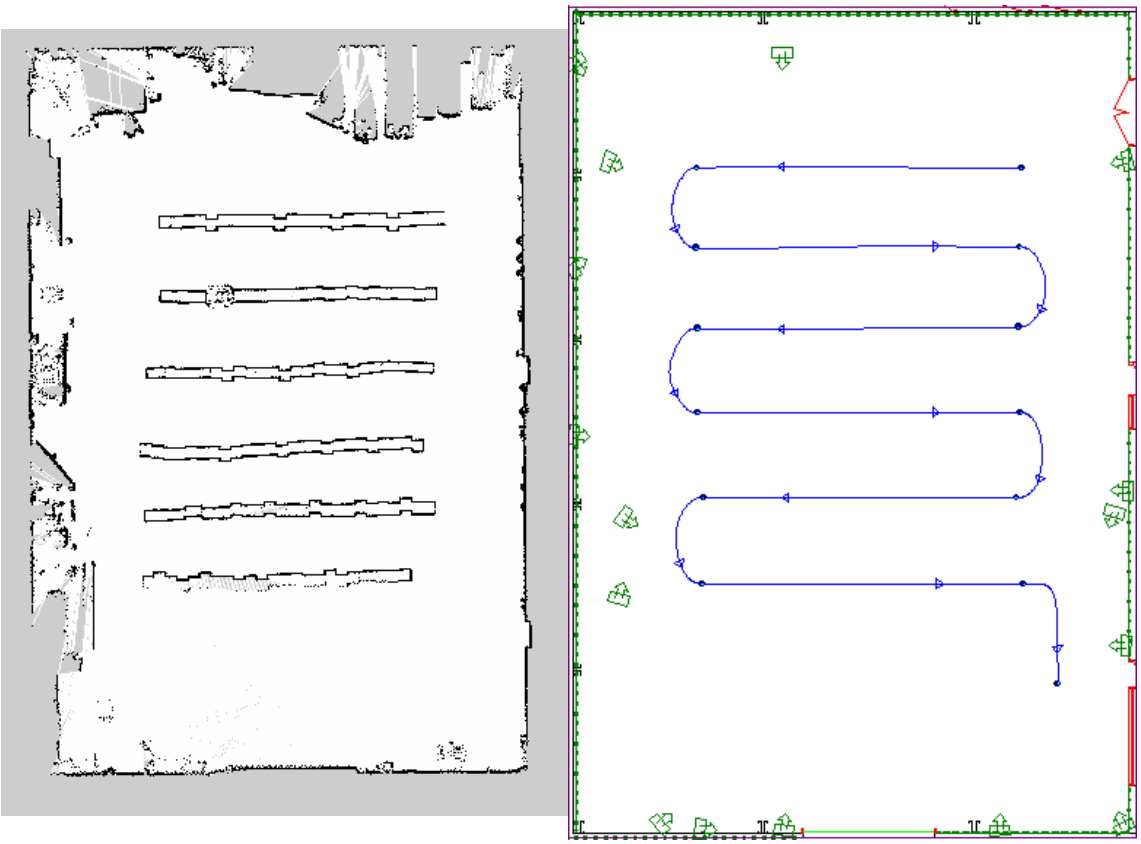

Figure 4.4: Map of the environment (made with HectorSLAM) for the fourth test case (left) and trajectory layout of the test case. The AGV starts at the top right and moves toward the bottom right. The green arrows are the positioning of reflectors.

4.2 Evaluation methods

After the physical tests were done the data from both ROS, using rosbag, and Kollmorgen NDC8 system was saved. For each of the SLAM algorithms, the actual localization was simulated five times for each test scenario and speed. This was done to both save time in physical testing and to ensure that both the SLAM algorithms were using the same scan data for a specific test run. While running the algorithms on the different logged data, the transform from the *map* frame to the *base_link* frame, representing the location of the AGV in the map in a specific point of time, for the whole run were saved. This data was later used to calculate both the trajectories travelled according to the SLAM algorithms and the deviation from ground truth using Matlab. This data was then used to compare the SLAM algorithms with the Kollmorgen NDC8 built-in dead reckoning with and without the use of a gyroscope. It should be noted that the AGV used has been tuned to have very good odometry. As it has thin wheels, good grip against the floor and has a light weight. This together with the good floor conditions in the testing area create very good conditions for dead reckoning. A consequence of this is that the evaluation of dead reckoning should be regarded as a best case-scenario as a full-scale AGV will most likely not have as well adjusted odometry or good floor conditions.

4.3 Summary

In this chapter the different test scenarios used in the thesis was presented. The four different scenarios were designed to replicate common industrial environments. This was done by placing boxes around the trajectories travelled by the AGV. In the first two tests the AGV drove in a U-shape along a wall with the difference being no other obstacles in the first test while boxes where placed along the trajectory in the second.
The third test case had the AGV drive straight with a set of boxes on both sides of the trajectory in order to simulate a corridor with evenly spaced pillars. In the fourth test case, the AGV was set to drive in zig-zag through rows of boxes placed in parallel to the trajectory in order to replicate rows of pallet racks in a warehouse.
To evaluate the results from the tests, data recorded from the test runs were simulated five times for each SLAM algorithm in order for both GMapping and HectorSLAM to work with the same data sets. The simulated trajectories were later compared to dead reckoning from the Kollmorgen NDC8 system.
As the AGV used has very well tuned odometry due to its light weight, thin wheels and good grip against the floor, the resulting dead reckoning data was considered to be a best case scenario as a full-scale AGV would have less well adjusted odometry and worse floor conditions.
In the next chapter, the results from the tests and simulations will be presented.

5

Results

This chapter aim to describe the different results from the different tests with different set speeds. For each test case there are a series of plots, these show the calculated trajectory traveled by the AGV for the different test runs calculated by the two SLAM methods. For each set of test data the SLAM algorithm was run five times in order to even out the stochastic nature of these methods. The deviation in meters and radians from ground truth at the end of the runs for all speeds and algorithms are presented in tables for each scenario. The resulting mean and variance across the five simulations are also presented in these tables. In the figures, the trajectories of all five runs of the simulations for the specific scenarios are shown together with ground truth and trajectories from odometry both with and without gyroscope.

5.1 Test 1 - Wall visible on one side

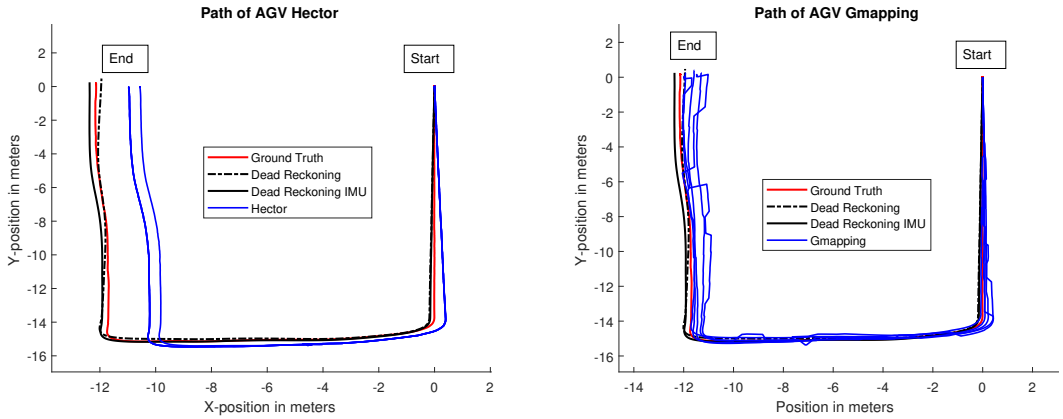

Figure 5.1: Figures showing trajectories from the fast speed run of the wall without pillars course. All five runs are shown in the figures.

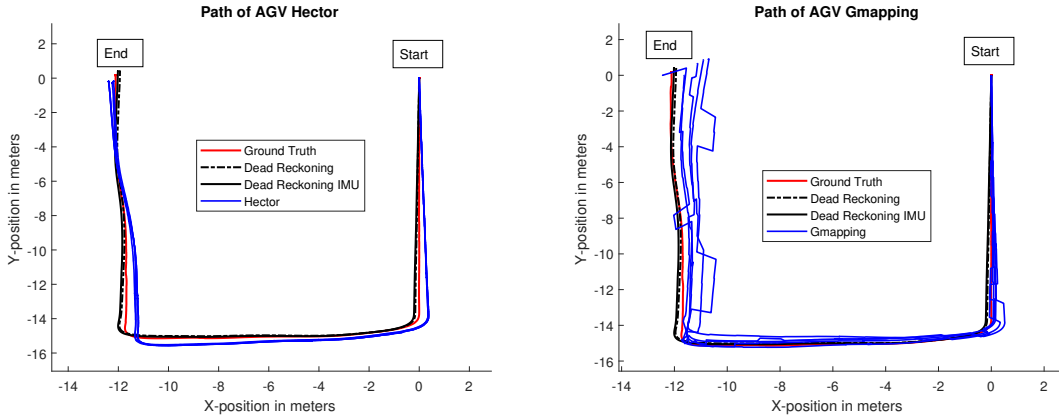

Figure 5.2: Figures showing trajectories from the medium speed run of the wall without pillars course. All five runs are shown in the figures.

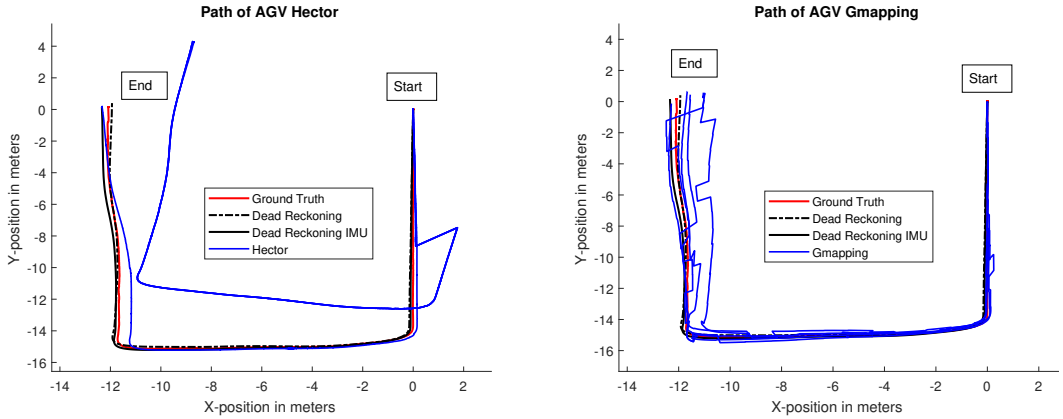

Figure 5.3: Figures showing trajectories from the slow speed run of the wall without pillars course. All five runs are shown in the figures.

SLAM Simulation	1	2	3	4	5	Mean $[m]$	Variance $[m]$
GMapping 0.5m/s	0.6059	0.2250	0.8305	0.6480	0.2030	0.5025	0.0765
GMapping 0.25m/s	0.3742	1.5669	1.3889	0.5142	1.0858	0.9860	0.2767
GMapping 0.125m/s	0.6092	0.3467	0.6524	1.1289	1.0715	0.7617	0.1096
Hector 0.5m/s	1.1902	1.5843	1.1888	1.1912	1.1889	1.2687	0.0311
Hector 0.25m/s	0.3680	0.4841	0.4754	0.4778	0.4535	0.4518	0.0023
Hector 0.125m/s	0.2443	5.3189	5.3191	5.3141	5.3191	4.3031	5.1482
Dead Reckoning							
Odometry 0.5m/s	0.3706						
Odometry 0.25m/s	0.3333						
Odometry 0.125m/s	0.2994						
Odom-gyro 0.5m/s	0.2523						
Odom-gyro 0.25m/s	0.2505						
Odom-gyro 0.125m/s	0.2510						

Table 5.1: Measured deviation in meters from reflector navigation at the end of the test runs in case one.

SLAM Simulation	1	2	3	4	5	Mean	Variance
GMapping 0.5m/s	0.1065	0.0555	0.0764	0.0814	0.0644	0.0769	0.0004
GMapping 0.25m/s	0.0213	0.1364	0.1443	0.0703	0.1066	0.0958	0.0026
GMapping 0.125m/s	-0.0494	-0.0454	-0.0714	-0.0634	-0.1045	-0.0668	0.0006
Hector 0.5m/s	0.0016	0.0028	0.0036	0.0026	0.0039	0.0029	0.0000
Hector 0.25m/s	-0.0091	-0.0142	-0.0342	-0.0363	-0.0391	-0.0266	0.0002
Hector 0.125m/s	-0.0352	-0.1799	-0.1798	-0.1989	-0.1798	-0.1547	0.0045
Dead Reckoning							
Odometry 0.5m/s	-0.0218						
Odometry 0.25m/s	-0.0212						
Odometry 0.125m/s	-0.0156						
Odom-gyro 0.5m/s	0.0108						
Odom-gyro 0.25m/s	-0.0235						
Odom-gyro 0.125m/s	0.0228						

Table 5.2: Measured angular deviation in radians from reflector navigation at the end of the test runs in case one.

For this test case the dead reckoning data shows some of the trends that were expected, slight increase in odometry quality as speed decreases. Dead reckoning with gyro seem to stay fairly constant in this case. Regarding SLAM, the fact that a decrease of speed doesn't yield improved results is somewhat unexpected, especially HectorSLAM yields very poor results for four runs with the lowest speed. The possible reasons for this is discussed in detail in the next chapter. GMapping exhibit some "jumpy" behaviour in all three speeds and the reason for this will be discussed in the next chapter.

5.2 Test 2 - Wall visible on one side with objects

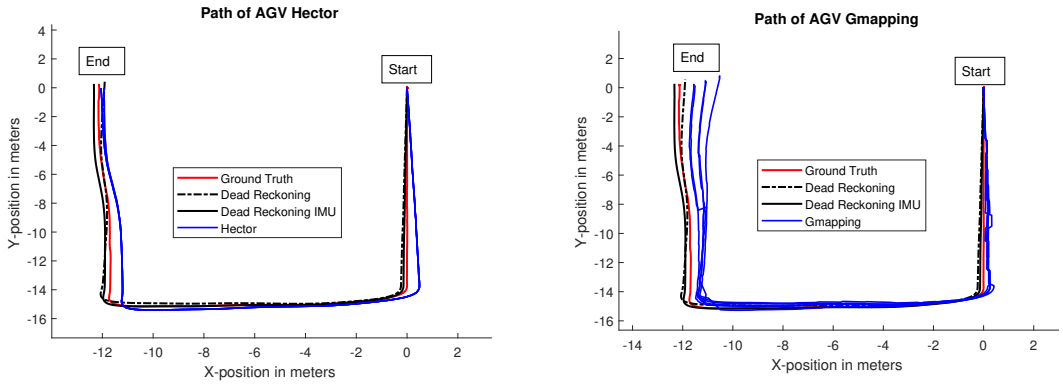

Figure 5.4: Figures showing trajectories from the fast speed run of the wall with pillars course. All five runs are shown in the figures.

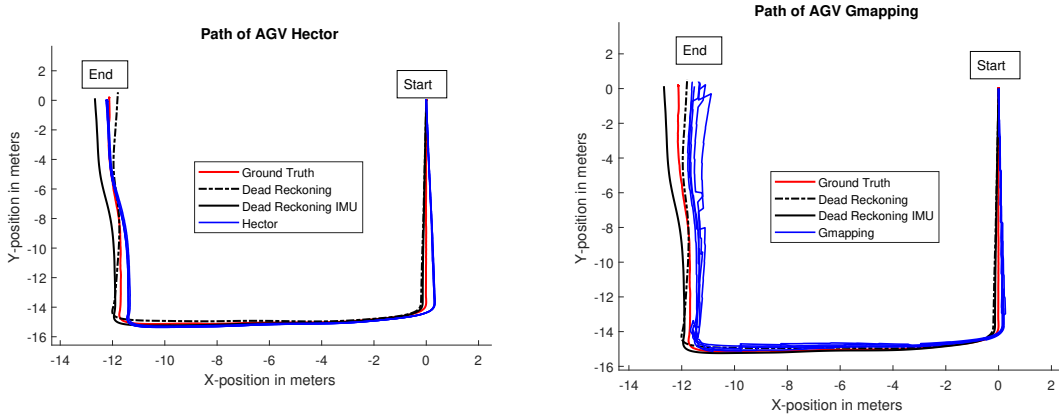

Figure 5.5: Figures showing trajectories from the medium speed run of the wall with pillars course. All five runs are shown in the figures.

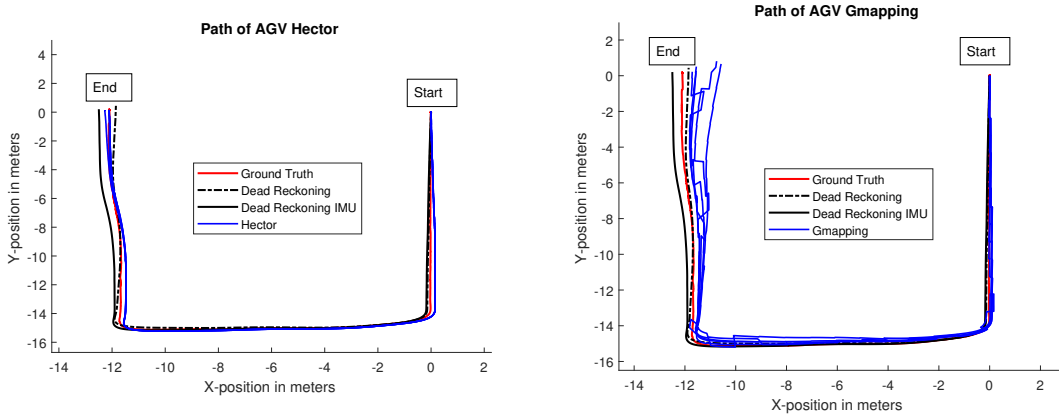

Figure 5.6: Figures showing trajectories from the slow speed run of the wall with pillars course. All five runs are shown in the figures.

SLAM Simulation	1	2	3	4	5	Mean $[m]$	Variance $[m]$
GMapping 0.5m/s	1.6942	1.0432	1.1076	0.5666	0.6240	1.0071	0.2061
GMapping 0.25m/s	0.8200	0.7967	0.8201	0.6084	0.5475	0.7185	0.0170
GMapping 0.125m/s	1.6016	0.5492	0.3921	0.6488	1.4967	0.9377	0.3213
Hector 0.5m/s	0.2560	0.2560	0.3242	0.3013	0.3268	0.2929	0.0012
Hector 0.25m/s	0.1547	0.2079	0.2079	0.2309	0.2414	0.2086	0.0011
Hector 0.125m/s	0.0957	0.1881	0.0727	0.1077	0.0601	0.1049	0.0025
Dead Reckoning							
Odometry 0.5m/s	0.4480						
Odometry 0.25m/s	0.2197						
Odometry 0.125m/s	0.3719						
Odom-gyro 0.5m/s	0.2197						
Odom-gyro 0.25m/s	0.2505						
Odom-gyro 0.125m/s	0.3963						

Table 5.3: Measured deviation in meters from reflector navigation at the end of the test runs in case two.

SLAM Simulation	1	2	3	4	5	Mean	Variance
GMapping 0.5m/s	-0.129	-0.0781	-0.0929	-0.0560	-0.0699	-0.0854	0.0008
GMapping 0.25m/s	-0.0780	-0.0510	-0.0771	-0.0630	-0.0520	-0.0642	0.0002
GMapping 0.125m/s	-0.1197	-0.0388	-0.0178	-0.0657	-0.0788	-0.0642	0.0015
Hector 0.5m/s	0.0131	0.0131	0.0049	0.0147	0.0053	0.0102	0.0000
Hector 0.25m/s	0.0038	0.0022	0.0022	0.0049	0.0047	0.0035	0.0000
Hector 0.125m/s	0.0087	-0.0034	0.0099	0.0087	0.0047	0.0057	0.0000
Dead Reckoning							
Odometry 0.5m/s	-0.0302						
Odometry 0.25m/s	-0.0340						
Odometry 0.125m/s	-0.0220						
Odom-gyro 0.5m/s	0.0107						
Odom-gyro 0.25m/s	0.0427						
Odom-gyro 0.125m/s	0.0154						

Table 5.4: Measured angular deviation in radians from reflector navigation at the end of the test runs in case two.

In this test, where boxes replicating pillars had been placed along the U-shaped trajectory, GMapping showed the biggest deviation from ground truth across all speeds while HectorSLAM showed the lowest. In table 5.3 the results of the test case of wall following with pillars on the other side is displayed. Dead reckoning shows the same trends as in the previous case in table5.1 as expected. The SLAM algorithms have performed better than in the previous case however the biggest difference is that the slowest case for HectorSLAM now produce a good result. GMapping is still a bit "jumpy" however.

5.3 Test 3 - Corridor with pillars on both sides

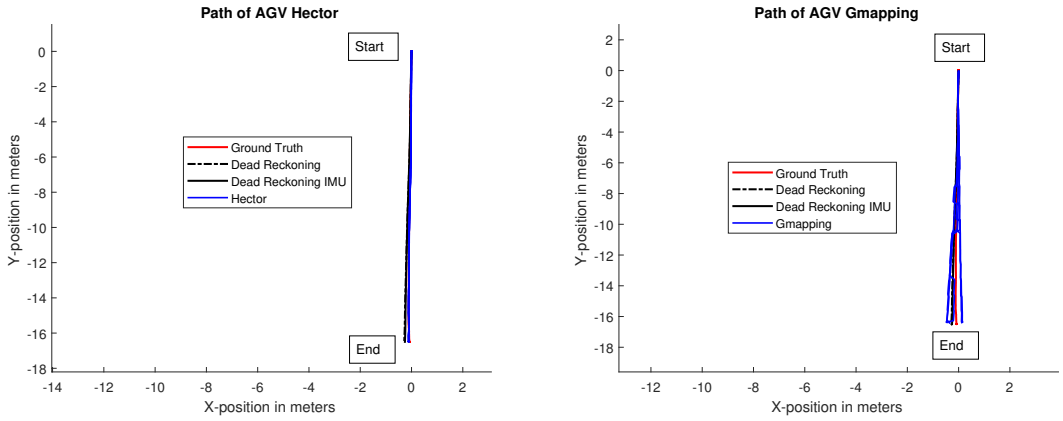

Figure 5.7: Figures showing trajectories from the fast speed run of the corridor course. All five runs are shown in the figures.

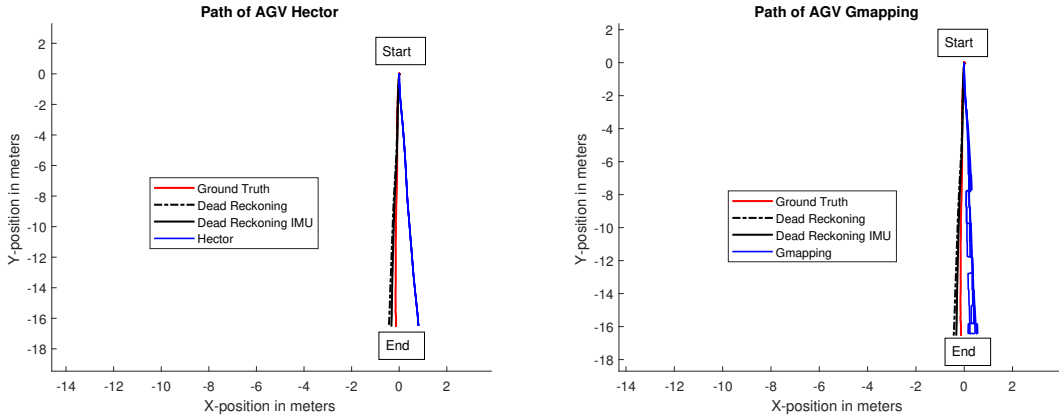

Figure 5.8: Figures showing trajectories from the medium speed run of the corridor course. All five runs are shown in the figures.

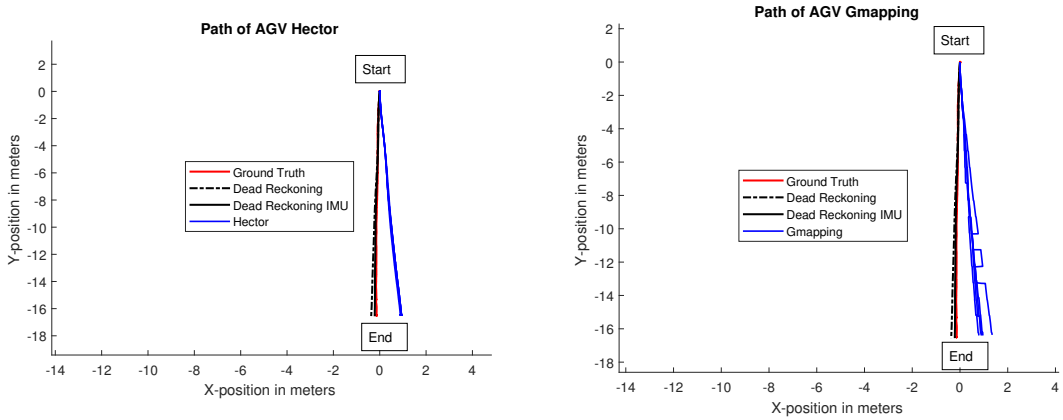

Figure 5.9: Figures showing trajectories from the slow speed run of the corridor course. All five runs are shown in the figures.

SLAM Simulation	1	2	3	4	5	Mean $[m]$	Variance $[m]$
GMapping 0.5m/s	0.2837	0.2666	0.3484	0.2351	0.3945	0.3056	0.0042
GMapping 0.25m/s	0.5983	0.6492	0.5584	0.3877	0.7132	0.5813	0.0151
GMapping 0.125m/s	0.9354	1.1027	1.0480	1.0393	1.4911	1.1233	0.0459
Hector 0.5m/s	0.0389	0.0392	0.0358	0.0344	0.0392	0.0375	5.1506e-06
Hector 0.25m/s	0.9342	0.9342	0.9342	0.9342	0.9342	0.9342	1.3534e-09
Hector 0.125m/s	1.0294	1.0792	1.0792	1.0812	1.0792	1.0696	5.0661e-04
Dead Reckoning							
Odometry 0.5m/s	0.2198						
Odometry 0.25m/s	0.3129						
Odometry 0.125m/s	0.2382						
Odom-gyro 0.5m/s	0.1915						
Odom-gyro 0.25m/s	0.2105						
Odom-gyro 0.125m/s	0.1029						

Table 5.5: Measured deviation in meters from reflector navigation at the end of the test runs in case three.

SLAM Simulation	1	2	3	4	5	Mean [m]	Variance
GMapping 0.5m/s	-0.0169	0.0291	0.0271	-0.0049	0.0421	0.0153	0.0006
GMapping 0.25m/s	-0.0041	-0.0050	-0.0010	0.0020	-0.0120	-0.0042	0.0000
GMapping 0.125m/s	0.0029	0.0019	-0.0061	-0.0091	0.0269	0.0033	0.0002
Hector 0.5m/s	0.0031	0.0031	0.0031	0.0031	0.0031	0.0031	0.0000
Hector 0.25m/s	0.0150	0.0160	0.0160	0.0160	0.0160	0.0158	0.0000
Hector 0.125m/s	0.0139	0.0060	0.0059	0.0059	0.0059	0.0075	0.0000
Dead Reckoning							
Odometry 0.5m/s	-0.0097						
Odometry 0.25m/s	-0.0156						
Odometry 0.125m/s	-0.0132						
Odom-gyro 0.5m/s	-0.0028						
Odom-gyro 0.25m/s	0.0004						
Odom-gyro 0.125m/s	0.0074						

Table 5.6: Measured angular deviation in radians from reflector navigation at the end of the test runs in case three.

Tables 5.5-5.6 shows the results of test case 3 the dead reckoning performs very well as expected while SLAM produces relatively poor results especially for low speeds. It is however to be expected that SLAM performs subpar in this case since it is the one with the least amount of information from the scan data. It is also the best case for dead reckoning since there are no turns.

5.4 Test 4 - Zig-zag

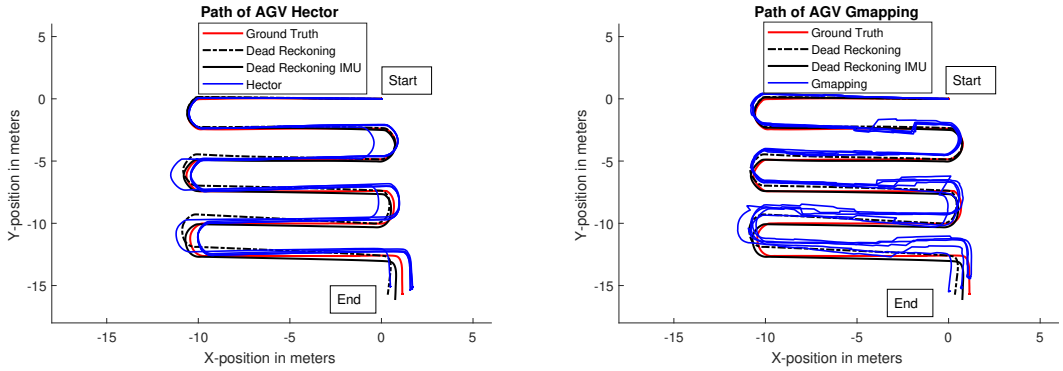

Figure 5.10: Figures showing trajectories from the fast speed run of the zig-zag course. All five runs are shown in the figures.

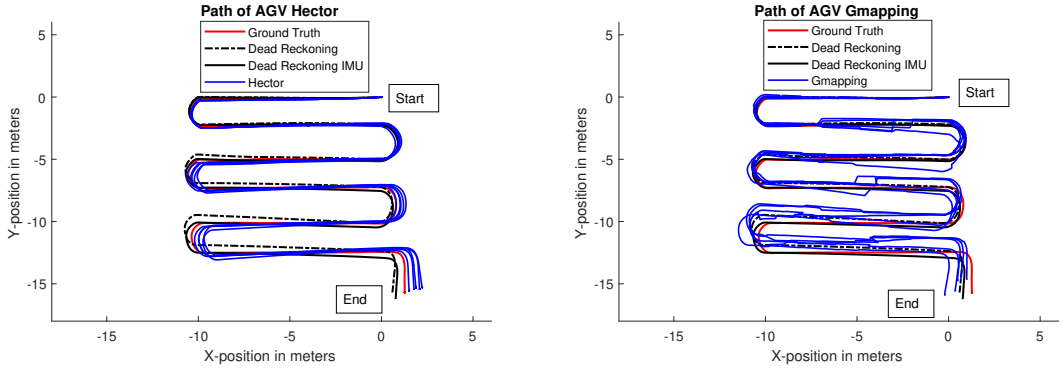

Figure 5.11: Figures showing trajectories from the medium speed run of the zig-zag course. All five runs are shown in the figures.

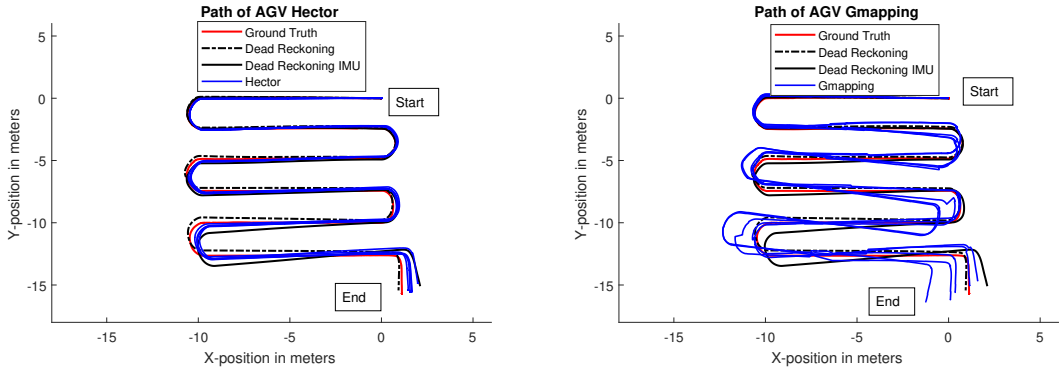

Figure 5.12: Figures showing trajectories from the slow speed run of the zig-zag course. All five runs are shown in the figures.

SLAM simulation	1	2	3	4	5	Mean $[m]$	Variance $[m]$
GMapping 0.5m/s	1.2001	1.3057	1.0546	1.333	0.6573	1.1101	0.0760
GMapping 0.25m/s	1.1868	1.2353	0.9022	1.1369	1.454	1.183	0.0393
GMapping 0.125m/s	0.7599	1.1188	2.4455	1.1474	0.6773	1.2298	0.5058
Hector 0.5m/s	0.5397	0.7848	0.6694	0.5399	0.8774	0.6822	0.0223
Hector 0.25m/s	0.8545	0.2819	0.7324	0.5842	1.0366	0.6979	0.0816
Hector 0.125m/s	0.5468	0.4538	0.5828	1.1865	0.4635	0.6467	0.0941
Dead Reckoning							
Odometry 0.5m/s	0.8101						
Odometry 0.25m/s	0.6683						
Odometry 0.125m/s	0.3037						
Odom-gyro 0.5m/s	0.6324						
Odom-gyro 0.25m/s	0.7242						
Odom-gyro 0.125m/s	1.1469						

Table 5.7: Measured deviation in meters from reflector navigation at the end of the test runs in case four.

SLAM Simulation	1	2	3	4	5	Mean	Variance
GMapping 0.5m/s	0.0046	0.0466	-0.0024	0.0346	0.0086	0.0184	0.0004
GMapping 0.25m/s	-0.0626	-0.0736	-0.1006	-0.0206	-0.1056	-0.0726	0.0012
GMapping 0.125m/s	-0.0197	-0.0216	-0.1416	0.1263	0.0463	-0.0021	0.0096
Hector 0.5m/s	0.0238	0.0554	0.0484	0.0234	0.0524	0.0406	0.0003
Hector 0.25m/s	0.0196	0.0138	0.0494	0.0204	0.0744	0.0355	0.0007
Hector 0.125m/s	0.0377	0.0415	0.0405	0.0985	0.0216	0.0479	0.0009
Dead Reckoning							
Odometry 0.5m/s	-0.0729						
Odometry 0.25m/s	-0.0586						
Odometry 0.125m/s	-0.0132						
Odom-gyro 0.5m/s	-0.0163						
Odom-gyro 0.25m/s	-0.0352						
Odom-gyro 0.125m/s	0.1764						

Table 5.8: Measured angular deviation in radians from reflector navigation at the end of the test runs in case four.

Table 5.7 shows the results for the final test case. This is the case where dead reckoning was expected to perform the worst and this can be seen clearly, especially the dead reckoning with gyroscope. Unlike the previous tests SLAM results do not vary much between different speeds and are instead fairly constant. GMapping still exhibit the "jumpy" behaviour as in the earlier test cases.

5.5 Summary

In this chapter the test results have been presented, for the first and second test case there is a difference in the quality of the resulting trajectories probably because of the added objects in the second case. For the third case no clear conclusions can be drawn at a glance. In the fourth case speed is shown to have a big impact on the result of the trajectories. However some overall trends can be observed with the most interesting one being that the lowest speed setting generally results in the worst results. Apart from that the results were somewhat expected in that a target rich environment generally results in a better result from the SLAM algorithm, while the highest speed setting also usually results in degraded results. Results from odometry were relatively constant for all speed settings.
In the next chapter these results and what they mean will be discussed in detail.

6
Discussion

In this chapter, discussion of the results from the different test cases will be presented. The aim of the thesis was to investigate the accuracy and repeatability of SLAM generated trajectories in an industrial environment. The SLAM trajectories were compared to ground truth and dead reckoning with and without gyroscope data. Discussion about the results and how to interpret them for each of the four test cases can be found below. Discussion about the choice of method and possible weaknesses in it along with future development is also included in this chapter.

6.1 Results

This section will discuss the results from the four test cases in different subsections and then summarize the trends observed across all the cases.

6.1.1 Test 1

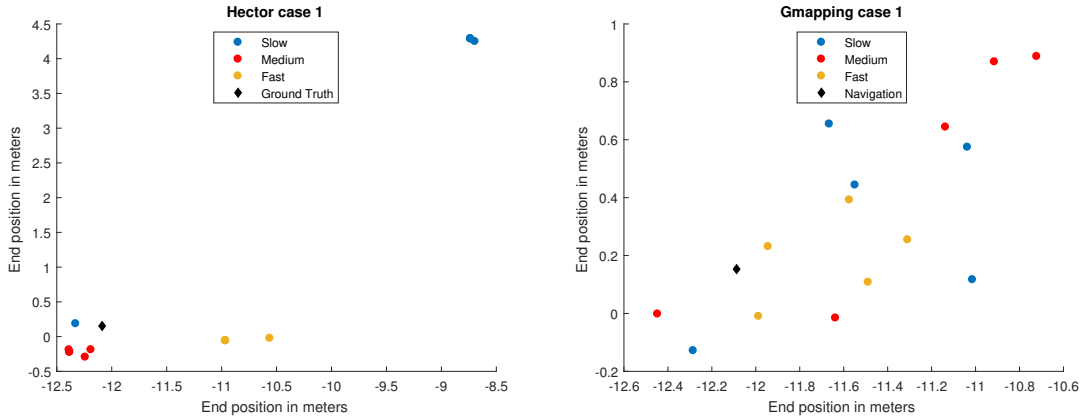

Figure 6.1: Figures showing the estimated end coordinates for both HectorSLAM and Gmapping in case 1

As can be seen from the first test, HectorSLAM had the worst results for the fastest and slowest speeds, while still having satisfactory results with medium speed. While the error produced from the fastest speed seem to originate from the lower wall being flat and the relatively low scan frequency along with no odometry information using HectorSLAM. The slower speed, however, show a jump at the bottom right

of the trajectory for four out of five test simulations. Although the cause of this is uncertain, the current guess of the authors is that the wall where the jump happens is too flat and the AGV believes it is standing still. This would explain the sudden jump in location when the bottom wall "suddenly appears". Thus low speeds in combination with low information from the scans create a version of the "long corridor" scenario where no movement can be calculated from the scans.
Regarding GMapping, the results show that it seemed to keep the correct trajectory due to using odometry alongside laser scans, but it exhibited some "jumpy" behaviour along the leftmost wall. The cause for this is a property of GMapping, as the algorithm works by choosing which of the concurrent maps has the highest probability of being true, resulting in a "jump" in position every time the filter updates which map is the most likely of being true. As the data logged from ROS is the *map* frame to the *base_link* frame, every map update give an update in position, which in turn give "jumpy" trajectories.
The odometry information from this test were as expected. As the speeds were relatively low and the floor conditions were good, the resulting risk of skidding on the surface was minimal. Due to this the results were expected to not be especially dependant on speed, which can be seen in the results. However, while the dead reckoning with only odometry was expected to be lowered, the expectations from odometry with a gyroscope was the reverse. As the accumulated error from gyroscope drift should get higher the longer the runtime of the test, seeing that the deviation is stable across all speeds was a surprise. However, the reason for this might be an increased accuracy of odometry counteract the inaccuracy from the gyroscope.

6.1.2 Test 2

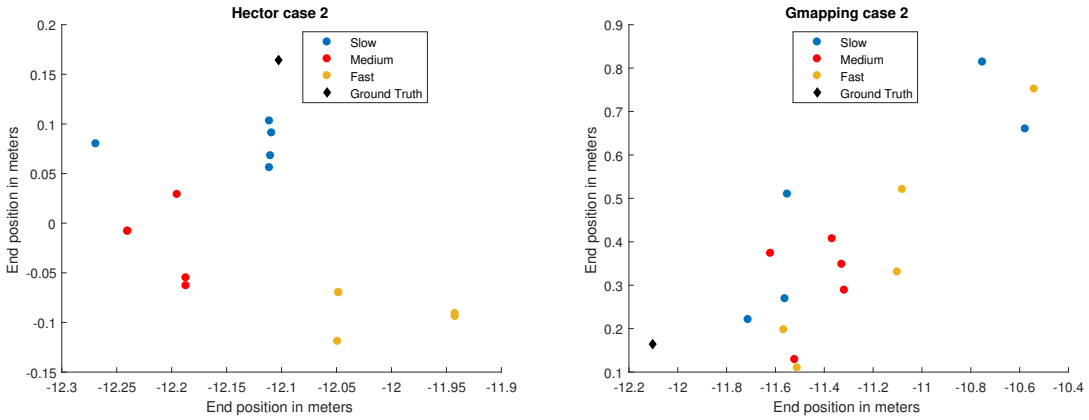

Figure 6.2: Figures showing the estimated end coordinates for both HectorSLAM and Gmapping in case 2

For the second test HectorSLAM showed some improvements across all speeds compared to the first test. For one, the horizontal trajectory is around the same length as that for the odometry for the faster speed and the deviation from ground truth

has been lowered significantly. The slowest speed show similar improvements. The sudden jump in the lower right corner is gone and the accuracy is greatly improved. As the only noticeable difference between this test and the first one is more information in form of pillars around the perimeter of the travelled trajectory, the likeliest cause of this improvement can be assumed to be the increased information from scan data.
In the case of GMapping, the biggest difference was the rightmost side of the trajectory as the amount of sudden jumps from test one has been decreased, and the jumps occurring seem to have had their distances decreased. As with HectorSLAM, it seems that the additional amount of information from scan data has been aiding the algorithm with decreasing the "jumpy" behaviour. As for the resulting deviation at the end of the run, the results seem fairly static, with the exception of the fastest speed, when compared to the first test.
As for odometry deviation, as this test case use the same trajectory and layout as the first test, no improvement from odometry data was expected and this seems to be what we see in the results.

6.1.3 Test 3

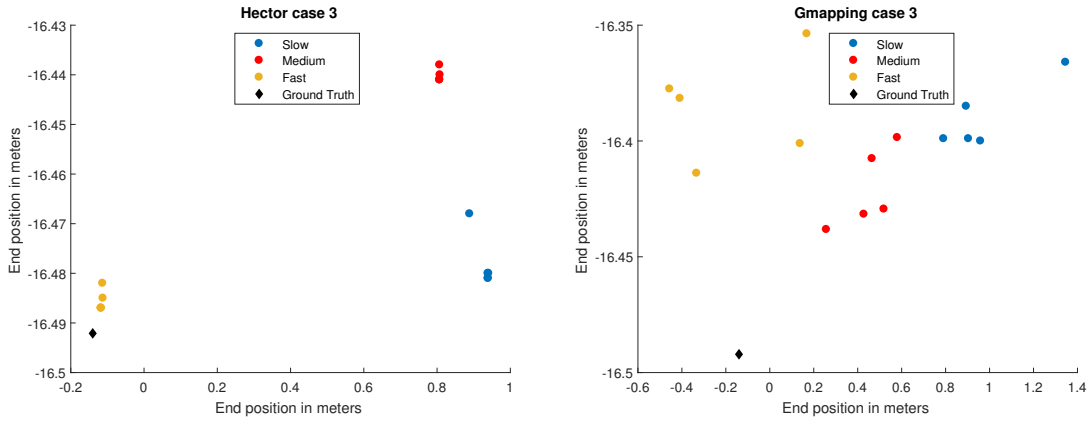

Figure 6.3: Figures showing the estimated end coordinates for both HectorSLAM and Gmapping in case 3

In this test, HectorSLAM seemed to perform exceptionally at the highest speed, but the performance worsened as speed went down. The reason for this is unknown, but it is believed that the cause can be attributed to the same problem HectorSLAM exhibited in test one. As this test is sparse in information, the AGV might believe it is standing still. This is especially true considering that HectorSLAM does not use odometry information at all. However, while the deviations increase, the trajectories seem to still be consistently wrong.
For GMapping, the same conclusions as from HectorSLAM can be done. The low amount of information in the scans results in bad trajectories across all speeds. While the fastest speeds seem to give decent results, this is most likely caused by

the fact that higher speeds equate to lower scan rate, giving the AGV time to move between scans and thus getting information that is different in relation to the scans.
As this was the shortest test in both time and distance, the odometry values were expected to be very good. Especially as the track contains no turns as it is a straight segment.

6.1.4 Test 4

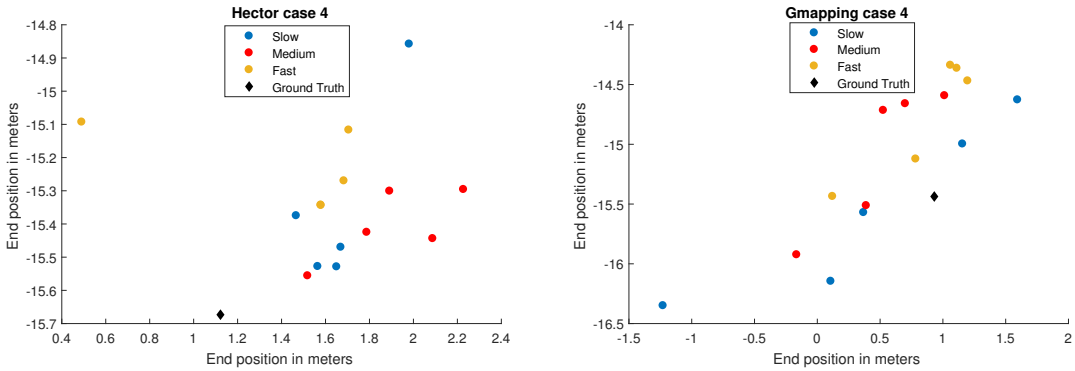

Figure 6.4: Figures showing the estimated end coordinates for both HectorSLAM and Gmapping in case 4

In this test, HectorSLAM performed consistently across all speeds. While it was expected that HectorSLAM would perform well using lower speeds, the accuracy at higher speeds came as little of a surprise as quick turns should cause HectorSLAM to start drifting due to it not using odometry. As this did not occur in the tests done, it seems that the turning speed might have been lower than expected. While the deviations seem to be the same regardless of speed, the trajectories differ some. In the fastest speed, the trajectories between the boxes seem to be shorter in at least one of the runs. However, this seems to not be consistent as most of the trajectories stop at around the same position.
GMapping, however, seems to have bigger problems on this test. The best result came from the fastest speed, getting worse for the lower ones. This was not entirely unexpected due to GMapping requiring larger loop closures than HectorSLAM to get good maps and thus positioning. As the tests were setup to avoid loop closures, the "jumpy" trajectories were somewhat expected as the map would be subpar.
As for odometry, this test was the longest in both time and distance. As such, the relatively bad odometry results were expected. As can be seen from the results, while odometry without a gyroscope got better with lower speed, the reverse is true for odometry with a gyroscope. This is due to an inherent property of the gyroscope that lets the drift reset when the AGV is stationary. As the test was setup to not let this happen, an expected increase in deviation occurred as the error due to gyroscope drift continued to accumulate.

6.1.5 Overall results

The tests display both expected and unexpected results, first of all the odometry generally shows similar results for all movement speeds, this is an expected result since it should not be very dependent on movement speed due to minimal risk of skidding on the surface due to good floor conditions and relatively low speeds. In the zig-zag case however there is a clear trend of increased odometry accuracy without a gyroscope but decreased accuracy with the gyroscope. This is most likely a result of the fact that this is by far the longest test both in terms of distance and time, by decreasing the speed the accuracy of the odometry is increased while the gyroscope accumulates more errors due to drift the longer the test run is. As the test cases where designed to not allow the gyroscope drift to reset in order to evaluate an erroneous result against SLAM. However, this drift seem to have a negligible effect on the accuracy of the odometry in every test case except the last one.
For the first two tests, the SLAM results are interesting as there is no clear trend that lower movement speeds produce better results. For GMapping this can be explained by the fact that it is configured to only update the filter every 0.5 meters moved in order to reduce computational load. For HectorSLAM no clear conclusion can be drawn, the current guess of the authors is that very low speeds combined with low range of the scanner results in not enough data being available to make good estimations. A possible scenario is that the wall where the AGV gets lost is too flat resulting in the robot thinking it isn't moving. For higher speeds this might not cause any issues since it is only for a limited amount of time, for lower speeds however the AGV gets disoriented. This is supported by the fact that this problem did not occur when we increased the available information per scan by adding the pillars in test two.
In the rest of the cases a clear pattern of that reduced movement speed results in a better result from the SLAM algorithm since more data is produced. This effect is especially clear for HectorSLAM since it only uses the scan data thus moving fast, especially rotating can cause problems. The fact that HectorSLAM outperformed GMapping even for higher velocities was an unexpected result. HectorSLAM was expected to outperform GMapping for lower speeds since it is not reliant on large loop closures in the same way that GMapping is, but the quick rotations for higher speeds were expected to cause bigger problems.

6.2 Choice of method

One downside with the tests done in this thesis is that the odometry for the AGV was tested once for each speed and scenario due to time constraints. A consequence of this is that, unlike HectorSLAM and GMapping, no statistical analysis of the odometry from the AGV was done. However, the values gotten from the tests done were deemed sufficient due to the AGV used having a very well tuned odometry.
As can be seen from some of the trajectories, the position of the AGV for GMapping exhibit "jumpy" behaviour. That is likely because the data logged from ROS was the transform between the *map* frame and the *base_link* frame, and every time GMapping changes which of the concurrent maps has the highest probability of being

true, the AGVs position in the map gets updated and thus skewing the trajectory by quickly switching position. This is, however, a side-effect of the mapping algorithm and is working in a way that is useful for mapping purposes but not when tracking the AGV trajectory. As these behaviours were unavoidable for most of the tests, it would imply that the performance of GMapping is subpar to HectorSLAM for trajectory analysis in these types of environments.
An interesting scenario that wasn't tested in this thesis would be if the AGV used was modified to have worse odometry to evaluate the SLAM methods in a worst-case scenario. That is, de-tuning the AGV odometry to be worse than that of a well tuned AGV, which was used in the tests. This is especially interesting considering that an ordinary AGV will most likely have worse tuned odometry than the small-scale AGV used during testing. As the odometry in the test scenarios has been well tuned, the odometry values gained should be seen as a best case-scenario and would very likely be worse when done on a real AGV. This due to the fact that odometry is based on, for example, wheel diameter which will change due to wear and tear or when the AGV is loaded with a heavy load. However, because HectorSLAM only use the scan data and not odometry, the resulting deviations from HectorSLAM can be assumed to be unchanged when scaling up to a full-scale AGV.
As the performance of HectorSLAM at higher speeds was unexpected due to quick turns being a known weakness of the algorithms, it is possible that the turning speeds done in these tests were lower than was expected before performing them. As such, it would be interesting to re-do the tests with higher turning speeds to get results that is more based in reality as it is unlikely that a real AGV will have turning speeds that HectorSLAM will be able to handle. Thus, it would be interesting to do an analysis to find a breakpoint of how fast the vehicle can turn versus the scan and update frequency needed by the algorithm would be important for practical applications.
It would also have been interesting to have some data collection from a real-life situation (a full-scale AGV driving in a semi-structured area) and do the same types of evaluation methods as for the Charmvagn. This would be interesting considering the Charmvagn has very well tuned odometry, whereas an AGV working in an industrial environment would most likely be performing worse due to more wheel slip and different wheel setup. An analysis between a best case in form of the well tuned Charmvagn and a worst case in form of an industrial AGV would have been interesting to do. However, this was not done due to both time constraints and no access to relevant test equipment.

6.3 Future research and development

The overall impression of the authors is that there is much potential in SLAM based on the results from this study. The results show that relatively accurate trajectories are produced even though the algorithms tested in a way they are not intended. However, as these tests only compared the SLAM algorithms against dead reckoning calculated with very good odometry, the next step would be to try these tests on a full-scale AGV where the odometry information is worse than that of the one used in this thesis. As the tests done against good odometry showed some

promise of being implemented in real systems, testing against dead reckoning with bad odometry would give even better information regarding whether or not SLAM is useful for reinforcing dead reckoning in AGV systems. If these new tests show promising results, the next natural step would be integration of a SLAM algorithm to be used in the vehicle controller instead of separately on a computer. Then integrating the SLAM localization into the "standard" navigation system and make the AGV switch between navigating a known area with reflector navigation, switch to SLAM navigation when entering an unknown area and switch back to reflector navigation when re-entering a known area. For practical purposes HectorSLAM especially is of interest since it is designed to a be a more light weight solution compared to GMapping and can thus be run on modest hardware.
Even though Google Cartographer was chosen not to be tested in this thesis due to time constraints and problems with getting it to work properly, earlier work show that the mapping capabilities of this algorithm outperform the other two [2] although at a cost of higher CPU requirements. Thus it would be interesting to see if the trajectories created would be closer to ground truth when compared to Hector SLAM and GMapping.

One important step for further development of SLAM-inspired dead reckoning would be to conduct further testing of where the limits of when it produces good results lie. While it is the opinion of the authors that this technology holds much promise the test results presented in this thesis could be expanded upon. As stated previously the fact that HectorSLAM performed better than expected at the highest speed tested was unexpected thus the first set of new tests to be conducted would be to test the limits of how fast the vehicle can move while still producing good results.
Another question that the results of this paper poses is where the threshold for how much *content* the environment visible to the laser must contain in order for the SLAM algorithm to produce a good result. The obvious way to do this is to simply conduct a test without any objects and then adding objects to the environment over a series of tests. The next step would be to develop a metric for how object rich the environment has to be in order for SLAM-inspired dead reckoning to work.

6.4 Summary

In this chapter the test results and how to interpret them in is discussed in detail. The first two tests largely behave as expected with the added environment in test case two resulting better trajectories overall. The jumpy behaviour seen in the trajectories generated by GMapping are attributed to the particle filter nature of that SLAM algorithm. The first test can be interpreted as not containing enough environment to create good trajectories, especially for HectorSLAM which does not use odometry. The third test produces relatively poor results for both of the algorithms, this is most likely due to a lack of environment or possibly due to the fact that each piece of the environment is an identical box meaning that the scan matcher has a harder time localizing. Test four produced good results from both GMapping and HectorSLAM at medium speed but overall HectorSLAM outperformed GMapping

in this test.
The overall result are in short that HectorSLAM performed much better than expected especially at the faster speeds.
The methods used in testing are also evaluated the main deficiency of this thesis is that due to time constraints the amount of test data was relatively small. The limits of how well the SLAM algorithms work at a high movement speed is also not evaluated properly since in initial testing HectorSLAM did not perform well at all for higher speed but in reality performed much better than expected
Finally some further research and development is suggested, this is a big field and there is a first of all comparing with odometry from a less well tuned vehicle is desirable since the one used in this work is *too good* making any comparison unfair to the SLAM algorithm. Investigation into seamless switching between SLAM based navigation and other types of navigation would also need to be investigated for this work to be of any real use.
In the next chapter conclusions about the results of this thesis are drawn.

7
Conclusion

In this thesis, we have explored whether SLAM algorithms can be used as a means to reinforce dead reckoning in areas where traditional navigation methods cannot be used. The two SLAM algorithms used for evaluation was chosen to be GMapping and HectorSLAM, both of which are available as ROS packages. The tests done show that SLAM algorithms have some potential as an assisting tool in dead reckoning even though the algorithms were used in a way they aren't designed to be used. Even though the SLAM algorithms exhibit some problems for test cases where information about the surrounding area is sparse, this is an inherent problem with SLAM algorithms as less information lead to worse maps and localization. Although HectorSLAM never exhibited the expected problems when doing quick turns and thus might have behaved better in testing than in reality, the overall impression of the test results is that HectorSLAM seems to outperform GMapping when building trajectories, due to GMapping exhibiting "jumpy" behaviour when updating the map. HectorSLAM also showed itself to be a powerful tool even though it is computationally less demanding than GMapping. These properties make HectorSLAM a good candidate for further testing and implementation into an AGV for real-life use. However, it should be noted that the usefulness of such an implementation is dependant on the areas where it is utilized as a lower amount of information will lead to worse performance as HectorSLAM only use information from the scan data to do SLAM, this could however be a good attribute since it would allow accurate trajectories in circumstances where odometry cannot be trusted.

7.1 Research questions

The research questions aimed to be answered in this thesis was set to be the following:

- How accurate and repeatable are trajectories produced by SLAM using a single plane LIDAR both with respect to deviation in distance and angle.

The short answer to this questions is that the trajectories produced can both be accurate and repeatable, however this will only be true under certain circumstances. The tests show that for trajectories the results are very impressive for environments that are object rich, and if the speeds are kept *moderate*. In this work the tested SLAM algorithms have been benchmarked against dead reckoning since finding ways to improve this is highly desirable, especially if it can be done with already existing sensors. This thesis shows that for environments that are object rich HectorSLAM will perform on par or better than well tuned dead reckoning. GMapping generally

perform worse but is still producing good results all things considered.
For the test cases that produce good SLAM trajectories the variance is generally very low. For some of the test cases the variance of the HectorSLAM trajectory is almost zero, this is in the context of AGVs a very important result since demands for repeatability and being able to run without operator interaction for extended periods of time is very important.
The importance of good environments is highlighted by the difference between test case one and two, the only difference between the tests is that more objects are added to the environment. However there is not enough data to do a proper analysis of how much environment is needed, this problem is also exacerbated by the fact that the LIDAR data is filtered to remove all scan data further away than four meters.

Bibliography

[1] Günter Ullrich. *Automated Guided Vehicle Systems. [electronic resource] : A Primer with Practical Applications.* Berlin, Heidelberg : Springer Berlin Heidelberg : Imprint: Springer, 2015., 2015. ISBN: 9783662448144. URL: http://proxy.lib.chalmers.se/login?url=http://search.ebscohost.com.proxy.lib.chalmers.se/login.aspx?direct=true&db=cat06296a&AN=clc.b2051613&lang=sv&site=eds-live&scope=site.

[2] Albin Pålsson and Markus Smedberg. "Investigating Simultaneous Localization and Mapping for AGV systems". 72. MA thesis. Göteborg: Institutionen för data- och informationsteknik (Chalmers), Chalmers tekniska högskola, 2017.

[3] H. Davey N.S. Godil. "Simple but novel test method for quantitatively comparing robot mapping algorithms using SLAM and dead reckoning". In: *Proc. SPIE 8741, Unmanned Systems Technology XV, 874112*. 2013.

[4] Christoph Reinke Patric Beinschob. "Graph SLAM based mapping for AGV localization in large-scale warehouses". In: *2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*. 2015.

[5] Anton Filatov et al. "2D SLAM quality evaluation methods". In: *Open Innovations Association (FRUCT), 2017 21st Conference of*. IEEE. 2017, pp. 120–126.

[6] W. Burgard et al. "A comparison of SLAM algorithms based on a graph of relations". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, pp. 2089–2095. DOI: 10.1109/IROS.2009.5354691.

[7] Z. Kurt-Yavuz and S. Yavuz. "A comparison of EKF, UKF, FastSLAM2.0, and UKF-based FastSLAM algorithms". In: *2012 IEEE 16th International Conference on Intelligent Engineering Systems (INES)*. June 2012, pp. 37–43. DOI: 10.1109/INES.2012.6249866.

[8] Siavash Hosseinyalamdary Yashar Balazadegan Sarvrood and Yang Gao. "Visual-LiDAR Odometry Aided by Reduced IMU". In: *International Journal of Geo-Information* 5.1 (2015), p. 24.

[9] Daobin Wang et al. "LiDAR Scan matching EKF-SLAM using the differential model of vehicle motion". In: *2013 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. June 2013, pp. 908–912.

[10] Sungshin Kim Hyunhak Cho Kyeong Kim. "Indoor SLAM application using geometric and ICP matching methods based on line features". In: *Robotics and Autonomous Systems* 100.1 (2018). DOI: https://doi.org/10.1016/j.robot.2017.11.011. URL: https://www.sciencedirect.com/science/article/pii/S0921889017301367.

[11] D. Herrero-Pérez H. Martínez-Barberá. "Autonomous navigation of an automated guided vehicle in industrial environments". In: *Robotics and Computer-Integrated Manufacturing* 26.4 (2010). DOI: https://doi.org/10.1016/j.rcim.2009.10.003. URL: https://www.sciencedirect.com/science/article/pii/S0736584509000994.

[12] Johann Borenstein, HR Everett, Liqiang Feng, et al. "Where am I? Sensors and methods for mobile robot positioning". In: *University of Michigan* 119.120 (1996), p. 27.

[13] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. "Introduction to Autonomous Mobile Robots, Second Edition". In: *Intelligent robotics and autonomous agents*. 2011.

[14] J. -. Gutmann and C. Schlegel. "AMOS: comparison of scan matching approaches for self-localization in indoor environments". In: *Proceedings of the First Euromicro Workshop on Advanced Mobile Robots (EUROBOT '96)*. Oct. 1996, pp. 61–67. DOI: 10.1109/EURBOT.1996.551882.

[15] Majd Alshawa. "lCL: Iterative closest line A novel point cloud registration algorithm based on linear features". In: *Ekscentar* 10 (2007), pp. 53–59.

[16] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (June 2006), pp. 99–110. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022.

[17] C. Cadena et al. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32.6 (Dec. 2016), pp. 1309–1332. ISSN: 1552-3098. DOI: 10.1109/TRO.2016.2624754.

[18] Joan Sola. *Simulataneous localization and mapping with the extended kalman filter*. Jan. 30, 2019. URL: http://www.iri.upc.edu/people/jsola/JoanSola/objectes/curs_SLAM/SLAM2D/SLAM%20course.pdf.

[19] Wolfram Burgard Giorgio Grisetti Cyrill Stachniss. "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters". In: *IEEE Transactions on Robotics* 23.1 (2007). ISSN: 1552-3098. DOI: 10.1109/TRO.2006.889486. URL: https://ieeexplore.ieee.org/document/4084563/.

[20] Kevin P Murphy. "Bayesian map learning in dynamic environments". In: *Advances in Neural Information Processing Systems*. 2000, pp. 1015–1021.

[21] Michael Montemerlo et al. "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem". In: *In Proceedings of the AAAI National Conference on Artificial Intelligence*. AAAI, 2002, pp. 593–598.

[22] Montemerlo. Michael and Thrun. Sebastian. "FastSLAM 2.0". In: *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics* (2007), pp. 63–90.

[23] Hans P Moravec. "Sensor fusion in certainty grids for mobile robots". In: *AI magazine* 9.2 (1988), p. 61.

[24] Arnaud Doucet et al. "Rao-Blackwellised particle filtering for dynamic Bayesian networks". In: *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2000, pp. 176–183.

[25] Arnaud Doucet, Nando De Freitas, and Neil Gordon. “An introduction to sequential Monte Carlo methods”. In: *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 3–14.

[26] Lu Feng and Milios Evangelos. “Globally consistent range scan alignment for environment mapping”. In: *Autonomous robots* 4.4 (1997), pp. 333–349.

[27] Stefan Kohlbrecher et al. “Hector open source modules for autonomous mapping and navigation with rescue robots”. In: *Robot Soccer World Cup*. Springer. 2013, pp. 624–631.

[28] S. Kohlbrecher et al. “A Flexible and Scalable SLAM System with Full 3D Motion Estimation”. In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. Nov. 2011.

[29] M Quigley et al. “ROS: An open-source Robot Operating System”. In: *ICRA Workshop on Open Source Software* 3 (Jan. 2009), pp. 1–6.