A Platform for Indoor Localisation, Mapping, and Data Collection using an Autonomous Vehicle

Andreas Teodor Coroiu
dat11aco@student.lu.se

Oscar Hinton
dat11ohi@student.lu.se

14th June 2017

Master's thesis work carried out at Combain Mobile AB.

Supervisors: Rikard Windh, rikard@combain.com
Björn Lindquist, bjorn@combain.com
Kalle Åström, kalle@maths.lth.se
David Gillsjö, david.gillsjo@math.lth.se

Examiner: Magnus Oskarsson, magnuso@maths.lth.se

Abstract

Everyone who has worked with research knows how rewarding experimenting and developing new algorithms can be. However in some cases, the hard part is not the invention of these algorithms, but their evaluation. To try and make that evaluation easier, this thesis focuses on the collection of data that can be used as positional ground truths using an autonomous measurement platform. This should assist Combain Mobile AB in the evaluation and improvement of their Wi-Fi based indoor positioning service.

How and which parts of the open-source community's work in the Robot Operating System (ROS) project to utilise is not obvious. This thesis therefore sets out to build a Minimum Viable Product (MVP) which is capable of supporting two different use cases: measure and explore inside an unknown environment, and measure inside a known environment given a map. This effectively leaves Combain with a viable product, and indirectly helps the community by aiding it in comparing and recommending the best tools and software libraries for the task.

The result of this thesis ends up recommending the following for measuring inside an unknown environment: the Simultaneous Localisation And Mapping (SLAM) algorithm Google Cartographer for navigation, and the exploration algorithm Hector Exploration for planning the exploration. To measure inside a known environment the following is recommended: the Adaptive Monte Carlo Localisation (AMCL) positioning algorithm and the Spanning Tree Covering algorithm.

Keywords: Area Coverage, Autonomous Vehicles, Exploration, Robotic Operating System (ROS), SLAM, Wi-Fi

Acknowledgements

We would first like to extend our sincerest gratitude to our academic supervisors: Prof. Kalle Åström and Ph.D. student David Gillsjö at Lund University, Faculty of Engineering. Their extensive knowledge and experience has been an immense help in ensuring the success of this thesis. We are grateful for the multiple in-depth discussions which have inspired multiple ideas, and we would like to express our thanks for the encouragement and critique of this thesis.

We would also like to extend our gratitude to everyone at Combain Mobile AB for allowing us to conduct our master's thesis at their company and for providing a workplace. Special thanks to Björn Lindquist and Rikard Windh for supervising this thesis and providing many good suggestions. Huge thanks to Anders Mannesson for helping us understand their existing indoor solution, his many in-depth reviews, and for listening to us ramble on without actually having any official role in this thesis.

This thesis builds upon the excellent work of the Robot Operating System community, which have our heartfelt thanks for making this thesis possible.

Last, we would like to thank our friends and families, whom have supported us during both this thesis, and the duration of our studies. We could not have done it without your support!

Contents

Glossary

Access point An access point is a piece of hardware that allow devices (e.g. smartphones) to wirelessly connect to a network. 13, 16, 57–59, 67

API In computer programming, an Application Programming Interface (API) is a set of subroutine definitions, protocols, and tools for building application software. In general terms, it is a set of clearly defined methods of communication between various software components [1]. 9, 11, 59

Ceres solver Ceres Solver [2] is an open source C++ library for modeling and solving large, complicated optimization problems. It can be used to solve Non-linear Least Squares problems with bounds constraints and general unconstrained optimization problems. 30

IMU An Inertial Measurement Unit (IMU) is an electronic device that can measure and report a body's specific force, angular rate, and sometimes the magnetic field surrounding the body, using any combination of accelerometers, gyroscopes and magnetometers [3]. 9, 11, 18

Loop closure Loop closure is the problem of recognizing a previously-visited location and updates the beliefs accordingly. 20, 26–28, 30, 31, 34, 36, 69

Odometry Odometry is an estimation of the robots movements over time using sensors and motor encoders. However since motor encoders are not completely accurate due to slippage, the odometry tends to be quite unpredictable in the long run. Kobuki improves the angular accuracy of the odometry by fusing it with an Inertial Measurement Unit.. 27, 30, 36

Plug-in A plug-in is an interchangeable component that can be used to extend the existing functionality of software. 14, 15, 20, 21, 45, 61

SLAM Simultaneous Localisation And Mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it [4]. 1, 10, 11, 13, 21, 27

Trajectory A trajectory is the path which the robot has travelled, expressed as a function of time. 21, 27, 35, 36, 59

VPN A Virtual Private Network (VPN) extends a private network across a public network, and enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network [5]. 10, 11, 19, 25

WGS84 World Geodetic System 1984 is a reference coordinate system for the Earth. One of more well known systems which uses WGS84 is the Global Positioning System (GPS). 10, 11, 21

Acronyms

AMCL Adaptive Monte Carlo Localisation. 1, 43, 44

API Application Programming Interface. 9, 59, *Glossary:* API

BFS Breath First Search. 47, 54

CPS Combain Positioning Service. 39, 62, 64

CPU Central Processing Unit. 35

CSV Comma-Separated Values. 59

GNSS Global Navigation Satellite System. 13

IMU Inertial Measurement Unit. 9, 18, 28, 30, *Glossary:* IMU

LIDAR Light Detection and Ranging. 17, 18, 20, 28, 30, 36, 68, 69

MVP Minimum Viable Product. 1

ROS Robot Operating System. 1, 3, 13–16, 20, 21, 24, 25, 27, 31, 36, 39, 44, 45, 54, 55, 58, 61, 64, 68, 69

RSS Received Signal Strength. 13, 16, 20, 21, 57–59, 62, 67

SLAM Simultaneous Localisation And Mapping. 1, 10, 13–16, 18, 20, 21, 27, 30, 31, 34, 36, 43, 45, 59, 69, *Glossary:* SLAM

USAR Urban Search And Rescue. 20

VPN Virtual Private Network. 10, 19, *Glossary:* VPN

WGS84 World Geodetic System 1984. 21, 39, 40, 58, 62, 64, *Glossary:* WGS84

Chapter 1
Introduction

This chapter introduces the problem statement for this Master's Thesis, and provides a comprehensive background. Furthermore it also contains a quick introduction to the Robot Operating System (ROS).

1.1 Background

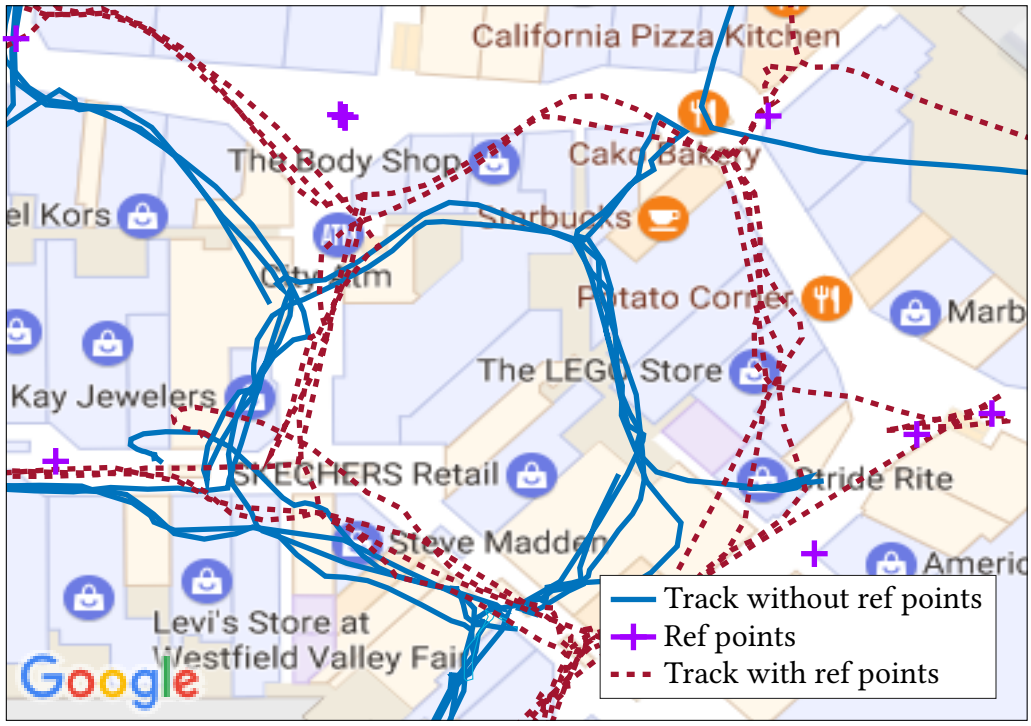
Combain Mobile AB is a world-leading provider of mobile positioning solutions that work without the requirement of a Global Navigation Satellite System (GNSS). With one of the largest crowdsourced database of Wi-Fi and cellular tower locations, they offer accurate positioning globally and have recently focused a lot on improving their indoor positioning.

Until recently, Combain has been utilising their database to perform a coarse trilateration-like localisation by measuring Received Signal Strength (RSS). Internal benchmarks show that the produced results can have an average error of 25 metres in urban areas. To further improve these results, Combain has a produced a research paper on incorporating the temporal factor into a Simultaneous Localisation And Mapping (SLAM) algorithm [6]. This algorithm is based on an optimisation problem, where both temporal and spatial data is taken into account and weighted using a combination of likelihood factors. This has made it possible for Combain to offer a localisation service with an average error of less than 10 metres, slicing the error in half.

The indoor positioning service can be accessed today using a smartphone application which performs SLAM to track both the device and access points in indoor environments [7]. This differs from the older trilateration method which used one-off, standalone measurements. The consequence is a negative battery impact and ultimately that the crowdsourcing of data can no longer be performed automatically in the background. Additionally, because the database currently only has an accuracy of 25 metres, initial SLAM sessions are needed to train and improve the database. The result is that an oper-

ator has to manually traverse whole buildings, and preferably at regular intervals, mark their current position on the map. Manual positioning unfortunately introduces another layer of error, making the result of the SLAM dependent on the accuracy and number of reference points, which are typically inaccurate up to several metres.

Comparison of tracks



Credit: *Combain Mobile AB*

Figure 1.1: Existing algorithm with and without the use of reference points. Notice the user-placed reference points which are positioned outside the stores. Notice also how the track without reference points has been incorrectly placed inside the stores.

1.1.1 Purpose and Goal

The importance of collecting accurate reference points can be seen in Figure 1.1. Namely, without the use of reference points, the path is incorrectly placed inside the stores. Combain would therefore like to evaluate whether it is possible to automate the data collection using an autonomous vehicle. Combain would also like to use this data as ground truth for improving and verifying their own algorithms. They would also like to evaluate if there exist any efficient methods to ensure complete coverage while collecting measurements. Lastly they would prefer a user friendly interface for controlling the collection.

The goal of this master's thesis is to evaluate the usage of different ROS plug-ins in the realisation of the above stated system. This will be done by implementing a subset of the system, mainly a singular robot which will attempt to cover an entire floor using

different algorithms and given different starter conditions. If necessary, custom ROS plug-ins will be implemented if there are no existing implementations with sufficient features.

One of the main challenges in this project is the requirement to offload as much of the main processing as possible to a remote server. The reason is that the autonomous vehicle would require less processing power and less manual configuration, since they would follow instructions from the server. This will take the form of a fleet management platform.

1.1.2 Use Cases

The report studies two primary use-cases for evaluating all of the (appropriate) methods. They will be referenced throughout the report as below.

UC-Map Gathering data in a known environment.

UC-NoMap Gathering data in an unknown environment.

1.2 Robot Operating System

The Robot Operating System [8] can best be described as a collection of open-source software frameworks for building robots. It consists of multiple interchangeable plug-ins which can be used as building blocks for many robotic projects.

ROS was originally developed at Stanford university during 2007 as a collection of prototypes. Willow Garage, a nearby robotics research lab and incubator, took an interest in the project and together with researchers at other institutions created the Robot Operating System. Most of the development continued at Willow Garage as an open-source project until the beginning of 2013, at which point the stewardship was transitioned to the Open Source Robotics Foundation; it was later announced that Willow Garage was being absorbed by another company. Development has since continued under the Open Source Robotics Foundation, and amongst the contributors are research institutions and companies, as well as individual developers.

This thesis uses the ROS platform as its primary foundation, since it contains many of the required features, like robotic control and SLAM algorithms. ROS readily allows usage of software written in either C++, Python or Lisp. And even though both authors have prior experience using C++, it was decided to use Python wherever possible since it typically allows for faster prototyping than C++. The authors also feel that the performance gain given by C++ would not be required.

1.2.1 Navigation Stack

ROS, being extremely modular, uses the concept of *global* and *local* planners. Together they make up the navigation stack called *move_base* and provide a standardised interface for other plug-ins to interact with. This interface is based on an A-to-B navigational model, meaning that the input consists of a *start* and a *goal* pose.

The two planners differ mainly in their abstraction level, where the global planner is responsible for planning a long term path which the local planner is then supposed to execute. The common practice seems to be that only the global planner uses the SLAM map, while the local planner uses raw sensor data. This allows the local planner to react much faster to changes in its environment and is therefore also tasked with avoiding nearby moving obstacles. In many cases this obstacle avoidance results in a deviation from the path made by the global planner, and a return to it further down the line.

Worth noting is that the global planner is continuously, at regular intervals, asked to recalculate its path as the robot moves. This allows the global planner to re-evaluate the optimal path based on eventual changes in the map. The local planner, on the other hand, is running and controlling the robot continuously, and therefore needs to be able to take these discrete updates into account as they become available.

1.3 Simulations

ROS provides multiple tools for running simulations, the primary tool used in this thesis is the Gazebo simulator [9], which is a 3D environment simulator. The Stage simulator [10] was also used, but to a lesser extent, during the later stages of the project. It was mostly used because of its ability to simulate environments using existing maps created during SLAM sessions.

ROS does not have built-in support for the simulation of Wi-Fi access points, meaning that it would not be possible to simulate data collection. After careful consideration it was decided to leave it like that and not implement any support. Partly because it would take valuable time, but also because it would ultimately not add any value, since the RSS are merely collected and stored.

1.4 Robot Hardware

For the robot hardware, it was decided to use the popular *Turtlebot 2* platform [11], which in turn is built upon the Kobuki base [12]. Kobuki, while sharing many similarities with commercially available robot vacuum cleaners, offers serial communication capability through USB for control. See Figure 1.2 for an image of the robot.

The main contributing factors for using the Turtlebot are good documentation and availability. LTH had recently, for the Wallenberg Autonomous Systems and Software Program (WASP), bought a couple of Turtlebots, which the authors had the opportunity to access, and use for the duration of the project.

1.4.1 Controller Unit

The controller unit is the part which runs ROS and its main responsibility is to interact with the hardware, such as the Kobuki base and the sensors. Since one of this thesis' goal was to attempt to offload as much computation as possible to a remote computer, an attempt was made to use a *Raspberry Pi* as the controller unit. A Raspberry Pi [13]

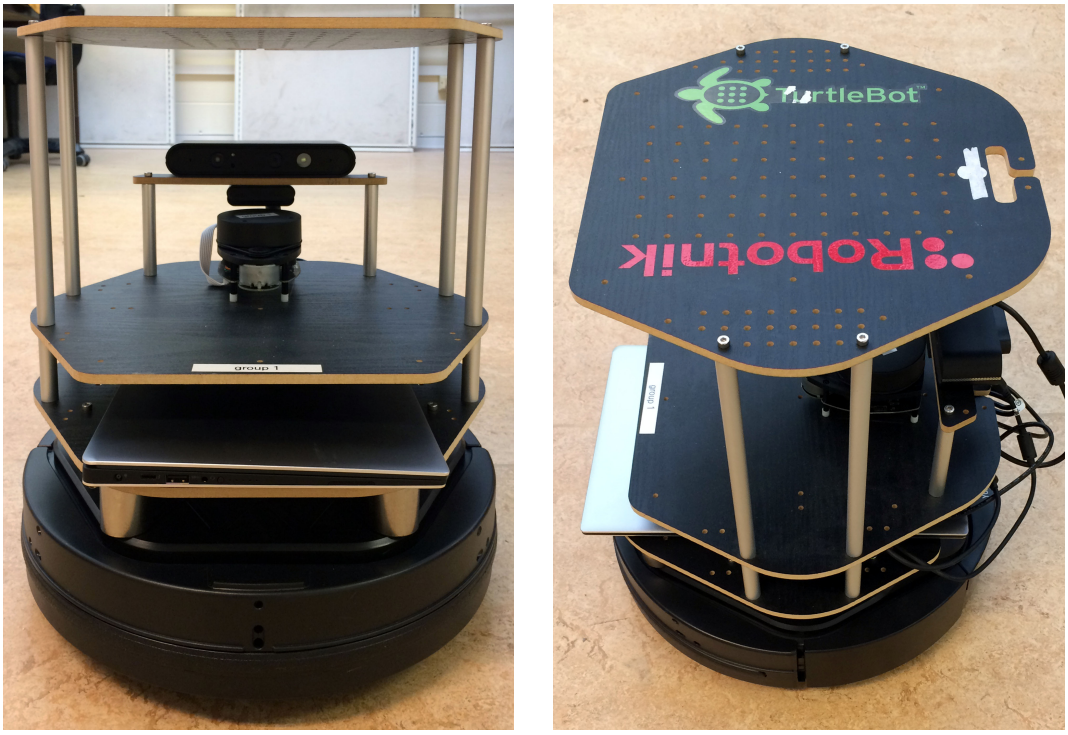


Figure 1.2: The Turtlebot used for the project. Note the base's similarities to commercial robot vacuum cleaners.

is a single-board computer, about the size of a credit card and is about as powerful as a smartphone.

After several evaluations it was decided to use a more powerful notebook as the control unit instead, since the latency between the remote computer and the Raspberry Pi proved to be too high. After switching to running processes locally on a notebook, the system became more responsive and managed to evade moving obstacles much better. The notebook was equipped with an Intel Core i7-6560U 3.2 GHz with 2 physical and 4 logical cores.

1.4.2 Sensors

The Turtlebot available at LTH was equipped with a 3D depth camera and a laser Light Detection and Ranging (LIDAR) device. Both sensors are usually used primarily for indoor mapping and positioning. Trials at the beginning of this thesis, showed that the LIDAR performed significantly better than the depth camera and was therefore used as the primary sensor, while the 3D depth camera was left reserved for future work.

RPlidar

The *RPlidar A1* [14] is a low-cost 360 degree laser LIDAR device, manufacturer by *Slamtec*. It has, according to the manufacturer, a maximum range of 6 metres, with a distance resolution of 0.2 cm and angular resolution of 1°. The sample rate is 2000 samples/s, and has a full scan frequency of 5.5 Hz.

The LIDAR's primary purpose is to scan and collect data about the environment. The data is then used to perform SLAM which generates an indoor map. However, since the LIDAR is only able to see obstacles at a specific height, it has a lot of problems detecting non-uniform obstacles such as chairs or tables, and even obstacles which are only visible at low heights, e.g. doorsills.

Orbbec Astra Pro

Astra Pro [15] is a 3D 720p RGB camera, manufactured by *Orbbec*. While it has a narrower field-of-view than the RPLidar, because it can see in 3D, it is able to see obstacles at different heights which the RPLidar cannot. It is also possible to detect a drop in floor level, like stairs.

Bumper sensor

The Kobuki base has a built-in bumper at the front which can sense when the robot has hit an obstacle and roughly where that obstacle is. The default behaviour, which reverses the robot away from the obstacle, was not modified.

Cliff sensor

The Kobuki base has 3 built-in IR LEDs which react when they detect a drop, e.g. stairs. The default behaviour, which puts the robot into full reverse away from the drop, was not modified.

Wheel drop sensor

The Kobuki base has a built-in drop sensor for each wheel which react when the wheel fall. Because the wheels are forced downwards by springs, they instantly fall into holes or other openings. The default behaviour, which completely disables the robot, was not modified.

Inertial Measurement Unit

The Kobuki base has a built-in Inertial Measurement Unit (IMU) in the shape of a three-axis gyroscope. The IMU is used internally to improve the angular accuracy of the *Odometry*, specifically to detect yaw (rotational) movement.

Odometry

The Kobuki base has a built-in odometry sensor which can measure the speed and distance travelled by the wheels. It uses the built-in *IMU* to improve the angular accuracy.

1.5 Network

One of the major potential problems with distributed systems in general is to ensure bi-directional communication between all devices. To solve this problem, a Virtual Private Network (VPN) was setup to simulate a local network over the Internet. This ensured that all devices always had bi-directional communication, even though the devices were actually on different networks, potentially behind firewalls. The only requirement for this solution was for the VPN host to be publicly accessible on the Internet. This was solved by renting a virtual private server. A more detailed overview of the network topology can be seen in Figure 1.3.

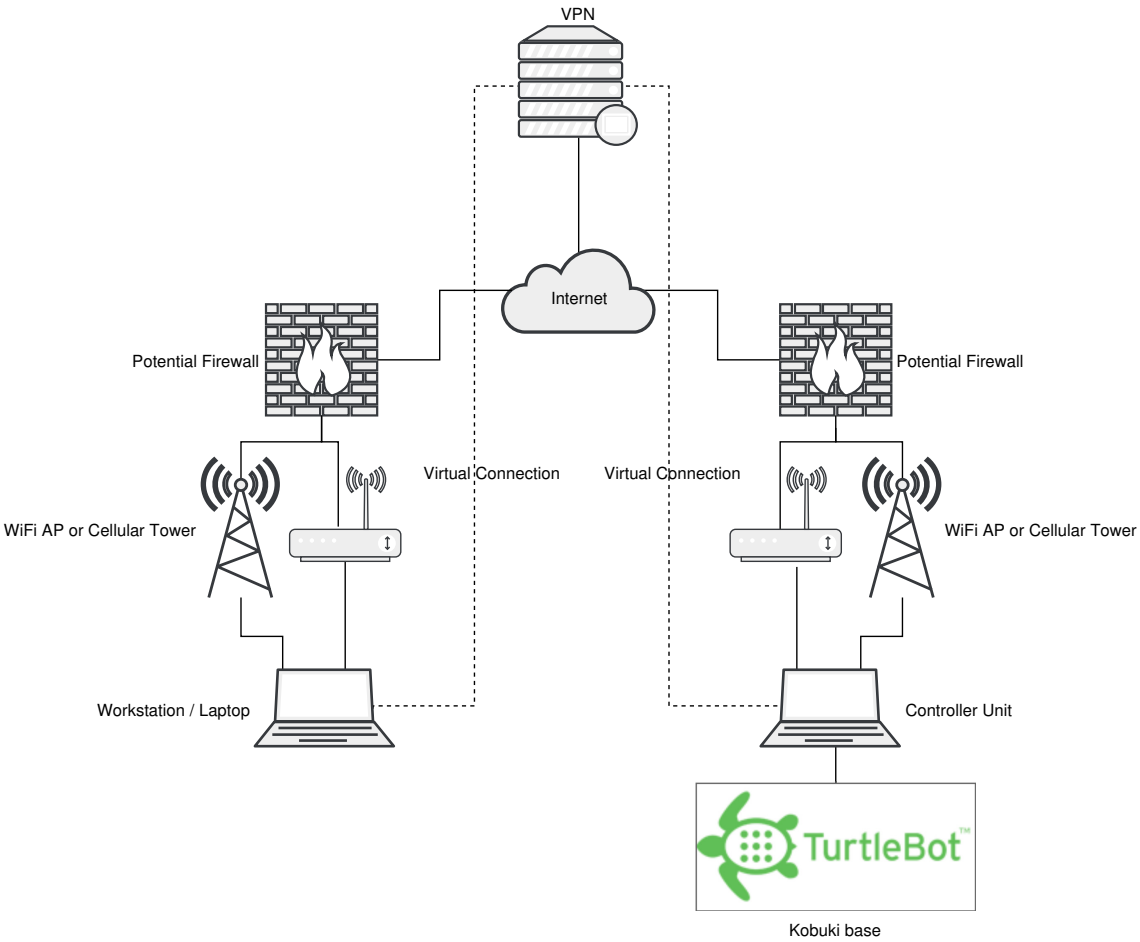


Figure 1.3: Illustration of the network topology. Note that the workstation and turtlebot are both on the same virtual network, which allows them to bypass potential firewalls.

1.6 Related Work

Autonomous exploration and area coverage are two already well-established fields in robotics. Similar solutions for the goals in this thesis already exist, but none provide all of

the requirements in a single platform. Below are some of the projects that can be found online.

Yamauchi et al. [16] developed ARIEL, a system for autonomous exploration and map building using continuous localisation. The article describes how all the parts of the system were built up and integrated from scratch. However, the article was published during the late 90's and the hardware and software used is long since obsolete. Riaz et al. [17] describe a similar system inspired by [16], but built on a more modern platform. Unfortunately, the system is based on proprietary software, and uses algorithms implemented from scratch, making it hard for the community to replicate the work.

The work by Kohlbrecher et al. [18] took a different approach and produced publicly available open-source plug-ins based on algorithms like the Hector SLAM algorithm by Kohlbrecher et al. [19] and the exploration algorithm by Wirth and Pellenz [20]. However, [18] focuses on Urban Search And Rescue (USAR) missions where modern high-performance LIDAR systems are available, and mentions that their performance is high enough not to require features like explicit loop closure. This might have been true in their case, but will be shown in this thesis, that it is unfortunately not a universal truth.

Some work on Wi-Fi RSS scanning in ROS has been done by Scholl et al. [21], but is in the form of a handheld unit running the algorithm mentioned above by [19]. Because of the algorithm used, the project could also suffer from errors that might have been corrected by loop closures. The article seems like a start towards the same goals as in this thesis, but no further information can be found.

This thesis acknowledges that the open-source community has implemented many high-quality algorithms and does not try to re-invent the wheel. Instead it evaluates and compares the most popular plug-ins available, to create a publicly accessible recipe for anyone to use. All of the considered plug-ins have been used as-is to the largest extent possible, and any extensions that have been made, are publicly available on the popular site GitHub under the organisation Robotslam[1]

1.7 Contribution

As seen in the previous section, there is a lot of existing material which cover the individual components used in this thesis. This thesis instead aims to build upon the existing knowledge to create a complete solution which can be used to solve the two use-cases **UC-Map** and **UC-NoMap**, described in Section 1.1.2. In order to achieve this, a comprehensive evaluation of the existing material and algorithms had to be completed in order to ensure the best possible result. Furthermore, a comprehensive platform was developed to ensure that the result of the project could be used by people who posses less technical knowledge. Both authors have contributed equally to the thesis, which in summary has:

- Evaluated the performance of the most popular ROS plug-ins to make it easier for future projects to make a proper choice based on their requirements.

[1] `https://github.com/robotslam`

- Extended the popular plug-ins Gmapping and Google Cartographer with the ability to output their trajectory over the standard ROS interface.
- Added waypoint-following capabilities to ROS by creating an entirely new global planner.
- Created a Wi-Fi RSS scanning plug-in which supports grouping of measurements by time.
- Created a fleet management system to easily operate and organise data-collection sessions by adding an abstraction layer on top of ROS.
- Extended the fleet management system with the ability to transform local coordinates to the latitude and longitude format used by World Geodetic System 1984.
- Extended the fleet management system with the ability to correct Wi-Fi RSS measurement positions using trajectory data.

1.8 Disposition

The thesis is structured as mostly isolated chapters which each describe a specific component of the thesis project. A short summary of the chapters are provided below.

Chapter 2 Presents the methodology used in this thesis and describes the different phases of the project.

Chapter 3 Provides a background to Simultaneous Localisation And Mapping, evaluates the different SLAM algorithms, and measures the accuracy of the algorithms in a known environment.

Chapter 4 Describes how the internal coordinates used in ROS can be converted into the World Geodetic System 1984 (WGS84) standard.

Chapter 5 Evaluates a few existing area coverage algorithms and describes how they were implemented.

Chapter 6 Describes frontier based exploration and evaluates the existing ROS implementations.

Chapter 7 Describes how the Wi-Fi measurements were collected and how they connect to the existing infrastructure at Combain.

Chapter 8 Describes the new platform and web interface for fleet management.

Chapter 9 Ties everything together and discusses the final result. It also introduces a couple of topics for future work.

Chapter 2
Methodology

This chapter will provide a brief overview of the methodology and the schedule of this thesis. It also describes the differences between simulation and real world trials.

2.1 Approach

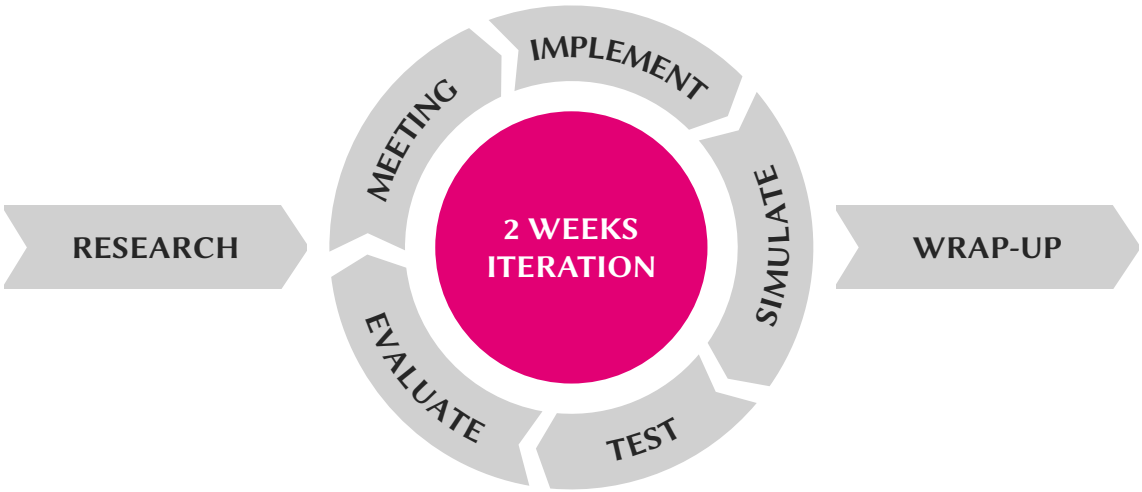


Figure 2.1: Illustration of the iterative process used during this thesis.

Due to Combain's initial uncertainty of what was possible to achieve during a thesis project, it was, at first, not possible to define any formal goals. Because of this situation and the agile nature of Combain's regular projects, it was decided to follow a more iterative process during this thesis (see Figure 2.1).

2.1.1 Initial Research

To get a better picture of what would be possible to accomplish during the length of this thesis project (20 weeks), an initial research stage was planned for the first 3 weeks. The purpose was to gather scientific material and look into how much ROS was capable of out-of-the-box. This effectively meant experimenting with ROS tutorials and demos, and evaluating how the existing functionality could be used in combination with published articles to provide the desirable results.

In reality, this stage started to melt into the iteration process almost immediately, and then continued as pure research in parallel to the iterations up until week 5.

2.1.2 Iterations

Figure 2.1 describes the iterative process, which served as the schedule's primary foundation. Each iteration lasted around two weeks, and consisted of mainly five different stages:

Meeting Each iteration started and ended with a meeting, in which the previous iteration's result was presented and evaluated. Afterwards the new iteration was planned with a new goal. This meeting was often complemented by a second meeting in the middle of each iteration which gave the opportunity to have more in-depth discussions.

Implement The problem statement defined in the *meeting* is researched and implemented. This can be anything from writing a new algorithm to tweaking existing implementations to better fit the requirements.

Simulate The implementation is tested and verified in a simulated environment. (see Section 2.2)

Test The implementation is tested and verified in a real-world environment. (see Section 2.2)

Evaluate The final step in each iteration is to evaluate the solution: Does it solve the problem defined during the *meeting* and is there anything that can be improved?

2.1.3 Wrap-up

The last couple of weeks were spent wrapping everything up. This mostly consisted of writing the last parts of the report and collecting more measurements, as well as collecting ground truth data about the environments. This data was then used to evaluate the accuracy of the final solution.

2.2 Simulated and Real World Trials

As previously described, ROS provides several tools to simulate robots and environments. In this thesis, a majority of the time was spent in a simulated environment, since it

provided several benefits, eg. that it was much faster to prototype, since there was no need to have physical access to the Turtlebot. Testing using a physical robot was mostly done at the end of each iteration to confirm real world performance.

A disadvantage with using the simulator was its difference to the real world. An example is that there does not seems to be any friction between the wheels and the ground, resulting in the wheels spinning freely even when the robot is driving in to a wall.

2.3 Schedule

The schedule in Figure 2.2 shows an overview of the time spent for each iteration in the project. Each iteration took around 2 weeks to complete.

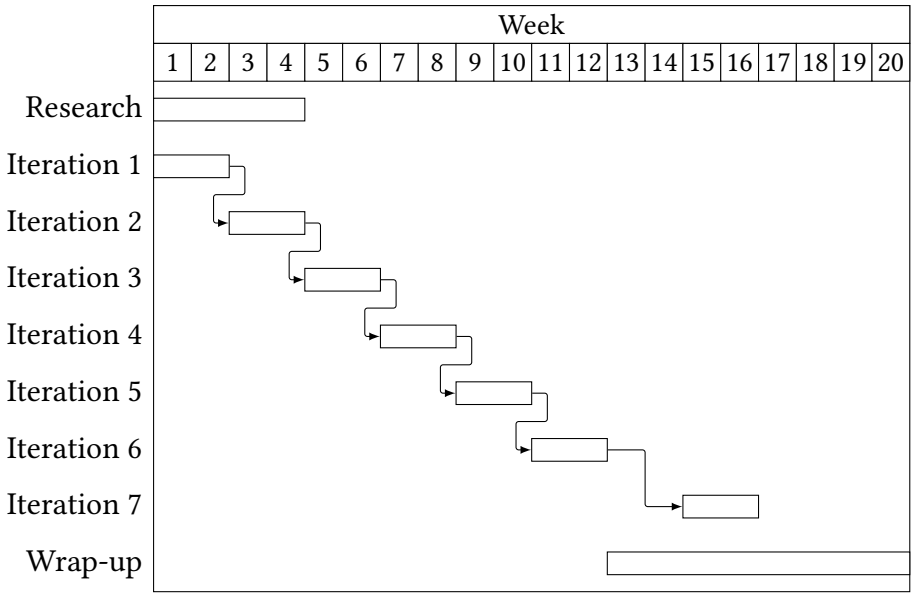


Figure 2.2: Schedule for the project. Notice how each iteration took 2 weeks to complete.

Research See Section 2.1.1.

Iteration 1 Set up computer environments to support ROS, experiment in simulated environments and defined preliminary use-cases. Implement Wi-Fi scanner for ROS.

Iteration 2 Experiment with real robots, discovering the differences between simulated and real robots, and fix configuration issues. Set up Raspberry Pi and network structure over VPN. Begin initial evaluation of different SLAM algorithms. Improve simulation, add support for localisation in a known map (**UC-Map**), measure drift and research conversion between coordinate systems.

Iteration 3 Begin work on the fleet management platform (see Chapter 8), implement coordinate transformation algorithm (see Chapter 4) and create an easy-to-use

interface for it. Save Wi-Fi measurements to file and convert to GPS coordinates. Implement global planner for following waypoints and switch over to using a notebook instead of a Raspberry Pi.

Iteration 4 Research exploration, and path coverage. Implement and evaluate path coverage based on the path transform algorithm (see Chapter 5). Path coverage works autonomously for the first time. Initial evaluation of most popular exploration algorithm.

Iteration 5 Experiment with 3D depth camera and major improvements to the fleet management platform's functionality and usability. Exploration works autonomously for the first time and the collected measurements are successfully exported to Combain's system.

Iteration 6 Automation of export to Combain's system and other general improvements to fleet management platform. Test cliff sensors. Further test exploration and improve positioning of measurement after loop closure using trajectories.

Iteration 7 Further investigation in area coverage, implemented spanning tree coverage and evaluated it against distance path transform.

Wrap-up Collect reference (ground truth) data, perform more measurements and evaluate the results. Finish report. See Section 2.1.3.

Chapter 3

Simultaneous Localization And Mapping

Simultaneous Localisation And Mapping is a well-researched topic in many areas, including mobile robotics. ROS provides several ready to use implementations of different SLAM algorithms. In this chapter the most popular algorithms, Gmapping, Cartographer, and Hector SLAM are presented and evaluated to see which one provides the best result for **UC-NoMap**. The focus will not be on researching or implementing new algorithms related to SLAM.

3.1 Motivation

Having an accurate SLAM algorithm can be motivated in many ways. While the goal of this thesis is not to create high-resolution indoor maps, they play a vital role as they are used to place the measurements in the real world. In order to achieve this, there is a need for an accurate description of how the robot has travelled, i.e. the trajectory.

3.2 OpenSlam's Gmapping

Gmapping is the default provided SLAM algorithm in ROS. It uses LIDAR and odometry measurements to locate and map its surroundings. It is based on Rao-Blackwellized Particle Filters and supports loop closure. The articles [22, 23] describe the actual algorithm in detail. The standard configuration of Gmapping also relies heavily on an externally provided odometry. This means that inaccurate sensor readings will be detrimental to the mapping result.

3.2.1 Impression

The first hands-on experience in the simulated world was promising, resulting in maps that mostly managed to correct themselves when the robot arrived back at known territory. A small issue was how the algorithm reacted when driving into a wall, where the whole map would drift at the same speed as the robot's simulated wheels, as if the map was moving away from the robot. The problem seemed to originate from the simulator reporting positive wheel speed even though the robot was not actually moving. The behaviour could not be replicated in the real world since the bump sensor would trigger and reverse the robot away from the wall.

When Gmapping was first used in the real world, the algorithm suffered from an issue that made the map "hairy" as seen in Figure 3.1a. After closer inspection, it was concluded that the hairs were missing LIDAR points which the algorithm interpreted as free space with the same radius as the LIDAR's specified maximum range. The hairs were successfully removed by configuring Gmapping to ignore LIDAR values past its maximum rated range. The algorithm then performed as good as it did in the simulated world.

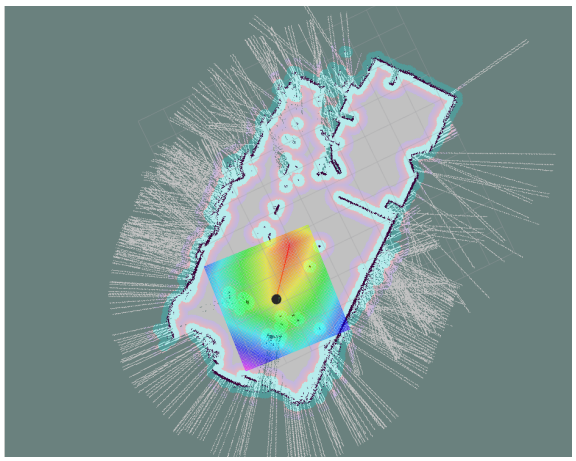
3.2.2 Loop Closure

Gmapping does not have a dedicated method for identifying loop closures, instead it is a consequence of how the algorithm itself works. In short, Gmapping always keeps a predefined number of randomly generated particles, i.e. hypotheses of how the robot has travelled. Combined with what the robot has seen at every scan a map can be drawn for every particle. Because these particles also have a continuously re-evaluated likelihood attached, the most likely particle is the one used to draw the map which is displayed to the user. The likelihood is calculated by comparing the last LIDAR scan with what the different particles predict. In long featureless corridors, it is therefore hard to determine if the particle which has travelled e.g. a few centimetres further than another is the correct one or not, because both predict equal scans. However, when the robot finally arrives back to a known feature-full territory, the algorithm quickly identifies the most likely particle and the map changes.

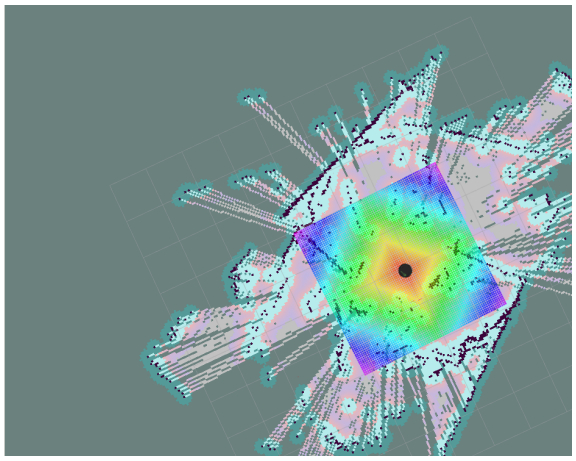
3.3 Hector SLAM

Hector SLAM is based on the paper by Kohlbrecher et al. [19] which does not require any other data than LIDAR measurements. It has support for IMU sensors to compensate for tilt but no loop closure.

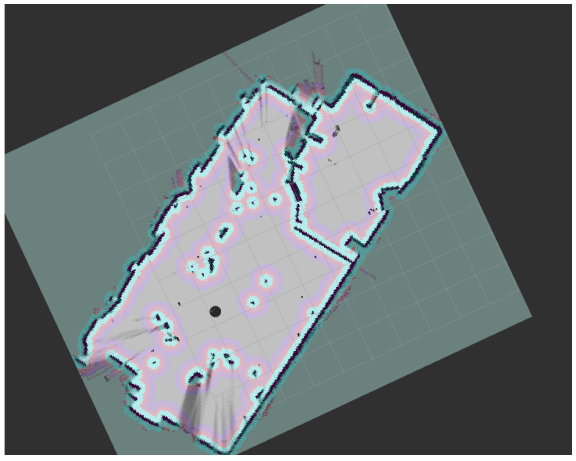
The Hector SLAM algorithm is described in [19] as consuming "low computational resources" and mention that other algorithms do not "leverage the high update rate provided by modern LIDAR systems". The mentioned LIDAR system is most likely a reference to the *Hokuyo UTM-30LX* LIDAR device used in [19], which has a scan frequency of 40 Hz (compared to the 5 Hz of the LIDAR device used in this project).



(a) Gmapping result.



(b) Hector result.



(c) Google cartographer result.

Figure 3.1: Mapping *exjobbsrummet* (room reserved for students doing their master's thesis) in the M-building on the LTH campus.

3.3.1 Impression

Hands-on experience in the simulated world showed great promise with little to no drift, i.e. the robot was always where the algorithm said it was. The results strengthened the claim made by Kohlbrecher et al. [19] that in many scenarios, optimisations such as loop closure are not needed. Furthermore, since the algorithm does not utilise the odometry, it did not suffer from the same problems as Gmapping in which the simulator reported movement on the wheels while the robot stayed still.

However, the real world testing was a disappointing exercise. While straight movements were tracked accurately, rotation was a complete disaster. The cause might have been the low scan rate of the LIDAR device. Because of this, the idea of using Hector SLAM was abandoned early in the project. An example of a map created by Hector can be seen in Figure 3.1b.

3.4 Google Cartographer

Cartographer is a real-time SLAM library developed by Google and was recently open-sourced in October 2016. It supports loop closure and IMU sensors to compensate for tilt and also enable 3D tracking. It is based on the paper by Hess et al. [24]. More information on how it differs from the other algorithms can be seen below.

3.4.1 Impression

Cartographer was unique in offering the same impression in both the simulated environment and the real world. Cartographer was also unique in being able to produce maps which contained information on the likelihood of an area being occupied or free. This was done by allowing the map to hold the entire range of values between black and white, representing the likelihood that a certain space contains an obstacle. In comparison, the other two algorithms only use the three discrete values: white, grey and black to represent: free, unknown and occupied space.

3.4.2 Loop Closure

Cartographer contains a more explicit loop closing method which is separate from its main algorithm, this makes it different to Gmapping. In short, Cartographer continuously creates small localised maps from very few LIDAR scans, called *submaps*. These submaps are stored as nodes in a graph structure, where geographical constraints are added between the nodes as edges. This graph is then treated as a large optimisation problem and solved using the Ceres solver, and the solution is then used to draw the map. Loop closure is accomplished, in a separate process, by identifying relationships between nearby, but previously unrelated nodes in the graph using scan matching. By adding an edge between these nodes, the next time the map is drawn the nodes will be linked and the loop closed.

3.5 Evaluation

A critical requirement in this thesis is to ensure the accuracy of the produced trajectory. This can be done by ensuring an accurate map, since the map serves as the foundation for the trajectory, which in turn is used to map the measurements to real world positions.

The tests were conducted by manually controlling the robot using the provided ROS package *teleop*, which allows for remote operation using a keyboard. All the sensor data was recorded using the ROS package *rosbag*, which allows the recorded data to be re-played at a later time. This way it was possible to run the exact same scenario multiple times using different SLAM algorithms.

3.5.1 Loop Closure

Loop closure is a critical part of mapping which continuously tries to match the current location with previously visited ones. It mitigates erroneous overlapping by matching new scans with old ones and adjusting the map, and current position, accordingly. As mentioned above, both Gmapping and Cartographer support some form of loop closure. An example of Cartographer performing loop closure can be seen in Figure 3.2. However, since loop closure does not refer to a single method, the evaluated algorithms each have their own techniques.

In practice, these techniques produced very different results. For example, in Figure 3.3 it can easily be seen how the randomisation part of Gmapping produces unique results every time the algorithm is executed on the same data set. This unfortunately means that Gmapping has the same negative intrinsic property that all particle filters have — the inability to deterministically guarantee a good solution.

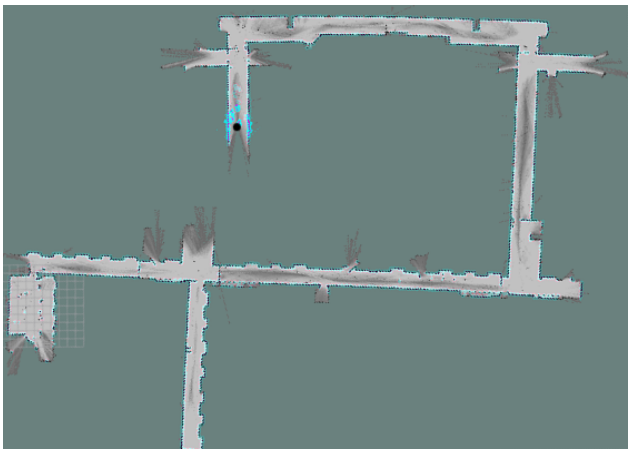
Cartographer, on the other hand, has succeeded in closing the loops every single time, as can be seen in Figure 3.3. Combined with the fact that Cartographer is based on a deterministic algorithm, it clearly stands out from the rest.

In summary, by failing to close loops 77% of the time, Gmapping performs significantly worse than Cartographer, something which can happen in even the best of circumstances due to the uncertain nature of the algorithm. When the loops *are* closed however, they seem to get a lot less distorted by Gmapping than Cartographer, as can be seen in Figure 3.4.

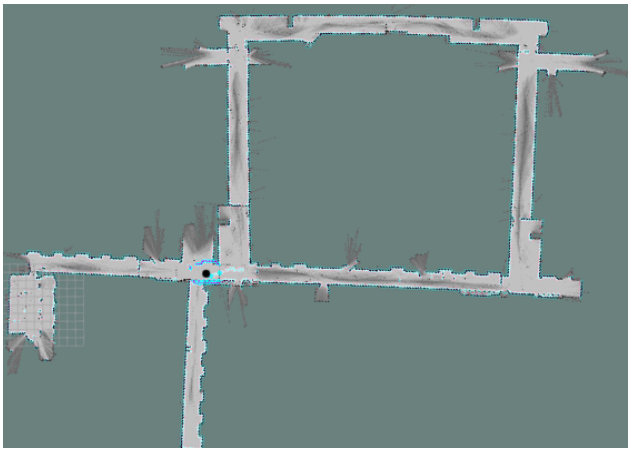
3.5.2 Accuracy

To determine the accuracy of the maps, the physical location was measured using a *laser rangefinder*. The rangefinder had a maximum range of 50 m with an accuracy of ±2.0 mm. However, because the rangefinder was operated by hand, the final result could realistically only have had decimetre accuracy.

Since Gmapping uses a non-deterministic algorithm, it was decided to replay the data 10 times and calculate the mean distances in order to get a more accurate comparison. The test results which can be seen in Table 3.1 shows that the mean error for Gmapping was 1.9%, Cartographer on the other hand had a higher mean error of 2.3%, but since it had a deterministic behaviour it tended to provide more reliable results.



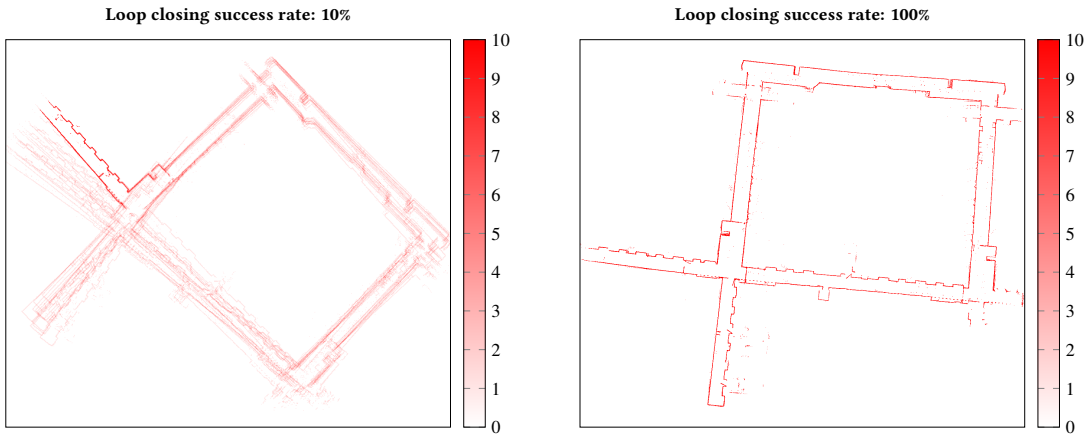
(a) Robot moving towards known territory.



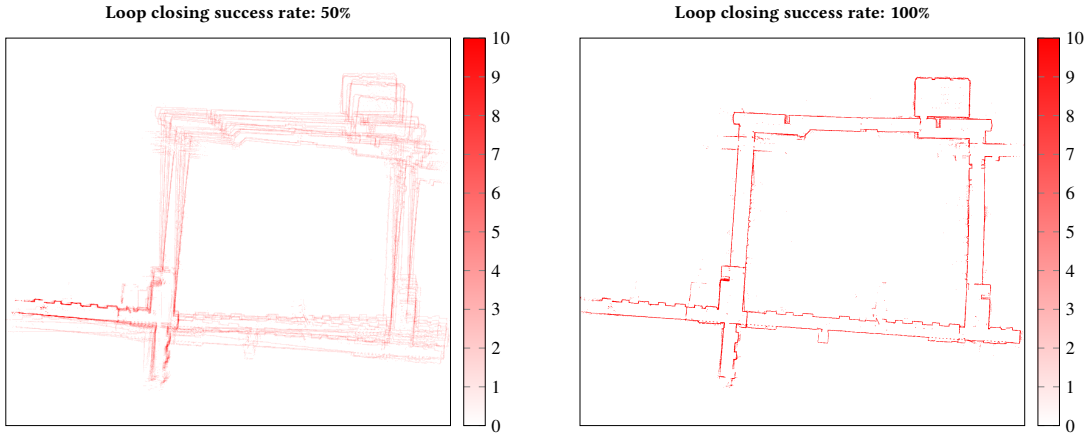
(b) Robot has arrived in known territory but has not yet recognized it.

(c) Robot has recognized the territory and performed appropriate adjustments to the map.

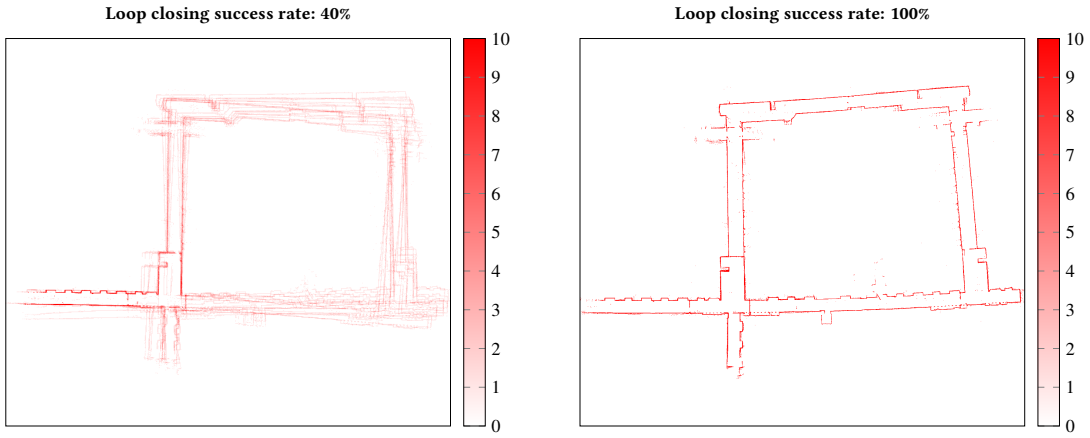
Figure 3.2: Cartographer performing loop closure in M-building, LTH. The dot representing the robot has been enlarged to improve visibility.



(a) Gmapping 2017-03-31-01 **(b)** Cartographer 2017-03-31-01

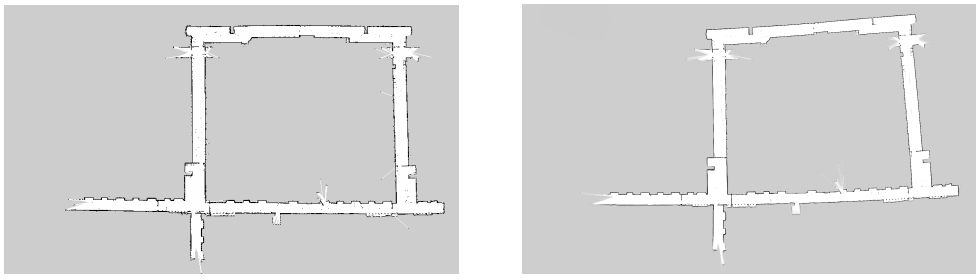


(c) Gmapping 2017-04-06-01 **(d)** Cartographer 2017-04-06-01



(e) Gmapping 2017-04-06-02 **(f)** Cartographer 2017-04-06-02

Figure 3.3: Visualisation of the different outputs from multiple runs of the same algorithm on the same data sets. Notice how Gmapping produces different maps every time while Cartographer does not. Notice also how Gmapping fails to close the loop and how Cartographer succeeds, but distorts the map in the process.



(a) Gmapping. **(b)** Cartographer.

Figure 3.4: A comparison between Gmapping and Cartographer when both have successfully closed a loop. Notice how Cartographer distorts the map compared to Gmapping.

Another critical criteria which Gmapping had large issues with was loop closure, as mentioned in Section 3.5.1. The length of the corridors were still accurate, but because of the failed loop closures, the last corridor failed to line up correctly.

Both algorithms had problems handling the left corridor where the error ranged between 0.3 m – 1.8 m. In comparison, the bottom corridor only had a maximum error of 0.3 m, lending to the conclusion that some corridors are easier to map than others.

Table 3.1: Measurements in M-huset at LTH. Note that for *Gmapping*, the error of each corridor was calculated as an average over 10 algorithm executions for each data set.

	Left	**Top**	**Right**	**Bottom**	**Error**
Reference	32.1	41.3	32.2	44.7	
Gmapping #1	31.7	40.7	32.0	43.3	2.6 (1.7%)
Gmapping #2	31.1	40.7	31.5	44.3	2.7 (1.8%)
Gmapping #3	31.8	40.2	31.5	43.4	3.4 (2.3%)
Cartographer #1	31.5	39.8	31.6	44.4	3.0 (2.0%)
Cartographer #2	30.3	40.9	32.0	44.3	2.8 (1.9%)
Cartographer #3	30.4	39.2	31.9	44.4	4.4 (2.9%)

3.5.3 Performance

One of the original criteria for the project was the ability to use a Raspberry Pi as the control unit for the robot. This put a lot of requirements on the algorithms since it limited the available data transfer bandwidth. For the SLAM algorithms to be able to run and produce usable results, the settings, primarily the resolution of the algorithms, had to be tweaked. This in turn resulted in a less than satisfying result, and ultimately lead to replacing the Raspberry Pi with a more powerful notebook. All of the results in this report have been produced on the notebook.

Figure 3.5 shows a performance benchmark of the evaluated SLAM algorithms. It shows how Cartographer uses much less resources on smaller maps, but eventually moves

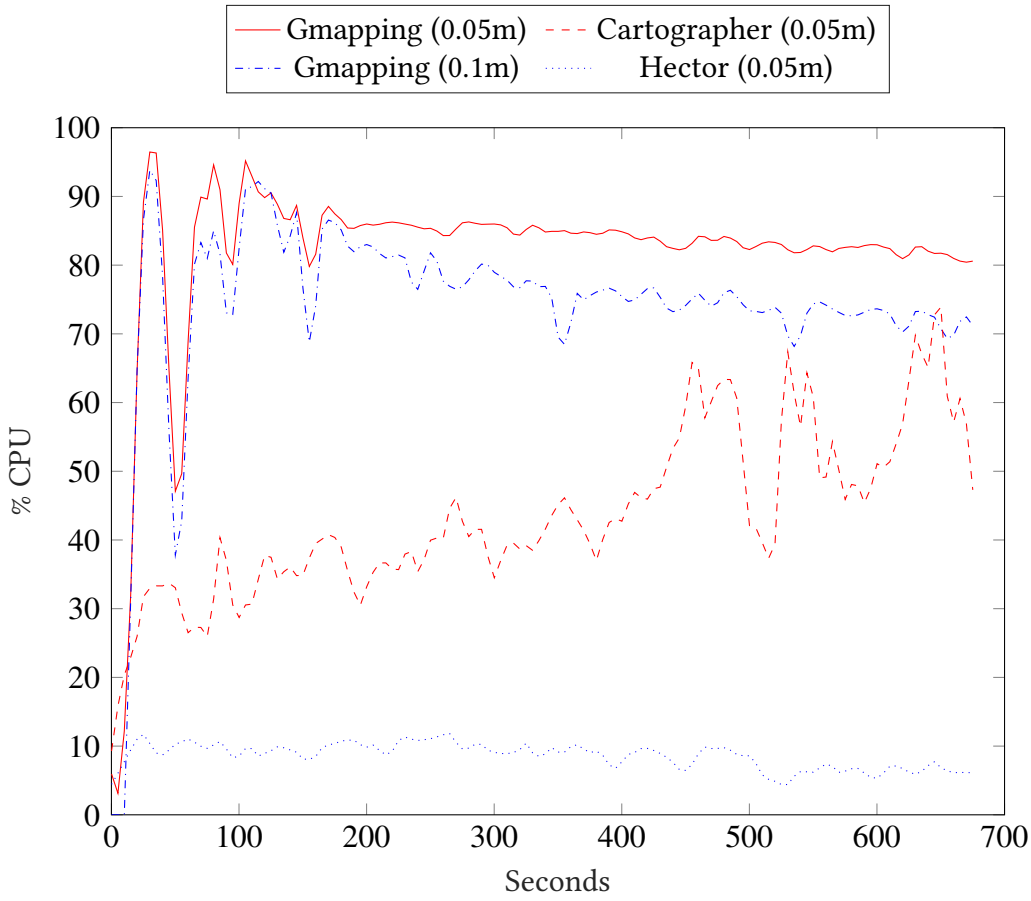


Figure 3.5: Performance comparison of the different algorithms running data set 2017-04-06-02, executed on the notebook. Note that Linux measurements treat each logical core as a separate Central Processing Unit (CPU), meaning that on a quad-core CPU, the maximum percentage is actually 400%. The above algorithms therefore either heavily utilise a single core, or lightly utilise many cores.

up as the map gets bigger. An interesting part is how, at the same time, Gmapping seems to have a downward trend, using up less resources as time goes on. The reason for this behaviour has not been uncovered, but a hypothesis is that Gmapping might begin to miss or throw away incoming data. This might happen because Gmapping does not buffer the input, which e.g. leads to unusable maps when speeding up the replay of the recorded data-sets. Either that or the measurement method was, in some way, inaccurate.

Either way, Gmapping had an average of 82% and 75% CPU usage for 0.05 m and 0.1 m resolutions, while Cartographer had an average of 44%. This means Cartographer is the clear winner over Gmapping in utilising the CPU efficiently.

3.5.4 Trajectory

The trajectory describes the path which the robot has travelled. In order to ensure that the collected measurements can accurately be mapped to specific locations, it is important

to ensure that the trajectory is accurate.

Simply storing the current position of the robot when a measurement is taken is not sufficient, since the trajectory can and will change during a loop closure. Therefore, an important criteria when selecting an algorithm, is to ensure the accuracy of the final trajectory. Figure 3.6 shows the difference between collecting the current position during each measurement and using the trajectory.

Sadly, neither *Gmapping* or *Cartographer* exposed the trajectory using the standard ROS interface. However, after some investigation it was discovered that both algorithms kept an internal representation of the trajectory. Both algorithms were therefore modified to enable access to the trajectory over the ROS interface.

Since *Hector SLAM* does not support loop closure, previous locations in the trajectory are never modified. In this case, it is therefore sufficient to simply save the current location when a measurement is taken.

3.5.5 Conclusion

The purpose of this chapter has been to present and evaluate which of the most popular ROS plugins for SLAM provide the best results for **UC-NoMap**. After gathering a wide range of statistics, the algorithm that has stood out the most has been Cartographer. However, while Hector SLAM has been a clear loser, the distinction between Gmapping and Cartographer has not been clear. Gmapping provides less warped maps and can use odometry data in a way that Cartographer can not. On the other hand, the warping only increases the mean error by 0.4 p.p. and judging by how good Hector performs in the original article [19], odometry is not a problem when a high-performing LIDAR is used. Therefore, after the risk of Gmapping failing at loop closure was factored in, Cartographer was chosen as the recommended algorithm.

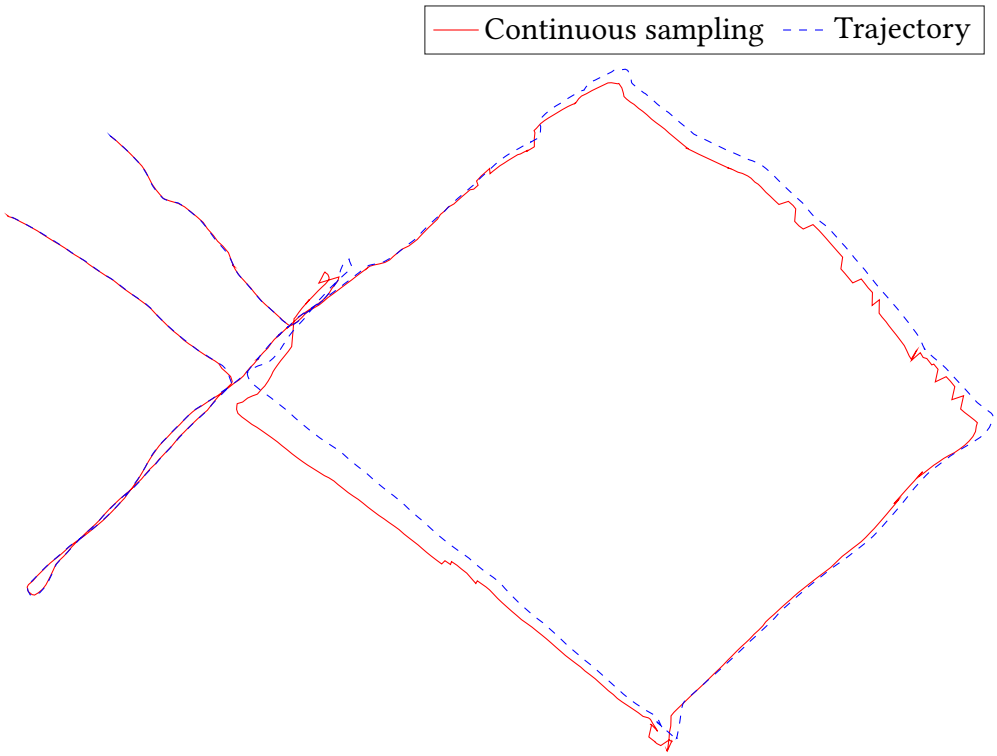


Figure 3.6: Comparison between continuously sampling and storing the position, and the final trajectory. Notice how the continuous sampling in the top corridor contains sudden jumps where Gmapping has switched between particles. The trajectory looks like a smoother and more accurate representation of how the robot might have moved. Notice also that Gmapping has failed to close the loop, and that the left-most parallel corridors are in reality the same corridor.

Chapter 4
Local Cartesian Coordinates to WGS84

ROS uses a local cartesian coordinate system to represent the position of the robot. In order for these coordinates to be usable by external systems, such as Combain Positioning Service (CPS), they need to be converted into a global reference coordinate system. Since CPS already supports the WGS84 standard for use when submitting reference points, it was the natural choice. This chapter presents how an affine transformation can be used to make this conversion with sufficient accuracy.

4.1 Transformation

A method of transforming points from one coordinate system to another is to use an affine transformation matrix. This method uses three reference points to create a transformation matrix between the two systems. The method is well known and well documented, but a quick summary can be read below.

If $\boldsymbol{u_{wgs}} \in \mathbb{R}^{2\times 3}$ are the coordinates of three points in WGS84 and $\boldsymbol{u_{local}} \in \mathbb{R}^{2\times 3}$ are the same points in the local coordinates the relationship between the two can be defined as

$$\boldsymbol{u_{wgs}} = \boldsymbol{A} \cdot \boldsymbol{u_{local}} + \boldsymbol{t}, \tag{4.1}$$

where $\boldsymbol{A}$ is the transformation matrix and t is an offset vector. Using an augmented matrix and an augmented vector, it is possible to represent both the translation and the offset using a single matrix multiplication,

$$\begin{bmatrix} \boldsymbol{u_{wgs}} \\ \boldsymbol{1} \end{bmatrix} = \left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{t} \\ \hline \boldsymbol{0} & \boldsymbol{1} \end{array}\right] \begin{bmatrix} \boldsymbol{u_{local}} \\ \boldsymbol{1} \end{bmatrix} \Leftrightarrow \boldsymbol{v_{wgs}} = \boldsymbol{T} \cdot \boldsymbol{v_{local}}. \tag{4.2}$$

In detail this is the same as

$$\begin{bmatrix} x_{wgs_1} & x_{wgs_2} & x_{wgs_3} \\ y_{wgs_1} & y_{wgs_2} & y_{wgs_3} \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} a & c & t_x \\ b & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{local_1} & x_{local_2} & x_{local_3} \\ y_{local_1} & y_{local_2} & y_{local_3} \\ 1 & 1 & 1 \end{bmatrix}. \tag{4.3}$$

Assuming that the reference points have been properly chosen i.e. not linearly dependent, the transformation is easily retrieved by multiplying with the inverse of $\boldsymbol{v_{local}}$ from the right on both sides:

$$\boldsymbol{T} = \boldsymbol{v_{wgs}} \cdot \boldsymbol{v_{local}^{-1}}. \tag{4.4}$$

The transformation matrix is now able to transform any points in any direction between the two coordinate systems.

4.2 Error Estimation

Because WGS84 coordinates are not linearly distributed, using a (linear) transformation matrix results in an approximation error. A quick estimation of this error was done by calculating the difference in length between a sphere (earth approximation) and a tangent plane (transformation plane built up by the transformation matrix).

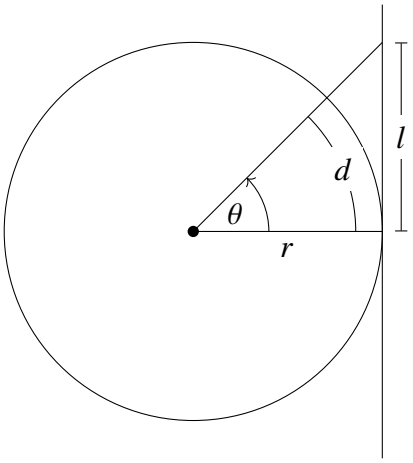


Figure 4.1: Cross section of an earth approximated as a sphere and a tangential transformation plane.

To calculate the error in any given direction, the distances d and l from the point of tangency between the sphere and its tangent plane (see Figure 4.1) need to be identified. By setting up the distance equations

$$\tan(\theta) = \frac{l}{r} \Leftrightarrow l = r \cdot \tan(\theta) \tag{4.5}$$

and

$$d = r \cdot \theta, \tag{4.6}$$

the error can be defined as the difference between the two distances

$$\varepsilon = l - d = r\tan(\theta) - r\theta = r(\tan(\theta) - \theta). \tag{4.7}$$

After setting $r = 63710088$ to the mean radius R_1 of earth [25] the error is plotted as a function of the distance d (see Figure 4.2). The plot shows that even a kilometre away from the point of tangency, the error is still below 0.1 µm. This leads to the conclusion

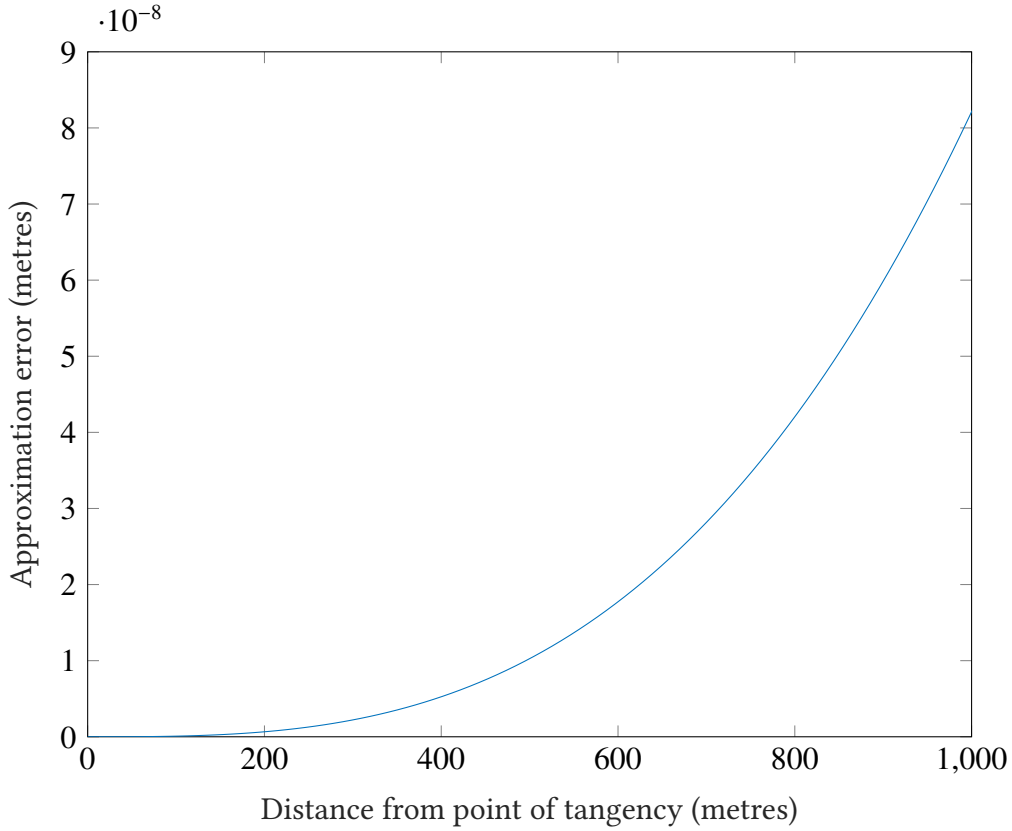


Figure 4.2: Affine transformation error plotted up to 1 km.

that using an affine transformation matrix yields far to small an error for it to have any meaningful affect on the transformation of measurements made inside a single building.

It is worth noting that two major assumptions have been made in this estimation. The first is that the transformation plane is a tangent plane while in reality it is actually a plane that cuts the sphere. This is because the three reference points do not coincide. However, compared to the radius of the earth, the distance between the reference points is so small that the points are assumed to converge. The second assumption is that the earth is a perfect sphere with the radius 63 710 088 m. This radius is actually the mean radius of the earth which means that the error estimation above is actually an estimation of the average case. However, this fact is assumed to have a negligible impact on short distances.

Chapter 5
Area Coverage

Area Coverage is another well-researched topic in mobile robotics, which fairly recently has made its way into households in the shape of robot vacuum cleaners and lawnmowers. In this chapter two existing algorithms: spanning tree covering, and distance transform path, are evaluated to see which can best be applied to this thesis.

5.1 Motivation

Area Coverage can be seen as a generalisation of the well known *Travelling Salesman Problem (TSP)* and is therefore NP-Hard, i.e. solving the optimisation cannot (currently) be done in polynomial time [26]. Therefore, an algorithm which finds an approximate solution is necessary. The criteria for selecting an algorithm are: it should finish in a finite time, cover as much of the area as possible, and return a path which can be completed as fast as possible.

The robots characteristics have to be taken into consideration since less distance travelled does not necessarily mean that the time required to complete the path is less. The *Turtlebot 2* tends to handle tight curves badly by slowing down drastically. Therefore, a solution which requires as few tight curves as possible would be preferable. The area coverage part of this thesis is used to solve the **UC-Map**.

5.2 Adaptive Monte Carlo Localisation

Chapter 3 covered how it is possible to create a map using SLAM algorithms in unknown environments. But in the event that the environment has already been mapped, and the map exists (**UC-Map**), it is preferable to use the existing map rather than creating a new one. The advantages are: better planning possibilities and better completion-time estimates. This is where Adaptive Monte Carlo Localisation (AMCL), a localisation

algorithm, comes in. AMCL shares some similarities with Gmapping in that they are both based on particle filters. However, instead of collecting sensor data and building a map, AMCL uses an existing map and identifies the robots position relative to it.

5.3 Global Planner

The coverage algorithms below, all produce a path as an output, but because the navigation stack in ROS is built to only accept a single goal as input (see Section 1.2.1), a custom solution was needed in order to combine the two.

A global planner called *remote_global_planner* was implemented. The planner accepts a pre-generated path, and ignores the *goal* input usually sent to the navigation stack. The path consists of a series of way-points, which get forwarded to the local planner for execution. This approach, as opposed to implementing the coverage algorithms straight into the planner, was chosen because of Combain's requirement to offload as much of the processing as possible (see Section 1.1.1). Having the planner accept remote plans opens up the possibility of calculating these paths on a remote server or even in advance.

One problem that was discovered was the fact that the local planner is fairly bad at following way-points, and prefers to ignore them if it can reach the goal faster by doing so. To avoid this issue, the remote_global_planner only sends a few way-points at a time, leaving the local planner without a path to skip to. The disadvantage with this method is that the robot might slow down or even stop moving for short moments while the local planner calculates a new path to the next way-points.

To allow the robot to adapt to dynamic changes in the world, the remote global planner uses one of the existing single-goal algorithms provided by ROS , to plan the path to the next way-point. Obstacles such as people walking by or other things not present when the map was generated is therefore avoided by the robot.

5.4 Execution Time Estimation

To be able to compare the algorithms below, a formula for modelling the execution time of a path was needed. By measuring how much time a path takes to execute if it contains turns, and then measure the same path without turns, the average impact that each turn has can be calculated. While performing these measurements it was found that $< 180^{\circ}$ turns actually improve the execution time, i.e. it takes less time to complete a path with these turns. This is a consequence of how the local planner tends to take shortcuts, as previously mentioned in Section 5.3. The resulting formula, valid for cell sizes around

0.5 m, is:

$$\begin{aligned} \text{Time} &= \frac{L}{0.5} - 0.36T_{45} - 0.14T_{90} - 2.8T_{180} + 5.3T_{max}, \\ \text{where Time} &= \text{predicted execution time in seconds,} \\ T_{45} &= \text{amount of turns where angle } \leq 45^\circ, \\ T_{90} &= \text{amount of turns where } 45^\circ < \text{angle} \leq 90^\circ, \\ T_{180} &= \text{amount of turns where } 90^\circ < \text{angle} < 180^\circ, \\ T_{max} &= \text{amount of turns where angle } = 180^\circ. \end{aligned}$$

5.5 Occupancy Grid

An occupancy grid is a way to represent a map as a discrete grid. Internally, ROS uses a probability occupancy grid, i.e. each cell has a value between 0–100 to represent the probability of the cell being occupied. However, not all ROS plug-ins take advantage of the entire range, and instead divide it into three parts: free, unknown and occupied. The *map_server* plug-in, which is used to save a map to disk, exhibits this exact behaviour. Some figures in this thesis will therefore have three colours, while some will have the entire range from black to white.

Since an occupancy grid stores the world in a grid map of a specific resolution, it suffers from the traditional issues of approximate representations. Namely lack of details below the specified resolution, which makes it impossible to deduct if whole cells are occupied or just some parts of them.

5.5.1 Downscaling

The occupancy grid is an approximate representation of reality using cells with sizes equal to the resolution. Because both algorithms presented below use occupancy grids as input, it would be possible to use the maps produced during the SLAM sessions (see Chapter 3) directly. However, these maps have a resolution of around 0.05 m – 0.1 m which, without any pre-processing, would result in a coverage of the same magnitude as the resolution, i.e. the robot would drive over every 0.05 m.

To be able to choose how detailed the area coverage should be, the occupancy grid is first down-scaled before used as input for the coverage algorithms. This is accomplished by using a method called box sampling. Simply put, if a coverage of d metres is desired, the process looks as follows:

1. Overlay the original occupancy grid O with another grid G of size d.
2. For each cell in G, calculate the average values of the corresponding cells in O.
3. Assign the average value to each cell in G and treat G as a new occupancy grid.

To further simplify the process, each cell in G is treated as occupied if its value is larger than v_{thresh}, and unoccupied otherwise. To guarantee a viable solution, v_{thresh} is set to $v_{free} + 1$, meaning that even a single occupied cell in O results in the entire cell in G being

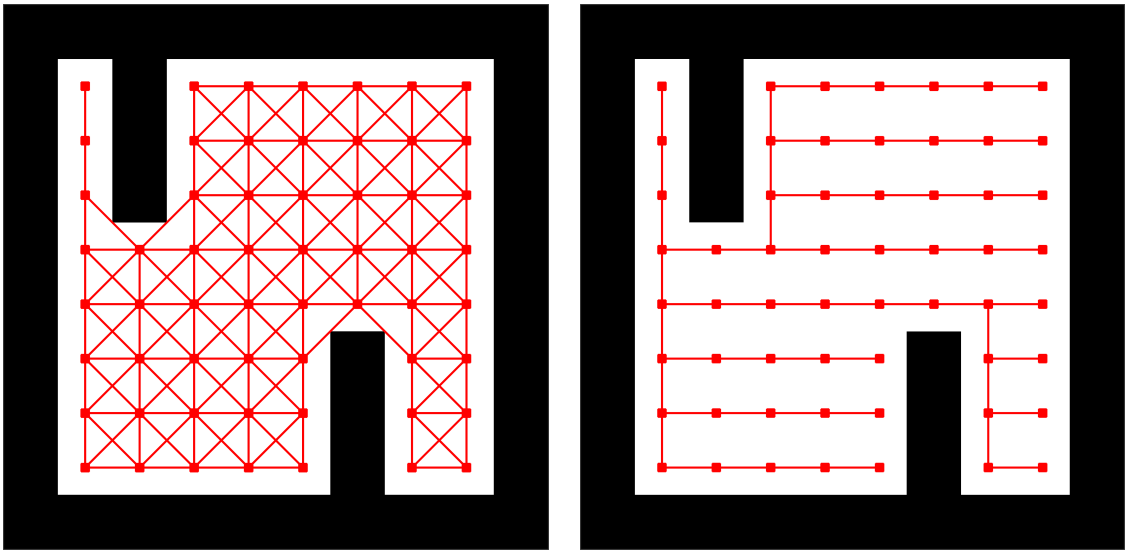


Figure 5.1: To the left is a visualisation of a graph representation of a 10 m × 10 m map. On the right is a visualisation of a spanning tree of the same graph.

marked as occupied. By using G as an input to the algorithms below, complete coverage of every, completely empty, d metre is guaranteed.

5.6 Spanning Tree Covering

The paper by Gabriely and Rimon [26] describes a novel method of using a spanning tree for area coverage. This can be accomplished by interpreting all of the free cells in the occupancy grid as vertices in a graph. By adding edges between all vertices that represent adjacent cells in the occupancy grid, a graph like the one in Figure 5.1 can be obtained. A spanning tree, which is defined as a sub-graph that is a tree and contains all the vertices with the minimum number of edges, can then be constructed by using e.g. Prim's algorithm [27]. An example of a spanning tree can also be seen in Figure 5.1.

After a spanning tree has been constructed, coverage can be obtained by: dividing all cells into four sub-cells, selecting any cell as the initial starting position and then walking along the tree until the initial cell is encountered again. In other words, imagine placing your left hand anywhere on the tree and simply walking straight, making sure to never stop touching the tree. Eventually you will have circumnavigated the entire tree in a counterclockwise direction and end up where you started. This also means that you will have covered the entire map. An example can be seen in Figure 5.2, which is the path generated by the spanning tree in Figure 5.1.

The paper [26] actually presents three different algorithms. However in this thesis, only the offline variant called Offline STC was considered, as it is the only one which uses known maps.

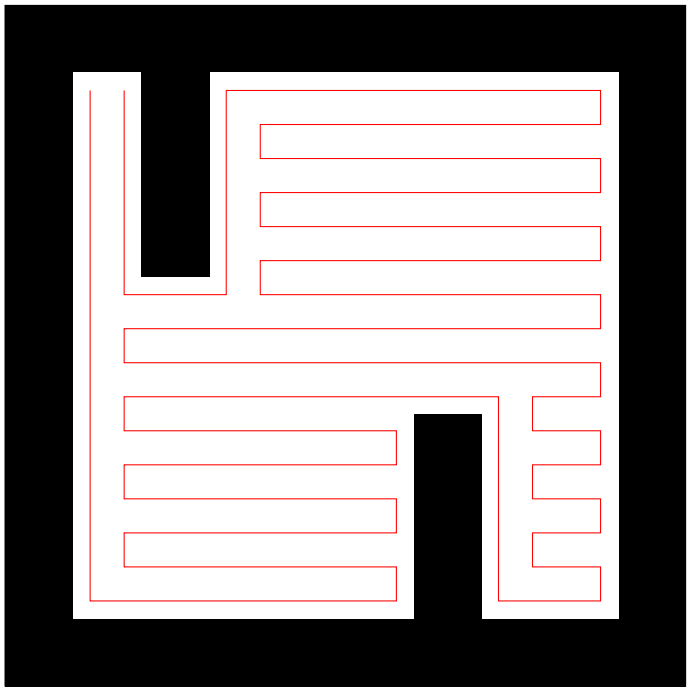


Figure 5.2: Generated path from the spanning tree in Figure 5.1, notice how the path circumnavigates the spanning tree.

5.7 Distance Transform Path Planning

The paper by Zelinsky et al. [28] presents "a solution to the problem of complete coverage based upon an extension to the distance transform path planning methodology". The algorithm is explained in further detail below.

5.7.1 Distance Transform

The distance transform T_{dist} is a matrix where every element describes the distance to a specific element v_{goal}. In this thesis, T_{dist} is calculated using the Breath First Search (BFS) algorithm. An illustration of a distance transform calculated using BFS can be seen in Figure 5.3. The coverage path is then created by moving along the path of steepest ascent. This means that the path moves away from the goal while keeping track of the cells it has already visited.

[28] mentions that "the robot only moves into a grid cell which is closer to the goal if it has visited all the neighbouring cells which lie further away from the goal". This thesis uses an alternative approach by introducing a backtracking method. The backtracking is activated when the robot ends up at a dead-end, formed either by obstacles or cells which have already been visited. To perform the backtracking, the algorithm simply walks backwards along the path until it reaches a neighbour which has not been visited. This neighbour is then added immediately after the dead-end cell, ignoring the backwards walk. This technically means that the path usually ends up across obstacles, but this is not a problem since the global planner will automatically adapt using the single-goal

algorithm.

5.7.2 Path Transform

The paper [28] also describes a transform for planning called path transform T_{path}. It builds upon the distance transform T_{dist} by combining it with a transform called obstacle transform T_{obst}. The obstacle transform can be described as a matrix where every element describes the minimum distance to the closest obstacle. Every element c in T_{path} is then calculated using the function:

$$PT(c) = \min_{p \in P} \left(\text{length}(p) + \sum_{c_i \in p} \alpha \text{obstacle}(c_i) \right). \tag{5.1}$$

According to the results in [28] T_{path} results in paths which tend to follow walls, resulting in overall straighter paths.

In this thesis, the path transform was implemented by modifying the distance transform algorithm to use Dijkstra's algorithm with $PT(c)$ as input. However, while the modification did add complexity to the algorithm, the results did not improve nearly as much. In fact, while in small maps like in Figure 5.3 the path had a tendency to follow walls, there was no improvement at all in bigger ones like in Figure 5.5. The reason why is not clear, maybe the map was simply too uneven, or maybe the implementation was erroneous in some way. Because of this, the path transform was not used in the final evaluation of this chapter.

5.8 Evaluation

The results from both algorithms are presented in Table 5.1, from which it is possible to see that spanning tree covering resulted in a drastically shorter distance travelled, with a more than 15% improvement for the small 20x20 example and a 60% improvement for *M-huset*. However since the distance transform path actually covered a larger area in *M-huset* this is not a completely fair comparison. One of the reasons the distance transform path algorithm performed badly was due to the need to backtrace whenever it got stuck, which might have been improved with a more efficient method such as finding the closest uncovered area and continuing.

Looking at the output of the spanning tree covering algorithm it is clearly visible that it resulted in a sub-optimal path due to a high number of necessary turns in the top and bottom corridors which was caused by the map being at a slight rotation. After rotating the map by 13° the amount of turns was decreased for both algorithms. The result was particularly noticeable on STC since it resulted in a 34 % decrease of the number of turns.

To verify the accuracy of the execution time model, a smaller area of *M-huset* was tested. The difference between the planned and the actual travelled path, together with the corner cutting, can be seen in Figure 5.6. The model predicted an execution time of 9 min 53 s for the distance transform path and 6 min 25 s for the spanning tree covering path. The actual execution times were 8 min and 5 min 25 s respectively, meaning that the prediction was off by a bit more than 20 % for both paths.

S	19			13	13	13	13	13	13	13	13	13	13	13	13
18	18			12	12	12	12	12	12	12	12	12	12	12	12
17	17			11	11	11	11	11	11	11	11	11	11	11	11
16	16			11	10	10	10	10	10	10	10	10	10	10	10
15	15			11	10	9	9	9	9	9	9	9	9	9	9
15	14			11	10	9	8	8	8	8	8	8	8	8	8
15	14	13	12	11	10	9	8	7	7	7	7	7	7	7	7
15	14	13	12	11	10	9	8	7	6	6	6	6	6	6	6
15	14	13	12	11	10	9	8	7	6	5	5	5	5	5	5
15	14	13	12	11	10	9	8	7	6	5	4	4	4	4	4
15	14	13	12	11	10	9	8	7	6			3	3	3	3
15	14	13	12	11	10	9	8	7	7			3	2	2	2
15	14	13	12	11	10	9	8	8	8			3	2	1	1
15	14	13	12	11	10	9	9	9	9			3	2	1	G
15	14	13	12	11	10	10	10	10	10			3	2	1	1
15	14	13	12	11	11	11	11	11	11			3	2	2	2

Figure 5.3: Illustration of a distance transform matrix, black squares represent obstacles, and the numbers are the distances to the goal. Notice, that diagonal neighbours are allowed.

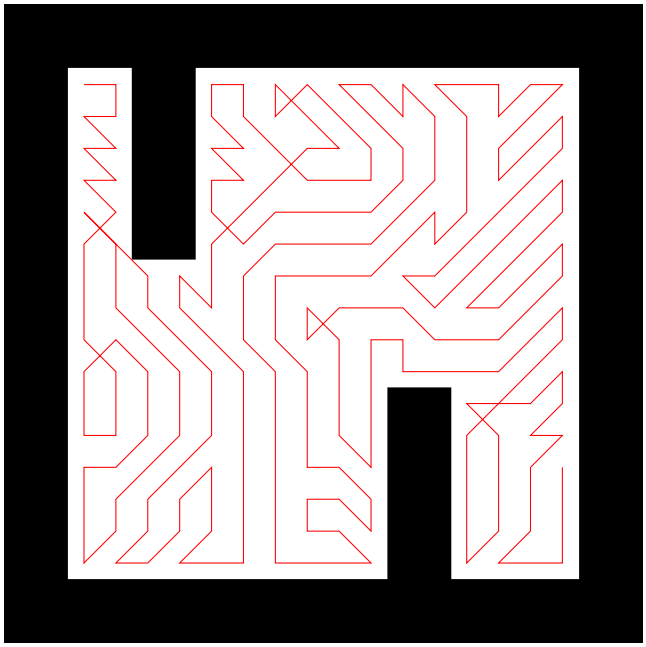


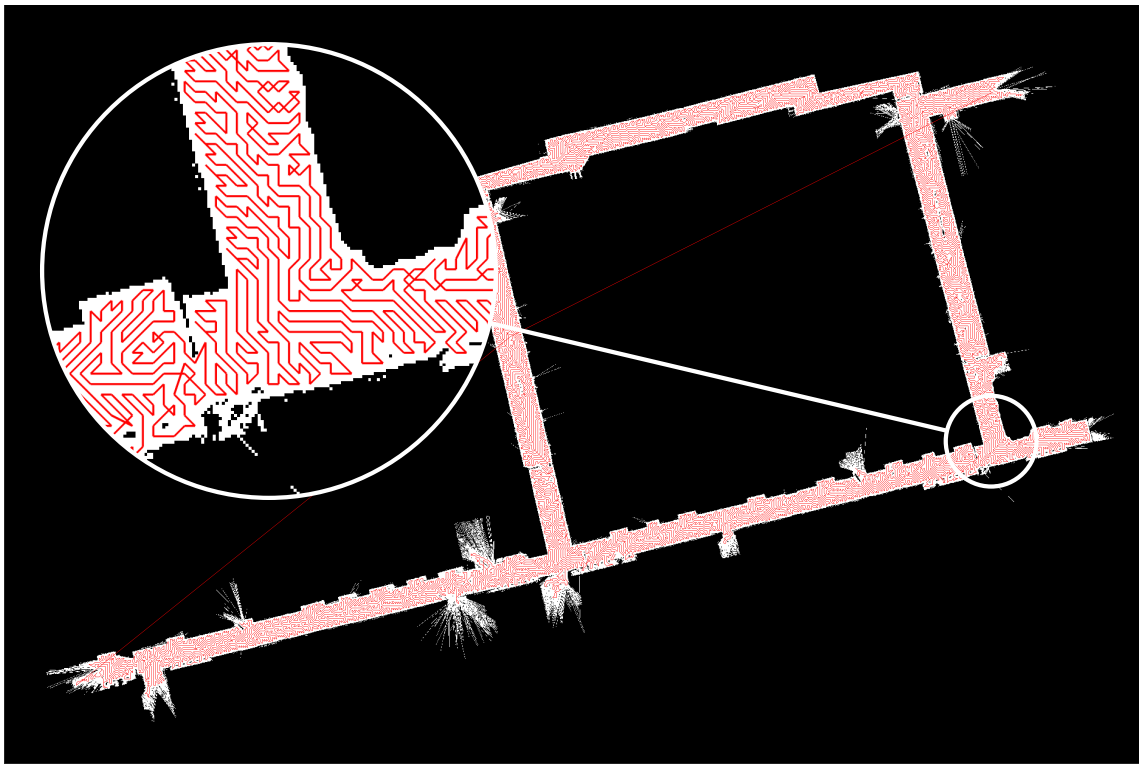
Figure 5.4: Illustration of the Planned Path from DT in Figure 5.3. The start position is in the top left corner. The path always pick the neighbouring node with highest cost.

Even though the prediction was off by 20 %, the model still seemed to provide an accurate difference between the two algorithms. However, using the formula to predict the execution times for the paths in Table 5.1 was an entirely different story. While the paths generated by the spanning tree algorithm for M-huset were believably predicted at 55 min, the prediction for the distance transform path turned out to be negative (−90 min). Because of this, the prediction model could unfortunately not be used for any type of comparison between these algorithms. The paths generated by the distance transform algorithm were simply to complex.

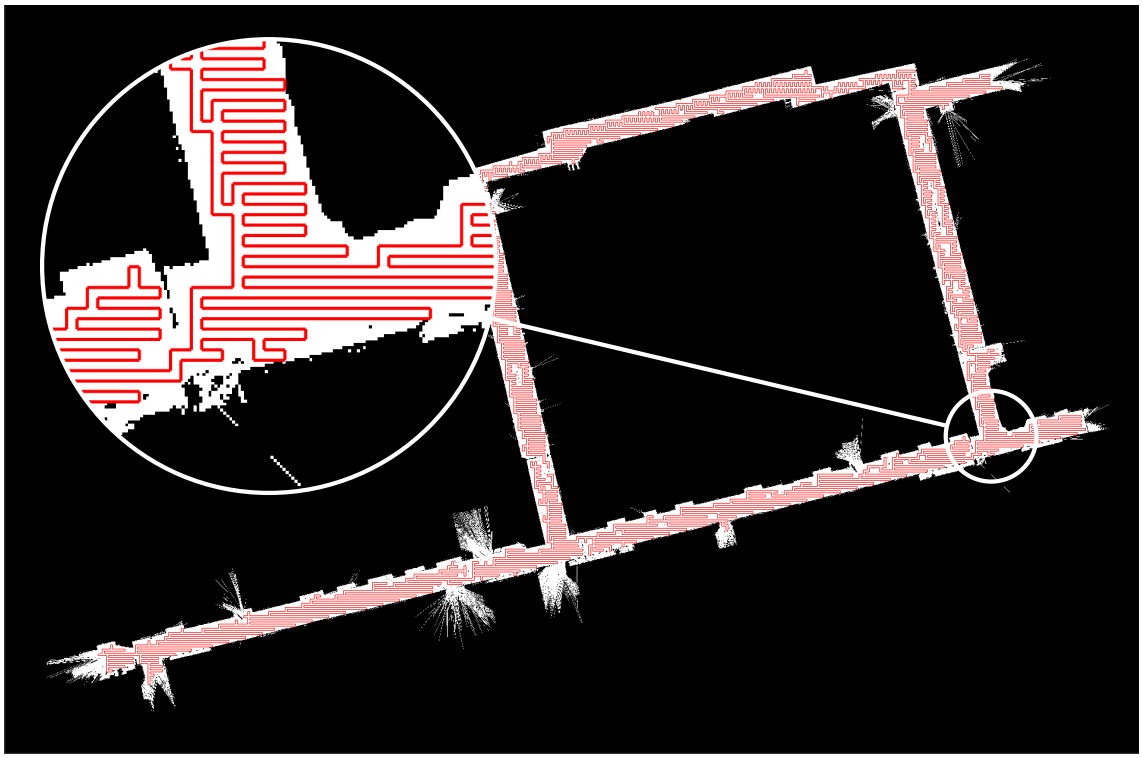
5.9 Conclusion

In this chapter the purpose has been to present and evaluate some popular algorithms for area coverage to see which provide the best results for **UC-Map**. As was mentioned in the beginning of this chapter, two criteria had been chosen for the evaluation of these algorithms. The primary was complete map coverage and the secondary was execution time.

As can be seen in Figure 5.6, both algorithms covered the majority of the area. However, it is quite clear that the distance transform algorithm provided a better coverage than the spanning tree algorithm. Regarding the second criteria, the spanning tree algorithm provided both better execution times during testing, and shorter paths. The conclusion in this chapter is therefore that the recommendation depends heavily on the specific use-case. Even so, the authors would like to recommend the spanning tree algorithm, as it is felt that the difference in path length is worth sacrificing some coverage over.



(a) Distance transform path

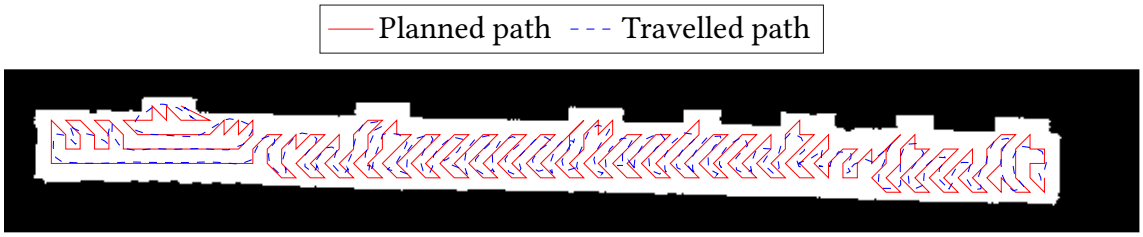


(b) Spanning tree covering

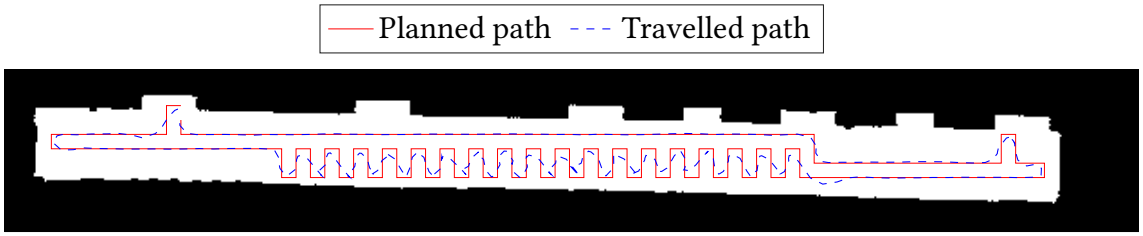
Figure 5.5: Visualisation of the full path in M-huset

Table 5.1: Statistics for the two area covering algorithms for different maps, using a 0.35 m cell size.

Map	Alg.	Coverage	Dist. [m]	Number of turns						
				Total	<	45°		90°		180°
20 × 20	**STC**	232 / 232	231	48	0	0	0	48	0	0
	DTP	232 / 232	272	144	0	57	0	29	61	1
M-huset	**STC**	10376 / 10384	1816	2317	0	0	0	2317	0	0
	DTP	12457 / 12457	2738	8556	48	3691	16	1233	3487	81
Rotated M-huset	**STC**	10560 / 10568	1848	1525	0	0	0	1525	0	0
	DTP	12487 / 12487	2703	7491	40	3168	17	1022	3208	36



(a) Distance transform path



(b) Spanning tree covering

Figure 5.6: Comparision betweeen the planned and actually travelled paths of the bottom left cooridor in M-huset.

However, it should be noted that the spanning tree algorithm, in the worst case scenario, would only provide a 50 % coverage.

Chapter 6
Exploration

This chapter covers autonomous exploration of an unknown map. A few of the existing algorithms are presented and evaluated to recommend the one which provides the best result for **UC-NoMap**.

6.1 Motivation

The exploration algorithm's main purpose is to provide the autonomy part of **UC-NoMap**. The algorithm takes a time consuming process and automates it by letting the robot explore the layout of a building by itself.

6.2 Frontier-based Exploration

The paper by Yamauchi [29] describes a novel method of using a frontier-based approach for exploration. According to the paper, "To gain the most new information about the world, move to the boundary between open space and uncharted territory". A frontier is hence defined as the edge between open space and the unknown. The paper does mention that "A Zeno-like Paradox where the new information contributed by each new frontier decreases geometrically is theoretically possible (though highly unlikely)", but also says that even in such cases "the map will become arbitrary accurate in a finite amount of time."

6.3 Selection of Frontier

One of the ways to improve the performance of the exploration is to ensure the selection of good frontiers. The paper by Holz et al. [30] evaluates different strategies for

selecting frontiers. It shows that selecting the closest frontier, results in a fairly optimal solution, though a slight improvement is possible by using *repetitive re-checking*[1] and *map segmentation*[2].

6.4 Existing Implementations

There currently exist multiple implementations of the frontier exploration for ROS. For this thesis it was decided to evaluate three different implementations to see if there exists a difference in the performance.

6.4.1 Frontier Exploration

The paper by Yamauchi [29] has an implementation in ROS appropriately named *frontier exploration* [31]. It requires a working navigation stack (see Section 1.2.1) and operates by sending goals to the global planner. A new goal to a new frontier is sent every time the navigation stack reports that the previous goal has been reached, or at regular intervals, meaning that the algorithm supports *repetitive re-checking*. The algorithm always chooses the closest frontier as found by BFS.

6.4.2 Hector Exploration

Hector exploration is based on the paper by Wirth and Pellenz [20] which builds on and extends the frontier exploration algorithm introduced in Yamauchi [29]. A difference between frontier exploration and hector exploration is how they navigate to frontiers. While frontier exploration takes advantage of an existing navigation stack, hector exploration replaces the global planner, allowing it to not only set goals, but also decide how to get there. Another consequence of hector exploration acting as a global planner, is that the navigation stack continuously asks it to re-evaluate its path. And because it might change its goal during any of these re-evaluations, the behaviour can be identified as *repetitive re-checking*.

Another difference is that, unlike frontier exploration which uses BFS to find the closest frontier, hector exploration employs an *exploration transform*. This transform, much like the path transform in Section 5.7, uses a weighted sum of the distance transform which describes the distance, and the obstacle transform which describes the discomfort of moving close to walls. The exploration transform is used for both path planning and picking the next frontier.

6.4.3 Nav2d Exploration

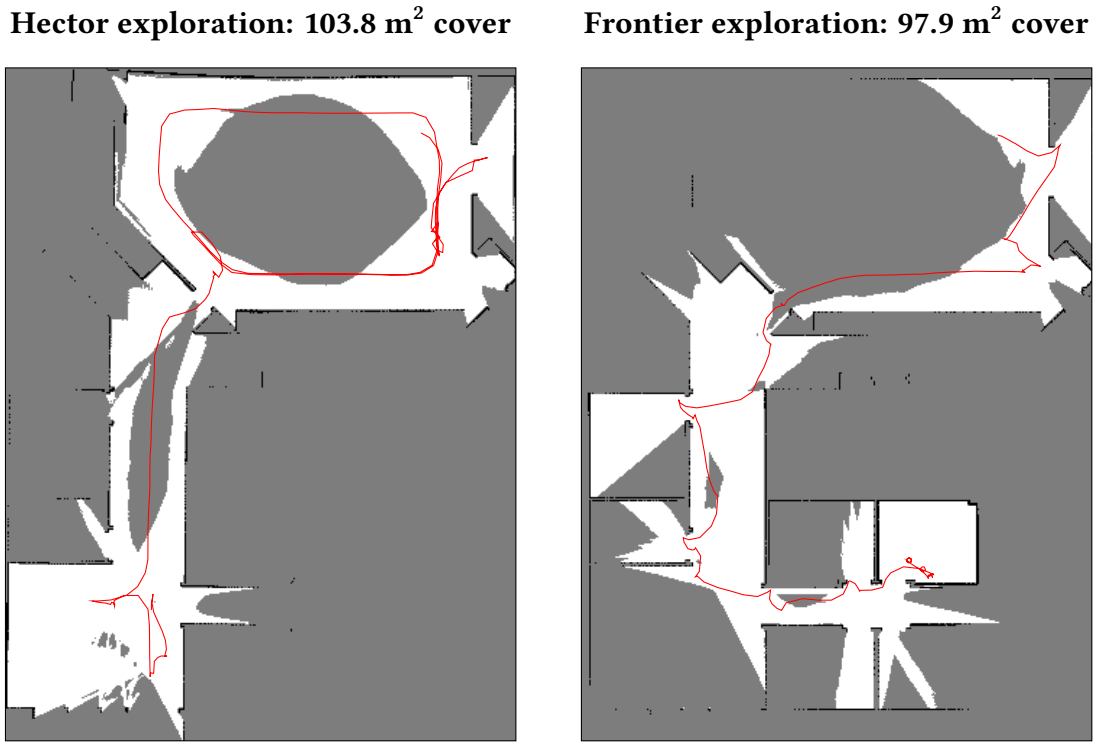
Nav2d exploration is another frontier-based exploration plug-in and is unique in its offering of multi-robot exploration capabilities. Its packages, however, target an old version of ROS and the authors were unfortunately not able to get it working.

[1] Repetitively check if there exists a closer frontier, since new frontiers can be discovered while moving.
[2] Dividing the map into segments and finish the segment before moving to the next segment.

6.5 Evaluation and Conclusion

The purpose of this chapter has been to present and evaluate which of the most popular ROS plugins for exploration that provide the best results for **UC-NoMap**. Unlike Chapter 3 and Chapter 5 which had widely differing algorithms, the ones compared in this chapter have all been based on the same frontier exploration method. Even so, in Figure 6.1 it can be seen how the behaviours of the algorithms have been distinctly different. Hector tended to follow walls and discover the room or corridor that it was currently in, before going to a new one. Frontier on the other hand, simply went for the nearest frontier, and resulted in a behaviour that never finished an entire room before going to the next one. Another downside with frontier was that it tended to drive very close to walls, which sometimes lead to the robot getting stuck. This behaviour was not present in hector.

After all of the information above was taken into consideration, hector, which was basically an extended version of frontier, was chosen as the recommended algorithm.



(a) Hector exploration.

(b) Frontier exploration.

Figure 6.1: A side by side comparison of how Hector exploration and Frontier exploration explore. Notice how Hector exploration tends to follow walls and discover entire rooms before continuing. Notice also how Hector has driven over the same place multiple times.

Chapter 7
Wi-Fi Scanning

This chapter goes over how the Wi-Fi measurements were collected, and describes the pre-processing which had to be applied prior to submitting the data to Combain CPS.

7.1 Introduction

Wi-Fi access points makes themselves visible by periodically sending out beacon frames. This in turns allows other devices, such as smartphones, to connect and gain internet access. Since these beacon frames are constantly broadcasted by all access points, without requiring any modifications to either hardware or software, they are perfect for regular data collection.

According to [32] the following equation can be used to describe the relationship between the RSS and the distance d:

$$RSS = C - 10n \log_{10} d, \tag{7.1}$$

where n is the *path loss factor* and C is the RSS at 1 m. The path loss factor depends heavily upon the environment and normally varies between 2 and 6 [33].

Figure 7.1 shows a heatmap of a Wi-FI access point.

7.2 Measuring

Most of the access points encountered during the duration of this thesis seemed to have a beacon-rate of 10 Hz. According to Combain, recording all those beacons would have been too much data. Instead, it was decided to temporarily record all frames, and then throw away all but the latest for each access points each second. This allowed better support for access points with lower beacon-rates, and also better handling of packet loss.

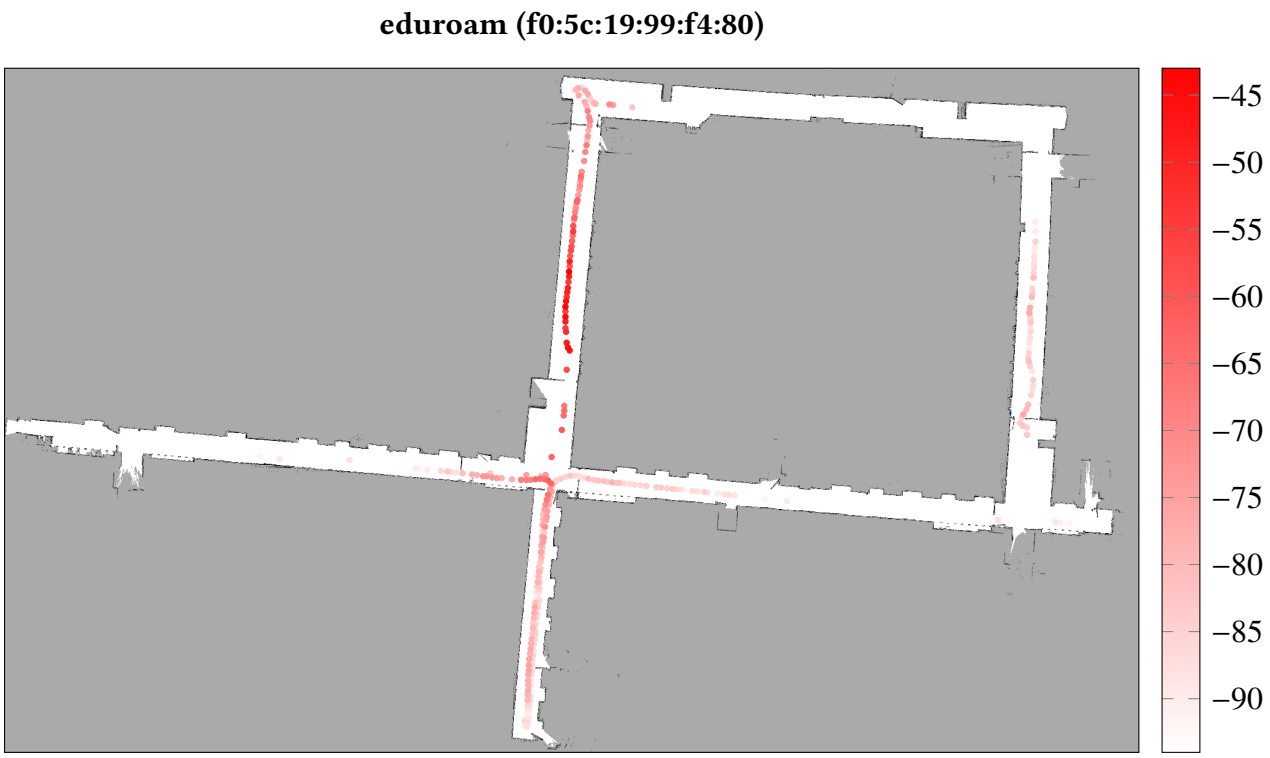


Figure 7.1: Visualization of the RSS for one of the access point for the network *eduroam*. A high RSS means better signal strength. The access point position is in a room next to the left corridor. Note that the network is most likely visible in the corridor to the right due to the free line of sight through windows.

The groups of beacon frames were also timestamped to enable the matching of groups to positions. An example of a beacon frame group can be see in Table 7.1.

7.3 Processing

As previously described in Chapter 4, the local coordinates in ROS were represented using a local cartesian coordinate system. Before they could be used further, they had to be converted to the WGS84 system. Furthermore, to allow Combain to apply further processing, an *age* field was also used to describe when the beacon frames were received relative to the timestamp.

The end result had the following format:

```
measurements: [
    {
        time: unixTimestamp,
        wifi: [
            { bssid, rss, ssid, age },
            ...
```

Table 7.1: List of Wi-Fi access points from which beacon frames were received during a single measurement. The RSS is measured in dBm.

BSSID	RSS	ESSID
C8:D7:19:9A:43:92	-89	Airsonett_LAB
52:9F:27:19:6B:B2	-43	Worxmate 4G
52:9F:27:19:6B:B1	-41	Combain Guest
50:9F:27:19:6B:B0	-42	Combain 4G
52:9F:27:19:6B:B3	-42	Worxmate Guest
C8:BE:19:62:B0:42	-79	Timeline
F0:99:BF:09:63:98	-78	KP Wi-Fi Network
A8:D0:E5:37:45:C0	-90	eduroam
A8:D0:E5:37:45:C2	-89	LND_INTRA
88:75:56:6E:1D:70	-90	Client
88:75:56:6E:1D:71	-88	eduroam
88:75:56:6E:1D:72	-90	Region Skane Publikt
D0:17:C2:B3:49:73	-27	Worxmate_GUEST
D0:17:C2:B3:49:71	-27	Worxmate
D0:17:C2:B3:49:70	-29	Combain
D0:17:C2:B3:49:72	-36	Combain_GUEST

```
            ]
        }
    ]
```

At the end of each measurement, a trajectory is collected and saved. As previously stated, this was then used to map the measurements to their corresponding location on the trajectory using the timestamp.

7.4 Exporting

Finally the processed measurements were exported to CPS using Combains existing internal Application Programming Interface (API). Combain then performed a SLAM optimisation, in which the positions of access points were calculated. The result of the optimisation was then saved and used to improve their existing models, which in turn would improve future queries.

The data could also be exported as Comma-Separated Values (CSV) files, in an internal format specified by Combain, which allowed for easier debugging.

Chapter 8
Fleet Management Platform

One of the important requirements of this project was to offload as much of the main processing as possible to a remote server. Combain also expressed a wish for a user friendly interface. In this chapter the platform which was built during this thesis is presented, and the current functionality is described.

8.1 Architecture

In order to assure the maintainability of the platform and allow for easier integration into Combain's existing projects, it was decided that the platform should be built with a similar architecture as their existing projects. The platform was therefore implemented as a multi-page web application, using Node.js and PostgreSQL. ROS has an existing plug-in called *rosbridge*, which allows for easy integration using WebSockets and in turn enables communication between Node.js and ROS.

All data was stored in a relational database, since both authors had prior experience using it for similar projects.

In order to allow for other projects to easily connect to ROS in a similar manner, it was decided to separate all ROS-related code into stand-alone isolated modules. In the future, these could easily be extracted and integrated into other projects.

8.2 Workflow

A lot of effort went into making an intuitive user interface, to make it easy-to-use even for people with limited technical experience or domain knowledge. The primary focus was on simplifying and streamlining the interface for executing the two use cases **UC-Map** and **UC-NoMap**. To achieve this, an entity-relationship model inspired by the real world was used. The following entities appear:

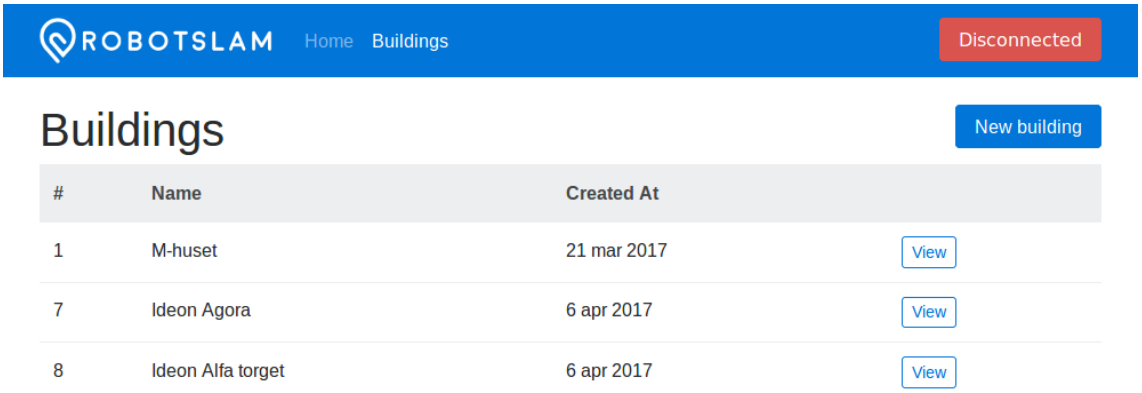


Figure 8.1: Screenshot of the buildings view.

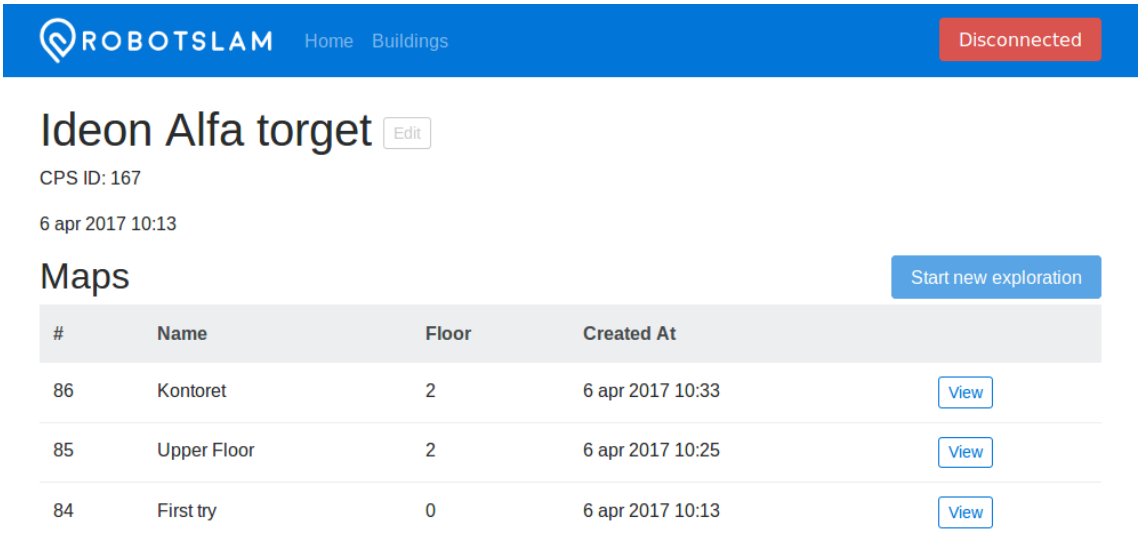


Figure 8.2: Screenshot of the building details view.

Building In the highest layer, the system is organised in buildings. Each building represents a specific location in the real world. To facilitate connection with CPS each building is assigned a *CPS ID* which refers to a corresponding reference in Combains system. In the future the interface for entering the *CPS ID* could be simplified by allowing bi-directional communication between this platform and CPS, however for the scope of this thesis, it was decided to minimise the impact on existing systems. The interface for selecting buildings can be seen in Figure 8.1.

Maps Each building can be explored multiple times, each time resulting in a different map. The maps are assigned to specific floors, of which they can cover either the complete floor or a part of it. The maps also contain meta-data such as the resolution and the origin of the robot i.e. where the robot started its exploration.

Measurements Each map further contains at least one measurement. A measurement is a set of collected RSS, mapped to positions in the map. The positions are converted to WGS84 when they are exported, this process can be read about in Section 8.3.

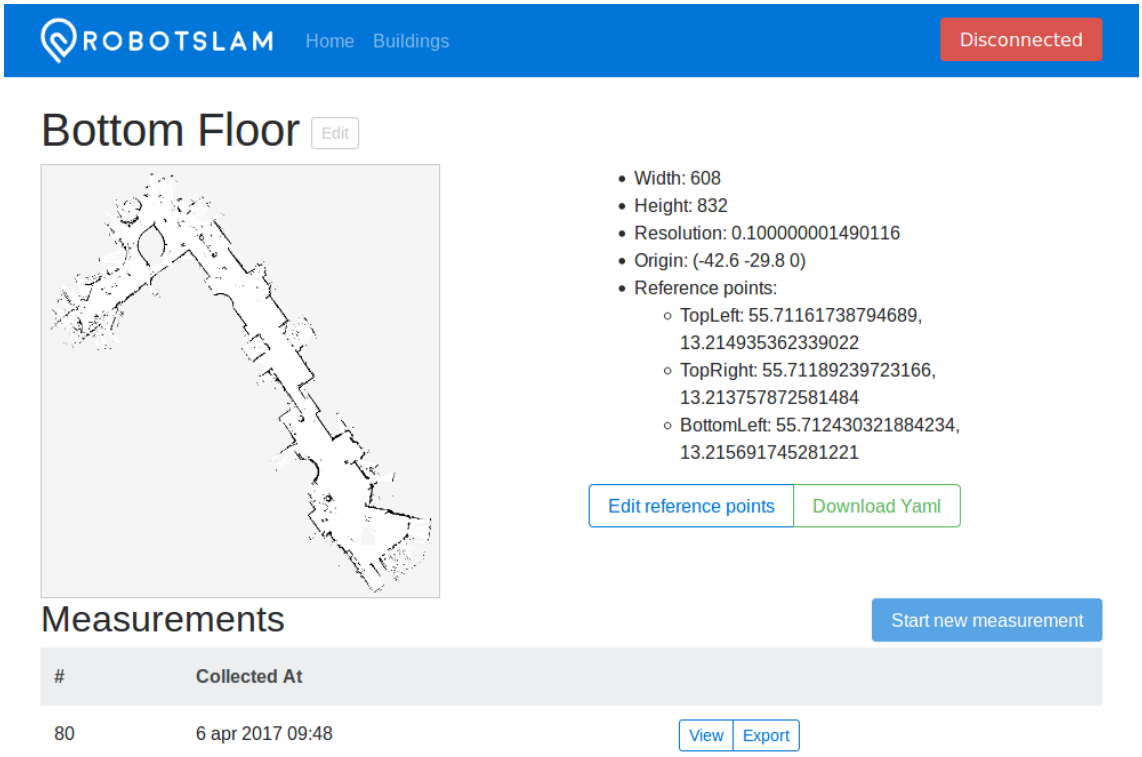


Figure 8.3: Screenshot of the map details view.

8.2.1 Exploring Building

For **UC-NoMap** a typical user-flow looks like the following:

1. Create or select an existing building which should be mapped. Figure 8.1 shows a screenshot of the building list.
2. Begin a new exploration, by selecting the floor, which is to be explored.
3. Wait for the exploration to be completed, and stop the exploration when sufficient area is explored.
4. Edit reference points, by dragging the overlaid map to its real position.
5. Export or visualise data.

8.2.2 Collecting Measurement in an Explored Building

For **UC-Map** a typical user-flow looks like the following:

1. Select the building which should be mapped. (See Figure 8.1)
2. Select a measurement (See Figure 8.2)
3. Start measurement.

4. Stop measurement.
5. Edit reference points.
6. Export or visualise data.

8.3 Placing the Map in the Real-World

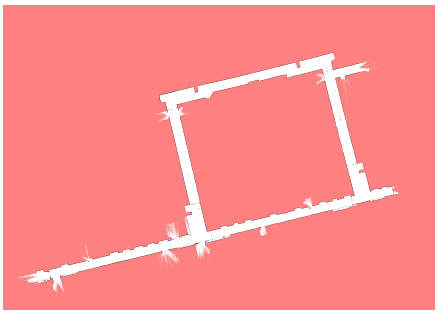
As mention in Chapter 4, the collected measurements were stored in a cartesian coordinate system. In order to export the data to CPS they first had to be converted to the WGS84 system, which was done by using an affine transformation matrix. This matrix was created by placing three pairs of reference points in both systems.

Specifying pairs of reference points can be a difficult and time consuming operation to perform manually. Therefore, a graphical tool was added to the web platform. The tool uses Leaflet, an open-source JavaScript library for interactive maps, to overlay an image on top of an OpenStreetMaps layer. For further visual aid, an indoor map layer from Micello is also added on top of the OpenStreetMaps layer. The resulting map of an exploration session is then placed on top everything else, and translated by the help of three graphical reference points in real-time using CSS. An example of the process can be seen in Figure 8.4.

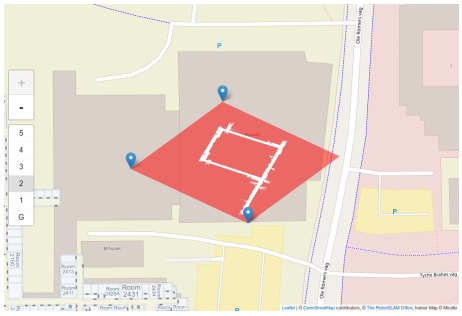
To ensure that the correct data-set is exported to CPS, another tool was built to visualise the trajectory of the scan session. By using the reference points along with the affine transformation matrix defined in the previous step, the trajectory can be converted into WGS84. The resulting path is then overlaid on top of the interactive Leaflet map. Figure 8.5 shows the complete path travelled during a scan session.

8.4 Conclusion

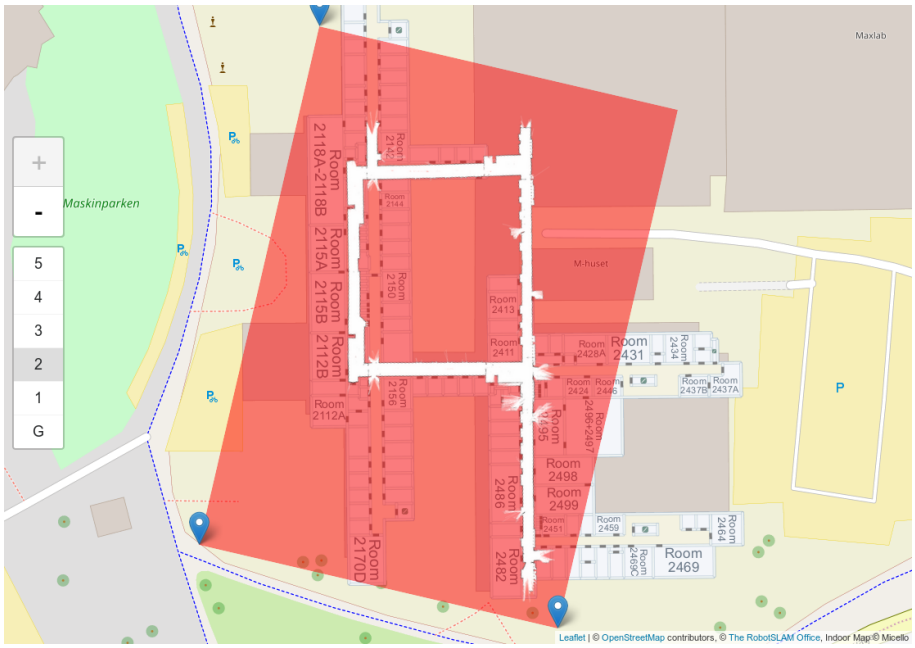
In this chapter a platform for controlling ROS based robots have been presented. The platform removes much of the manual work usually required and provides handy tools for manipulating data. By using the web interface with a carefully crafted workflow the system can be used without requiring good technical experience or extensive domain knowledge.



(a) Gmapping SLAM result.



(b) Before fitting.



(c) After fitting. Notice how the bottom corridor is a bit too long.

Figure 8.4: Mapping with Gmapping in M-building, LTH. The background is added for clarity and is not present in the actual results.

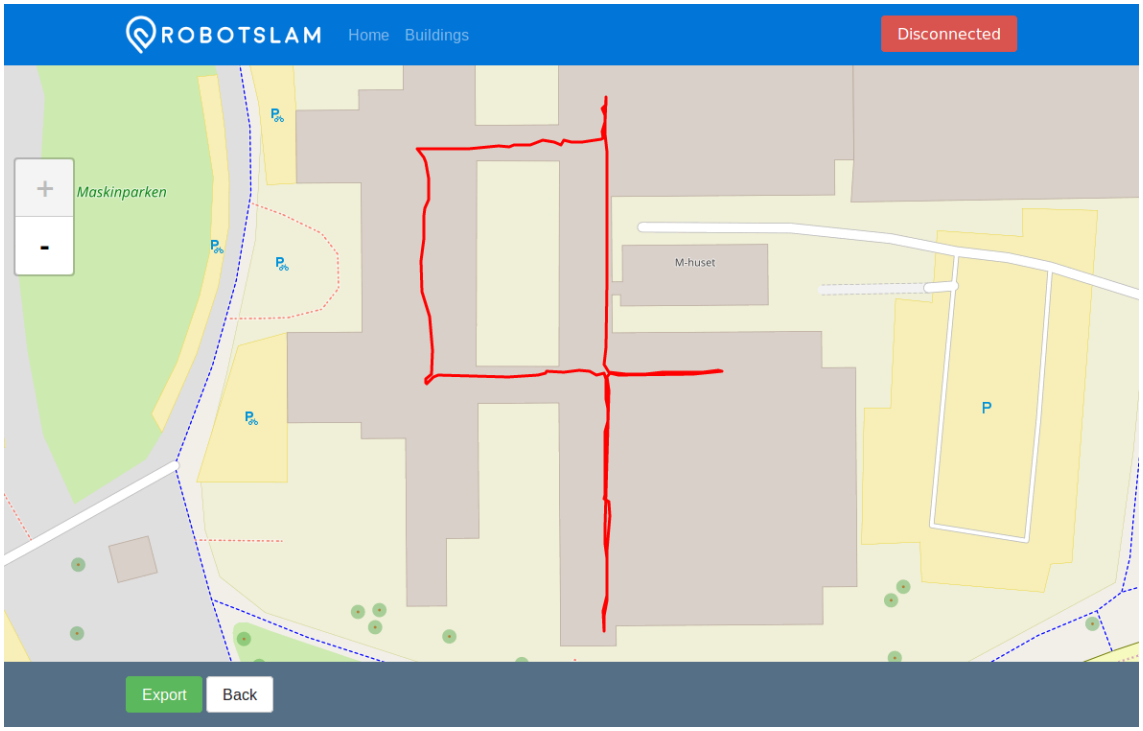


Figure 8.5: Visualisation of the SLAM Trajectory.

Chapter 9
Discussion

This chapter combines individual conclusions from previous chapters to form a complete picture. It then discusses the advantages and disadvantages of the results, and how they may be improved in future projects.

9.1 Future Work

This thesis has introduced a solid foundation built upon state-of-the-art research in autonomous vehicles and robotics. However, because of the broad scope of this thesis, there are several areas which future projects could dive deeper into.

9.1.1 Alternative Data Sources

In this thesis, the main focus was on collecting measurements of Wi-Fi RSS. However, the resulting system is not limited to this. Bluetooth beacons e.g. have some advantages compared to Wi-Fi access points: they consume less power, can be calibrated to increase the accuracy [1] and they support iOS devices. Collecting Bluetooth beacons could therefore be a natural next step.

Another interesting direction, could be to use the platform built in this thesis, to collect completely different types of data, e.g. camera images, audio or air quality. This data could then be used for research or even for commercial applications.

9.1.2 Controller Unit

Due to performance issues, half-way through the project it was decided to abandon the idea of using a Raspberry Pi as a controller unit. Unfortunately, the solution suffered

[1] By providing TX-power information.

from several issues originating from high network-latency and low performance. The network latency meant that any action performed by the robot was based on information which was at least 120 ms old. This resulted in choppy movement and the inability to dodge obstacles in time. In an attempt to get rid of the latency, most of the processing was moved to the Raspberry Pi. This unfortunately overloaded the Raspberry Pi resulting in the processing lagging behind, and ultimately led to the same choppy movements as before. In the end, the Raspberry Pi was replaced with a more powerful notebook.

Future projects could focus on balancing the processing more intelligently between the cloud and the local controller unit. A hybrid solution might for example enable smooth navigation without sacrificing map fidelity.

9.1.3 Exploration

This thesis evaluated existing implementations of frontier-based algorithms, and presented a recommendation for an algorithm that has the ability to map entire buildings. However sometimes, like in Figure 6.1, the paths travelled by the robot were less than optimal. Looking into this issue might improve the time required to explore, possibly using better selection criteria.

9.1.4 Total Area Coverage

Area coverage is a well researched topic. During the initial research more algorithms than the ones discussed in this thesis were found, but due to time constraints, it was not possible to evaluate them. A new project which puts focus on evaluating more of the existing algorithms, or improving the current ones, is needed to get a complete picture of what is currently feasible. For example, it might be possible to improve the area coverage in spanning tree covering by using a dynamic grid size.

9.1.5 Obstacle Avoidance

In this thesis, the obstacle avoidance only uses the information gained from the LIDAR. However, ROS provides support for obstacle avoidance using other sensors, such as a 3D camera or sonar. A 3D camera could provide multiple advantages over the LIDAR, e.g. it would be able to see things at different heights which in turn would allow it to detect non-uniform obstacles and changes in elevation e.g. stairs.

9.1.6 Fleet Management Platform

The fleet management platform created in this thesis supports the most basic functionality. However, for it to be able to serve as a generic platform, there are many areas which could be improved. For example, it currently only supports a single active robot, and it is not possible to track its progress. The authors wanted to create a real-time map which shows the current exploration progress and the location of the robot, however, due to time constraints this was not possible.

9.1.7 SLAM

Additional work on the SLAM algorithms is necessary in order to better handle areas with see-trough obstacles, more specifically glass obstacles such as windows or mirrors. There are several existing papers with different solutions that could be implemented in the future. Wang and Wang [34] describes a modification to the existing Gmapping algorithm, which can effectively detect and avoid glass using only a LIDAR, while Zhang et al. [35] describes how LIDAR and sonar data can be fused to provide more accurate measurements.

9.2 Conclusions

In this thesis the most popular SLAM implementations for ROS have been presented and evaluated. Cartographer presented itself as the only algorithm which, with 100% accuracy, managed to perform a correct loop closure. Although the map was slightly more distorted than Gmapping, the improved loop closure made up for it. Cartographer was therefore the authors recommended SLAM algorithm. It should be noted however, that Gmapping was not too bad either, and it was the authors belief that much of the problems with loop closure was due to the LIDAR. A LIDAR with higher accuracy would probably improve the result of both Gmapping and Cartographer. Nonetheless, with the hardware that was available in this thesis, Cartographer is more than capable of supporting the use-case **UC-NoMap**.

Although there was no clear winner for area coverage, with spanning tree covering and distance transform path both having their advantages and disadvantages. Both authors believe that for this use case, the spanning tree covering performs better since the improvement in speed is more important. However, both algorithms are available and can be used to support the use-case **UC-Map**.

Two exploration algorithms, both based on the frontier strategy, were evaluated. Both algorithms performed similarly, however Hector exploration had more features and discovered a larger area faster. Together with Cartographer, Hector should be able to discover most buildings and is therefore a solution to the use-case **UC-NoMap**.

This thesis has presented a working prototype for autonomous measurement collection in both explored and unexplored buildings. By using the fleet management platform the system can be used without the requirement of high technical expertise or domain knowledge. The prototype supports exploring and mapping a completely unknown environment, and methodically and in detail cover the same environment after it has been mapped. The prototype therefore supports all of the use-cases which were presented in Chapter 1.

Bibliography

[1] *Application programming interface*. 17th May 2017. URL: https://en.wikipedia.org/wiki/Application_programming_interface.

[2] Sameer Agarwal, Keir Mierle et al. *Ceres Solver*. URL: http://ceres-solver.org.

[3] *Inertial measurement unit*. 27th Apr. 2017. URL: https://en.wikipedia.org/wiki/Inertial_measurement_unit.

[4] *Simultaneous localization and mapping*. 7th June 2017. URL: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping.

[5] *Virtual private network*. 27th Apr. 2017. URL: https://en.wikipedia.org/wiki/Virtual_private_network.

[6] S. Burgess et al. 'Smartphone positioning in multi-floor environments without calibration or added infrastructure'. In: *IEEE International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. Madrid, Spain, Oct. 2016, pp. 1–8. DOI: 10.1109/IPIN.2016.7743653.

[7] *Combain CPS - Android Apps on Google Play*. 7th Mar. 2017. URL: https://play.google.com/store/apps/details?id=com.combain.cpsdemo&hl=en.

[8] *ROS.org | Powering the world's robots*. 7th June 2017. URL: http://www.ros.org/.

[9] *Gazebo*. 7th June 2017. URL: http://gazebosim.org/.

[10] *Stage Manual*. 7th June 2017. URL: http://playerstage.sourceforge.net/doc/Stage-3.2.1/.

[11] *Turtlebot*. 9th Feb. 2017. URL: http://www.turtlebot.com/.

[12] *About | KOBUKI*. 9th Feb. 2017. URL: http://kobuki.yujinrobot.com/about2/.

[13] *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi.* 17th May 2017. URL: https://www.raspberrypi.org/.

[14] *A1 Series - RPLIDAR - Slamtec - Leading Service Robot Localization and Navigation Solution Provider.* 9th Feb. 2017. URL: http://www.slamtec.com/en/Lidar/A1.

[15] *Astra Pro - Orbbec.* 9th Feb. 2017. URL: https://orbbec3d.com/product-astra-pro/.

[16] B. Yamauchi, A. Schultz and W. Adams. 'Mobile robot exploration and map-building with continuous localization'. In: *IEEE International Conference on Robotics and Automation (ICRA).* Vol. 4. Leuven, Belgium, May 1998, 3715–3720 vol.4. DOI: 10.1109/ROBOT.1998.681416.

[17] Z. Riaz et al. 'A fully autonomous indoor mobile robot using SLAM'. In: *IEEE International Conference on Information and Emerging Technologies (ICIET).* Karachi, Pakistan, June 2010, pp. 1–6. DOI: 10.1109/ICIET.2010.5625691.

[18] Stefan Kohlbrecher et al. 'Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots'. In: *RoboCup 2013: Robot World Cup XVII.* Ed. by Sven Behnke et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 624–631. ISBN: 978-3-662-44468-9. DOI: 10.1007/978-3-662-44468-9_58.

[19] S. Kohlbrecher et al. 'A flexible and scalable SLAM system with full 3D motion estimation'. In: *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR).* Kyoto, Japan, Nov. 2011, pp. 155–160. DOI: 10.1109/SSRR.2011.6106777.

[20] S. Wirth and J. Pellenz. 'Exploration transform: A stable exploring algorithm for robots in rescue environments'. In: *IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR).* Rome, Italy, Sept. 2007, pp. 1–5.

[21] P. M. Scholl et al. 'Fast indoor radio-map building for RSSI-based localization systems'. In: *2012 Ninth International Conference on Networked Sensing (INSS).* Antwerp, Belgium, June 2012, pp. 1–2. DOI: 10.1109/INSS.2012.6240574.

[22] G. Grisetti, C. Stachniss and W. Burgard. 'Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling'. In: *IEEE International Conference on Robotics and Automation (ICRA).* Seattle, Washington, USA, Apr. 2005, pp. 2432–2437. DOI: 10.1109/ROBOT.2005.1570477.

[23] G. Grisetti, C. Stachniss and W. Burgard. 'Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters'. In: *IEEE Transactions on Robotics* 23.1 (Feb. 2007), pp. 34–46. DOI: 10.1109/TRO.2006.889486.

[24] W. Hess et al. 'Real-time loop closure in 2D LIDAR SLAM'. In: *IEEE International Conference on Robotics and Automation (ICRA).* Stockholm, Sweden, May 2016, pp. 1271–1278. DOI: 10.1109/ICRA.2016.7487258.

[25] H. Moritz. 'Geodetic Reference System 1980'. In: *Journal of Geodesy* 74.1 (2000), pp. 128–133. ISSN: 1432-1394. DOI: 10.1007/s001900050278.

[26] Y. Gabriely and E. Rimon. 'Spanning-tree based coverage of continuous areas by a mobile robot'. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 2. 2001, 1927–1933 vol.2. DOI: 10.1109/ROBOT.2001.932890.

[27] R. C. Prim. 'Shortest connection networks and some generalizations'. In: *The Bell System Technical Journal* 36.6 (Nov. 1957), pp. 1389–1401. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1957.tb01515.x.

[28] A. Zelinsky et al. 'Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot'. In: *IEEE International Conference on Advanced Robotics (ICRA)*. Atlanta, GA, USA, 1993, pp. 533–538.

[29] B. Yamauchi. 'A frontier-based approach for autonomous exploration'. In: *IEEE Computational Intelligence in Robotics and Automation (CIRA)*. Monterey, CA, USA, July 1997, pp. 146–151. DOI: 10.1109/CIRA.1997.613851.

[30] D. Holz et al. 'Evaluating the Efficiency of Frontier-based Exploration Strategies'. In: *ISR (41st International Symposium on Robotics) and ROBOTIK (6th German Conference on Robotics)*. Munich, Germany, June 2010, pp. 1–8.

[31] *frontier_exploration - ROS Wiki*. 4th May 2017. URL: http://wiki.ros.org/frontier_exploration.

[32] P. Kumar, L. Reddy and S. Varma. 'Distance measurement and error estimation scheme for RSSI based localization in Wireless Sensor Networks'. In: *IEEE Fifth International Conference on Wireless Communication and Sensor Networks (WCSN)*. Allahabad, India, Dec. 2009, pp. 1–4. DOI: 10.1109/WCSN.2009.5434802.

[33] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. 2nd ed. Prentice Hall, Jan. 2002. ISBN: 0130422320.

[34] Xun Wang and JianGuo Wang. 'Detecting glass in Simultaneous Localisation and Mapping'. In: *Robotics and Autonomous Systems* 88 (2017), pp. 97–103. ISSN: 0921-8890. DOI: 10.1016/j.robot.2016.11.003.

[35] T. Zhang et al. 'Sensor fusion for localization, mapping and navigation in an indoor environment'. In: *International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*. Puerto Princesa, Philippines, Nov. 2014, pp. 1–6. DOI: 10.1109/HNICEM.2014.7016188.