



UNIVERSITY OF AGDER

Simultaneous Localization and Mapping in
Repeating Environments

BY

THOMAS ÅNENSEN

25/05-2018

SUPERVISOR:

KRISTIAN MURI KNAUSGÅRD

COURSE:

MAS-500 SPRING 2018

This master's thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

UNIVERSITY OF AGDER, 2018

FACULTY OF TECHNOLOGY AND SCIENCE

DEPARTMENT OF ENGINEERING

Acknowledgements

After five years of studying at the University of Agder, this thesis will signify as my final task carried out on the university. As such, I would like to thank the Faculty of Technology & Science for all the knowledge I have gained throughout the years of studying at Campus Grimstad. Additionally, I would like to thank Kristian Muri Knausgård for the encouragement and guidance he provided during the making of this thesis.

Further, I would like to thank Steffen Solberg at KVS Technologies for reaching out to me and making this whole thesis happen. In addition to orchestrating this thesis, he has helped me during the entire thesis, either over phone or in person. This was greatly appreciated. I would also like to thank Tor Morten Finnestad at KVS Technologies. Without his technical support, the SPURV would still be driven with a Ethernet cable sticking out of it.

Next, my gratitude goes out to my friends and peers from both in and outside of the university. Without your support, friendship and occasional bad jokes, the last few years would have been impossible to get through.

Lastly, my everlasting gratitude goes to my loving family. Without them I would still struggle to find my way in life. Their guidance and support means the world to me.

Thomas Ånensen

Abstract

In this thesis, the *GMapping* and *Hector SLAM* algorithms are going to be compared to find the best solution to solve the simultaneous localization and mapping problem. Robot Operating System are going to be the proprietary system form this thesis, and both *GMapping* and *Hector SLAM* have compatible programs for Robot Operating System. A testing area at the University of Agder Campus Grimstad was defined, and both algorithms were tested using the same testing data. The testing area consisted of areas with repeating elements and few features. The test data was collected by driving a SPURV Research robot through the testing route, recording necessary data. The data was played back in such a way that the algorithms interpreted it as if the SPURV was driving live. The results can be seen in Figure 1 and shows that *GMapping* and *Hector SLAM* are both viable solutions for the simultaneous localization and mapping problems tested in this thesis. Figure 1 a) and c) are the results of *GMapping* and Figure 1 b) and d) are the results of *Hector SLAM*. Both solutions depict the testing area well.

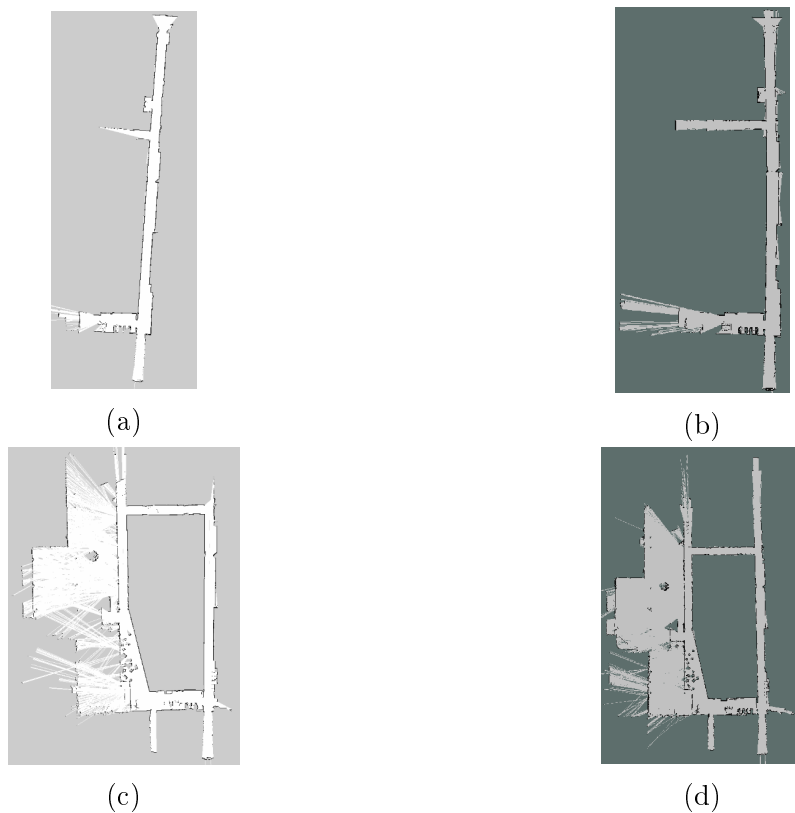


Figure 1: Comparison of maps generated from GMapping (left) vs Hector SLAM (right)

Contents

List of Figures	v
List of Tables	ix
List of Abbreviations	x
1 Introduction	1
1.1 Project Description	1
1.2 Objective	2
1.3 Concept	2
2 Theory	3
2.1 Robot Operating System	3
2.2 Mathematical Representation of SLAM	4
2.3 Bayes Filter with Static State	6
2.4 Occupancy Grid Mapping	8
2.5 Scan Matching	12
2.6 Particle Filters	13
2.7 Monte Carlo Localization	18
2.8 Kalman Filter	19
2.9 Extended Kalman Filter	21
2.10 Rao-Blackwellization	23
2.11 Feature-Based FastSLAM 1.0 and 2.0	24
2.11.1 FastSALM 1.0 with Known Correspondence	24
2.11.2 FastSLAM 2.0	28
2.12 Grid-Based FastSLAM	32
3 Method	36
3.1 Working Method	36
3.2 Selection of SLAM Approach	37
3.3 Software	38
3.4 GMapping	38
3.5 Hector SLAM	39
3.6 Why GMapping and Hector SLAM?	39
3.7 The SPURV Robot	40
3.8 Test 1 - Initial Testing	41
3.8.1 Test Plan for Test 1	41

3.8.2	Execution of Test 1	43
3.9	Test 2 - Controlling SPURV, Identify Topics	46
3.9.1	Test Plan for Test 2	46
3.9.2	Execution of Test 2	47
3.10	Fixing the Odometry of the SPURV	51
3.11	Test 3 - Verify Odometry, Driving Speed	53
3.11.1	Test Plan for Test 3	53
3.11.2	Execution of Test 3	53
3.12	Test 4 - GMapping Parameter Test	58
3.12.1	Test Plan for Test 4	58
3.12.2	Execution of Test 4	58
3.13	Test 5 - Odometry Investigation	61
3.13.1	Test Plan for Test 5	61
3.13.2	Execution of Test 5	61
3.14	Test 6 - Hector SLAM	62
3.14.1	Test Plan for Test 6	62
3.14.2	Execution of Test 6	63
3.15	Test 7 - MATLAB Plot of Odometry	65
3.15.1	Test Plan for Test 7	65
3.15.2	Execution of Test 7	65
3.16	Test 8 - Repeating, Featureless Environment	66
3.16.1	Test Plan for Test 8	66
3.16.2	Execution of Test 8	66
4	Results	68
4.1	Test 1 - Initial Testing	68
4.2	Test 2 - Controlling SPURV, Identify Topics	69
4.3	Test 3 - Verify Odometry, Driving Speed	70
4.4	Test 4 - GMapping Parameter Test	71
4.5	Test 5 - Odometry Investigation	75
4.5.1	Results of Test 5 Part 1	75
4.5.2	Results of Test 5 Part 2	76
4.5.3	Results of Test 5 Part 3	76
4.6	Test 6 - Hector SLAM	77
4.7	Test 7 - MATLAB Plot of Odometry	80
4.8	Test 8 - Repeating, Featureless Environment	83
4.8.1	Results of Test 8 Part 1	83
4.8.2	Results of Test 8 Part 2	84
5	Discussion	86
5.1	Test 1 - Initial Testing	86
5.2	Test 2 - Controlling SPURV, Identify Topics	87
5.3	Test 3 - Verify Odometry, Driving Speed	88
5.4	Test 4 - GMapping Parameter Test	90
5.5	Test 5 - Odometry Investigation	92
5.6	Test 6 - Hector SLAM	94
5.7	Test 7 - MATLAB Plot of Odometry	97

5.8	Test 8 - Repeating, Featureless Environment	98
5.9	Evaluation of the Testing Method	100
6	Conclusions and Recommendations	101
7	Further Work	103
	Bibliography	104
A	All Results from Test 5 Part 2	A - 1
B	Setting Up ROS Workspace "gmapping_ws"	B - 1
C	Building the ROS Package in "gmapping_ws"	C - 1
D	Testing of "slam_gmapping"	D - 1
E	Wired Connection to the SPURV	E - 1
F	Wireless Connection to the SPURV	F - 1

List of Figures

1	Comparison of maps generated from GMapping (left) vs Hector SLAM (right)	ii
1.1	SPURV Responder robot in a tunnel [9]	2
2.1	ROS module diagram [17]	4
2.2	Example of a grid map [23]	8
2.3	Example of an inverse sensor measurement model [23]	11
2.4	Algorithm for an inverse measurement model of a range finder [23]	11
2.5	Particle distribution after passing through a non-linear function [23]	13
2.6	Target and proposal distribution with the resulting samples [22]	15
2.7	Particle filter algorithm [22]	16
2.8	Roulette wheel approach to resampling [22]	16
2.9	Stochastic universal sampling [22]	17
2.10	Algorithm for low variance resampling [22]	17
2.11	Monte Carlo localization algorithm [22]	18
2.12	Kalman filter algorithm [23]	21
2.13	Extended Kalman filter linearisation [23]	21
2.14	Extended Kalman filter algorithm [23]	23
2.15	FastSLAM 1.0 algorithm [23]	27
2.16	FastSLAM algorithm for occupancy grid maps [23]	32
3.1	Illustration of the V-model working method [11]	36
3.2	SPURV Research used in thesis	40
3.3	Sketch showing the placement of sensors on the SPURV	40
3.4	Sketch showing the coordinate system used on the SPURV, as seen from the rear	41
3.5	Depiction of the route used to perform Test 1	43
3.6	Continuation of the route used in Test 1	44
3.7	Function in .bashrc to set correct the IP-addresses to the SPURV	45
3.8	Function in .bashrc to set the IP addresses back to the local computer	48
3.9	Launch file used in test 2 for pointcloud_to_laserscan algorithm in ROS	49
3.10	Launch file used in test 2 for GMapping in ROS	50
3.11	Launch file used in test 3 for pointcloud_to_laserscan with highlighted alterations	55
3.12	Launch file used in test 3 for GMapping in ROS	56
3.13	Launch file used in test 3 for GMapping with highlighted alterations	58
3.14	Debris found in the Long Hallway during Test 4	60
3.15	Debris found in the Long Hallway during Test 5	61
3.16	Launch file used in test 6 for hector_slam algorithm in ROS	63

3.17	MATLAB code used to plot /odom topic from rosbags	65
3.18	Testing area used in the first part of Test 8	66
3.19	Featureless area used in the second part of Test 8	67
4.1	Map generated using GMapping from the second test	68
4.2	Transition between /cloud topic to /scan topic	69
4.3	Map generated using GMapping from the second test	69
4.4	Maps generated using GMapping from the third test	70
4.5	Maps generated using GMapping from the fourth test	71
4.6	Maps generated using GMapping from the parameter test	72
4.7	Maps generated using GMapping from the parameter test continuation	73
4.8	Maps generated using GMapping resulting from Test 5 part 1	75
4.9	Maps generated using GMapping resulting from Test 5 part 2	76
4.10	Maps generated using GMapping resulting from Test 5 part 3	76
4.11	Maps generated using Hector SLAM	77
4.12	Maps generated using Hector SLAM continuation	78
4.13	Plot of the odometry of the SPURV during testing from MATLAB	80
4.14	Plot of the odometry of the SPURV during testing from MATLAB continuation	81
4.15	Results of Test 8 Part 1 with Hector SLAM, GMapping and odometry	83
4.16	Results of Test 8 Part 2 with Hector SLAM, GMapping and odometry	84
4.17	Position estimate from Hector SLAM during Test 8 Part 2	85
5.1	Map generated using GMapping from the second test	87
5.2	Maps generated using GMapping from the third test	88
5.3	Comparison of maps from Test 2 and Test 3	89
5.4	Comparison of maps generated by GMapping at brisk walking speed from Test 3 and Test 4	90
5.5	Comparison of maps generated by GMapping at slow walking speed from Test 3 and Test 4	90
5.6	Best resulting map using GMapping from Test 4	91
5.7	Map showcasing the "spiral" tendency from Test 4	91
5.8	Map showcasing the "spiral" tendency from Test 5	92
5.9	Maps generated using GMapping resulting from Test 5	93
5.10	Comparison of maps generated from GMapping (left) vs Hector SLAM (right)	94
5.11	Comparison of maps generated from GMapping (left) vs Hector SLAM (right) continuation	95
5.12	Comparison of maps generated by GMapping (left) and Hector SLAM (right)	96
5.13	Plot of odometry vs map generated in GMapping from Test 2	97
5.14	Plot of odometry vs map generated in GMapping from Test 5 Part 1	97
5.15	Results of Test 8 with Hector SLAM and GMapping	98
5.16	Position estimates coming from Test 8 from odometry and Hector SLAM	99
A.1	All test results from Test 5 Part 2	A - 1
A.2	All test results from Test 5 Part 2 continuation	A - 2
D.1	Map produced by logged data	D - 2
E.1	Rear panel of the SPURV Research	E - 1

E.2	Functions in .bashrc to set correct IP-addresses	E - 2
E.3	Entries in the ssh config file	E - 2
E.4	Alias used in .bashrc	E - 3
F.1	Finding neighbouring networks in WinBox	F - 1
F.2	Set-up of the wireless connections of the SPURV	F - 2

List of Tables

3.1	Testing parameters investigated in Test 4	60
4.1	Testing parameters used in Test 4	74
4.2	Parameters used in Test 5 part 1	75
4.3	Parameters used in Test 5 part 2	76
4.4	Parameters used in Test 5 part 3	77
4.5	Hector SLAM results	79
4.6	Corresponding data to the odometry plots	81
4.7	Error between starting point and end point from Test 7	82
4.8	Error between starting point and end point from Test 8 Part 1	84
4.9	Error between starting point and end point from Test 8 Part 2	85
5.1	GMapping vs Hector SLAM	95
5.2	GMapping vs Hector SLAM	96
A.1	Test Parameters used in Test 5 Part 2	A - 1

List of Abbreviations

EKF:	Extended Kalman filter
KF:	Kalman filter
MCL:	Monte Carlo localization
PF:	Particle filter
Pose:	Position and orientation
ROS:	Robot Operating System
SLAM:	Simultaneous localization and mapping

Chapter 1

Introduction

KVS Technologies has developed an unmanned ground vehicle called "SPURV". This vehicle is going to reduce the risks associated with fire fighting in tunnels, underground structures and large industrial sites and buildings. Currently, the robot is operated by teleoperation, however, the company would like the robot to perform tasks autonomously. Using Simultaneous Localization and Mapping, abbreviated SLAM [23], the robot is going to be able to generate a map in unknown environments and navigate using this map. This, in turn, is going to give the fire fighters vital information about what to expect, which will make their job easier and safer.

KVS Technologies are collaborating with UiA and UiS to make the SPURV autonomous. The student at UiA is looking into mapping and localization in repeating environments while the student at UiS looks into path planning and obstacle avoidance. To make the SPURV able to autonomously perform the required task of independently driving through the repeating environments, both path planning, obstacle avoidance, localization and mapping has to be present and working in real-time.

1.1 Project Description

The "SPURV" robot is going to use Simultaneous Localization and Mapping to generate a map of an unknown environment with repeating features. This technique enables the SPURV to generate a map of an unknown environment and use this map to localize itself. Sensors are going to be used to construct the map and localize the robot. In this thesis, different SLAM algorithms are going to be researched and compared against each other. This was done to find the best algorithm suited for repeating environments with few features. The following requirements has been set for the algorithms:

- The algorithms must be compatible with Robot Operating System, as this is the system used on the SPURV
- Operational speed is going to be walking speed
- The algorithm needs to be able to store the maps created by the mapping process
- The algorithm must be able to map large areas

- The algorithm should work for areas which can not be reached with GPS signals

The robot is going to be used in road tunnels with both straight and jagged walls. Therefore, the algorithms has to be able to handle environments with repeating elements and few features. The algorithms researched in this thesis does not have to take smoke filled environments into account. When the robot starts the first time in a new unknown environment, the reference coordinate system of the robot is defined by the position and orientation of the robot when it is switched on. This will also be the reference coordinate system used in the map created by the SLAM algorithms.

1.2 Objective

This thesis is going to account for the theory needed to understand the probabilistic nature of navigating in an unknown environment, creating a map and localizing the robot simultaneously. Additionally, the programming in Robot Operating System and the tests conducted to verify the performance of the Simultaneous Localization and Mapping algorithm is going to be described. The results of the tests are going to be discussed before giving a conclusion and recommendation on further work that can be done. Different SLAM algorithms are going to be tested to find the best approach for repeating environments. All figures used in this thesis has been either made for the thesis or used with permission from the respective author(s).

1.3 Concept

The robot is equipped with a Lidar laser range finder, two cameras and odometry based on Ackerman steering. The primary sensor used during the Simultaneous Localization and Mapping procedure is the laser range finder. This sensor will provide a good field of view to detect features of the surrounding environment which is going to be crucial when constructing the map. In Figure 1.1 the SPURV robot is shown in a typical tunnel environment in which it should be able to operate. SLAM was chosen as the preferred method of navigation due to a lack of pre made maps of the areas where the SPURV would be operating.



Figure 1.1: SPURV Responder robot in a tunnel [9]

Chapter 2

Theory

Simultaneous Localization and Mapping, abbreviated SLAM, is a method in which a robot can navigate in unknown territory and still retain a certain accuracy regarding its own position and heading. This is necessary when the robot is going to autonomously follow a certain path, for instance in a map. In this way, the robot can operate in an environment where there is no prior knowledge of said environment. This is especially useful when there is no opportunity of placing beacons or landmarks in the environment nor use GPS for localization of the robot. Precise localization is crucial for a robot to be truly autonomous and operate without human interaction.

The challenging part of the SLAM problem is to be able to localize the position and orientation of the robot while a map is being constructed. To be able to construct a good map, the position at which the sensor observations was taken has to be known. At the same time, it is challenging to estimate the placement of the robot accurately without a map. In other words, a good map is needed to estimate the position and orientation of the robot, and an accurate position and orientation is needed to construct a map. Because of this strong dependency between localization and mapping, the SLAM problem is often referred to as a "chicken-and-egg" problem [4] [18] [23].

2.1 Robot Operating System

Robot Operating System, or ROS, is an open source framework for writing robot software [12]. It consists of a collection of libraries and tools which can be used to simplify the creation of complex and robust robot algorithms to control a wide array of robots. ROS is designed from the ground up to support collaborative robotics software development. The ROS system enables each developer group to have the ROS code repository on their own server [14]. Although it would be easier to have all repository stored on a single server, the advantage is that each group gets control over their own code. Additionally, each group gets the recognition and credit they deserve based on their work. This is one of the greatest strength of ROS. By enabling the development of software to be modular, the community can reuse the modules that is need for the individual projects. This enables the ROS software to be shared, used and expanded upon by the ROS community.

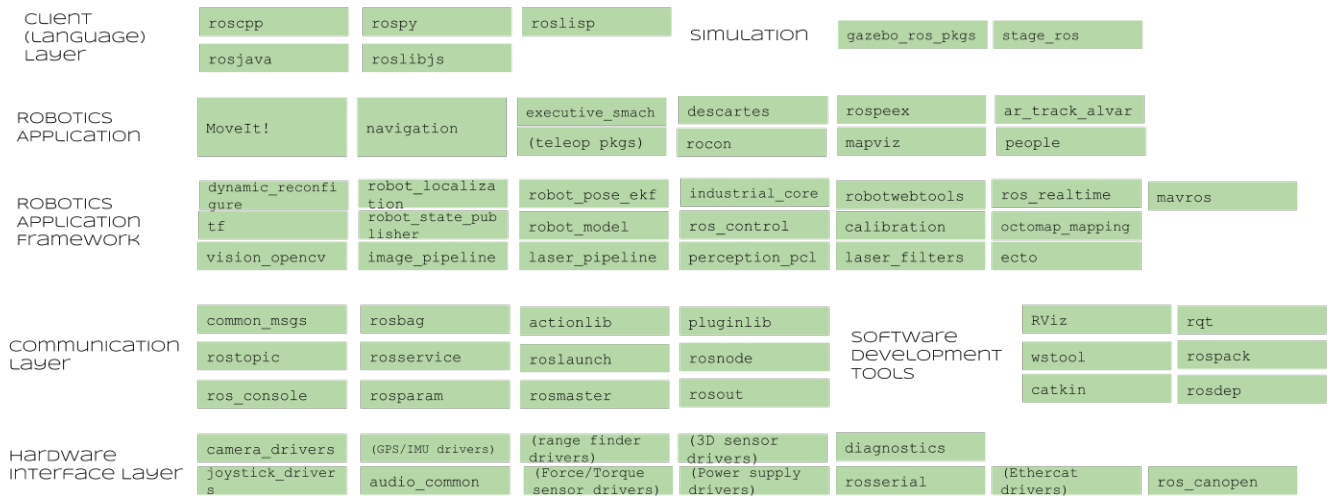


Figure 2.1: ROS module diagram [17]

In a nutshell, ROS is a collection of packages and libraries. Figure 2.1 shows some of the packages found in ROS. The ROS node plays a great part in the ROS system. In the nodes, most of the programs are executed. Information to or from the different nodes are communicated using ROS messages, rostopics, rosservice and rosparameters, to name a few ways information is transferred between the nodes. Additionally, information can be recorded and played back using the rosbag package. RViz and rqt are great development tools, where RViz lets the user visualize the sensor data and rqt can be used to plot data published by ros topics and visualize the systems running in ROS [16].

2.2 Mathematical Representation of SLAM

In this thesis, the main approach to the SLAM problem would be the 2D SLAM problem. The reason was due to the Lidar laser range sensor, which would be the main sensor used for SLAM. The Lidar scans in the 2D plane, and thus creating a 2D map seemed sensible. To be able to create 3D maps, additional sensors were needed, for instance, to measure the difference in elevation or generate 3D point clouds of the environment.

To be able to describe the SLAM problem in probabilistic terminology, some terms has to be defined. These ideas and concepts are based on chapter 5.6.4 *Terminology* in the book *Introduction to Autonomous Mobile Robotics* [18] and chapter 5.1 *Introduction* in the book *Probabilistic Robotics* [23].

Position and orientation will hereafter be denoted as *pose*. x_t is here defined as the pose at time t . For a 2D planar motion, this will typically be presented as a three-dimensional vector $x_t = [x, y, \theta]^T$ where x and y is the Cartesian coordinates and θ is the heading. The path from the initial pose, x_0 , to the current pose, x_t , is given as

$$x_{0:t} = \{x_0, x_1, x_2, \dots, x_t\} \quad (2.1)$$

where

$x_{0:t}$: poses of the robot from time $t = 0$ to time $t = t$

Now the control data has to be defined. These data are usually encoder readings from the wheels of the robot or the control commands given to the robot. The control data at time t will be denoted as u_t . This gives the control data from time $t = 0$ to $t = t$ as

$$u_{0:t} = \{u_0, u_1, u_2, \dots, u_t\} \quad (2.2)$$

where

$u_{0:t}$: control data to the robot from time $t = 0$ to time $t = t$

Next, the map of the environment at time t is defined as m_t , giving the map from $t = 0$ to $t = t$ as

$$m_{0:t} = \{m_0, m_1, m_2, \dots, m_t\} \quad (2.3)$$

where

$m_{0:t}$: map of the environment from time $t = 0$ to time $t = t$

Lastly, the measurement data at time t is defined as z_t , giving the measurements from $t = 0$ to $t = t$ as

$$z_{0:t} = \{z_0, z_1, z_2, \dots, z_t\} \quad (2.4)$$

where

$z_{0:t}$: measurement from time $t = 0$ to time $t = t$

As robot pose, control data, map and measurement data now is defined, the SLAM problem can be defined as well. The goal is to get a probability distribution of the pose x and map m given control data u and observations from the measurement z . There are two main forms of SLAM, the *full SLAM problem* and the *online SLAM problem*.

The online SLAM problem is based on finding the momentary pose x_t of the robot as well as the full map m given the measurement $z_{1:t}$ and the control data $u_{1:t}$. This is given as

$$p(x_t, m \mid z_{1:t}, u_{1:t}) \quad (2.5)$$

where

x_t : momentary pose of the robot

m : map

$z_{1:t}$: observations from measurement from time $t = 1$ to $t = t$

$u_{1:t}$: control data from time $t = 1$ to $t = t$

Here, $p(x_t, m \mid z_{1:t}, u_{1:t})$ can be informally translated to "the probability distribution of the current pose x_t and map m given all observations $z_{1:t}$ and all control data $u_{1:t}$ ".

In contrast to the online SLAM problem, the full SLAM problem calculates the posterior of the entire path of the robot, along with the map. This takes the entire path $x_{1:t}$ into consideration and not just the latest pose x_t . The full SLAM problem is given as

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \quad (2.6)$$

where

$x_{1:t}$: entire path of the robot

m : map

$z_{1:t}$: observations from measurement from time $t = 1$ to $t = t$

$u_{1:t}$: control data from time $t = 1$ to $t = t$

In this thesis, the focus is going to be on the online SLAM problem.

Lastly, two more concepts has to be introduced: the *motion model* and the *measurement model*. The motion model is the probability distribution of the current pose x_t given the previous pose x_{t-1} and the latest control data u_t . This is expressed as $p(x_t \mid x_{t-1}, u_t)$. The measurement model is the probability distribution of the current observation z_t given the current robot pose x_t and the map m . This is expressed as $p(z_t \mid x_t, m)$.

2.3 Bayes Filter with Static State

For problems in robotics which can be formulated as a problem with binary states that does not change over time, the Bayes filter with static state can be applied to address such problems. This technique will be applied later when discussing occupancy grid mapping. This theory is based on the concepts described in chapter 4.2 *Binary Bayes Filters with Static State* in the book *Probabilistic Robotics* [23]. The belief of the state can be represented as a function of only the measurement.

$$bel_t(x) = p(x \mid z_{1:t}, u_{1:t}) = p(x \mid z_{1:t}) \quad (2.7)$$

The binary state has two possible values, x or \bar{x} where

$$bel_t(\bar{x}) = 1 - bel_t(x) \quad (2.8)$$

Commonly, this belief is set up as a *log odds ratio*. The *odds* of a state x is defined as the ratio of the probability of this event divided by the probability of its negate

$$\frac{p(x)}{p(\bar{x})} = \frac{p(x)}{1 - p(x)} \quad (2.9)$$

the log odds is therefore the logarithm to this expression

$$l(x) = \log \left(\frac{p(x)}{1 - p(x)} \right) \quad (2.10)$$

The belief can be reinstated by using the following expression

$$bel_t(x) = 1 - \frac{1}{1 + exp\{l(x)\}} \quad (2.11)$$

By applying Bayes rule on equation (2.7), the following expression is derived

$$p(x | z_{1:t}) \stackrel{Bayes}{=} \frac{p(z_t | x, z_{1:t-1}) p(x | z_{1:t-1})}{p(z_t | z_{1:t-1})} \quad (2.12)$$

Then, by applying the Markov assumption, the expression becomes

$$p(x | z_{1:t}) \stackrel{Markov}{=} \frac{p(z_t | x) p(x | z_{1:t-1})}{p(z_t | z_{1:t-1})} \quad (2.13)$$

Bayes rule can be applied on the measurement model $p(z_t | x)$, giving

$$p(z_t | x) \stackrel{Bayes}{=} \frac{p(x | z_t) p(z_t)}{p(x)} \quad (2.14)$$

and by combining equation (2.13) and (2.14), the following expression is found

$$p(x | z_{1:t}) = \frac{p(x | z_t) p(z_t) p(x | z_{1:t-1})}{p(x) p(z_t | z_{1:t-1})} \quad (2.15)$$

By using the same methodology, the expression for \bar{x} can be found

$$p(\bar{x} | z_{1:t}) = \frac{p(\bar{x} | z_t) p(z_t) p(\bar{x} | z_{1:t-1})}{p(\bar{x}) p(z_t | z_{1:t-1})} \quad (2.16)$$

Dividing (2.15) by (2.16) leads to cancellation of difficult to calculate probabilities, resulting in

$$\begin{aligned} \frac{p(x | z_{1:t})}{p(\bar{x} | z_{1:t})} &= \frac{p(x | z_t)}{p(\bar{x} | z_t)} \frac{p(x | z_{1:t-1})}{p(\bar{x} | z_{1:t-1})} \frac{p(\bar{x})}{p(x)} \\ &= \frac{p(x | z_t)}{1 - p(x | z_t)} \frac{p(x | z_{1:t-1})}{1 - p(x | z_{1:t-1})} \frac{1 - p(x)}{p(x)} \end{aligned} \quad (2.17)$$

Now, by applying the log odds ratio of the belief, the following expressing is the result

$$\begin{aligned} l_t(x) &= \log \left(\frac{p(x | z_t)}{1 - p(x | z_t)} \right) + \log \left(\frac{p(x | z_{1:t-1})}{1 - p(x | z_{1:t-1})} \right) + \log \left(\frac{1 - p(x)}{p(x)} \right) \\ &= \log \left(\frac{p(x | z_t)}{1 - p(x | z_t)} \right) - \log \left(\frac{p(x)}{1 - p(x)} \right) + l_{t-1}(x) \end{aligned} \quad (2.18)$$

and with prior initial belief

$$l_0(x) = \log \left(\frac{1 - p(x)}{p(x)} \right) \quad (2.19)$$

2.4 Occupancy Grid Mapping

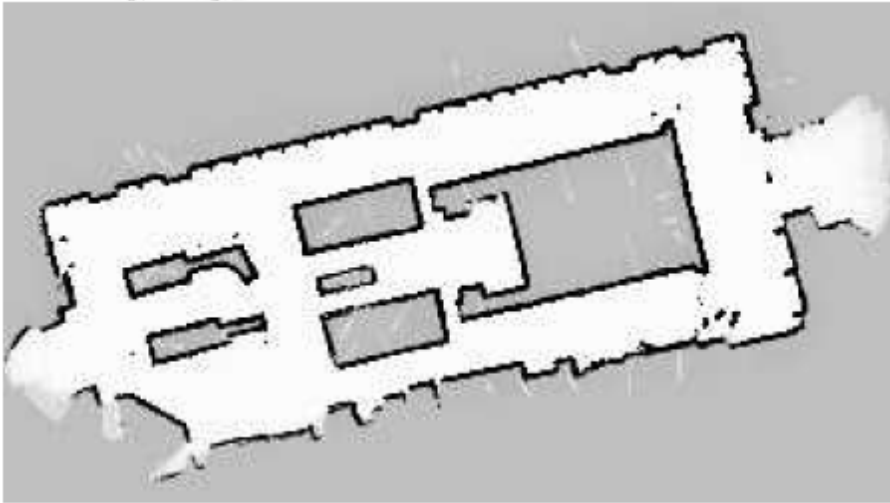


Figure 2.2: Example of a grid map [23]

Given a known pose of the robot, occupancy grid mapping is a good way to address the problem of generating consistent maps from noisy and uncertain measurement data. The general idea of the occupancy grid is to represent the world with a grid of evenly spaced binary variables, also known as *grid cells* [23]. One of the big advantages of this mapping method is that there is no need for a feature detector, as the map is constructed using sensor data. An example of such a map representation is shown in Figure 2.2. The concepts described here are based on chapter 9 *Occupancy Grid Mapping* in the book *Probabilistic Robotics* [23].

There are three underlying assumptions when using occupancy grid mapping:

1. The area that corresponds to a cell is either completely free or occupied
2. The world is static
3. The cells are independent of each other

In this section, there is a restrictive assumption that the robot poses are known. This problem is also known as *mapping with known poses*.

The main goal of the occupancy grid mapping is to calculate the map based on the available data

$$p(m \mid z_{1:t}, x_{1:t}) \quad (2.20)$$

where

m : map

$z_{1:t}$: set of measurements up to time t

$x_{1:t}$: path of the robot with known poses

Let m_i denote the grid cell with index i , and the map is a set of these grid cells

$$m = \{m_i\} \quad (2.21)$$

Each grid cell, m_i , has a binary value attached to it: "1" for occupied space or "0" for free space. The standard occupancy grid approach is to calculate the map for a collection of separate problems. The reason for this is due to the computation load of calculating the posterior probability for a single map. This computational load would be too high, making the calculation of the map infeasible. The resulting sub problems of the map are to determine

$$p(m_i \mid z_{1:t}, x_{1:t}) \quad (2.22)$$

for all grid cells m_i . The posterior of the map is approximated to the product of its marginals

$$p(m \mid z_{1:t}, x_{1:t}) = \prod_i p(m_i \mid z_{1:t}, x_{1:t}) \quad (2.23)$$

The occupancy grid problem is now a binary estimation problem with static state. A binary Bayes filter with static state can be applied, and this filter does not need a prediction step, only the correction step. The procedure in chapter 2.3 will be applied for the probability distribution of a grid cell, $p(m_i \mid z_{1:t}, x_{1:t})$. By using the log odds representation, the following expression is found

$$l_{t,i} = \log \left(\frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})} \right) \quad (2.24)$$

giving the probability

$$p(m_i | z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + \exp\{l_{t,i}\}} \quad (2.25)$$

The constant l_0 is the prior of occupancy represented as a log odds ratio

$$l_0 = \log \left(\frac{p(m_i = 1)}{p(m_i = 0)} \right) = \log \left(\frac{p(m_i)}{1 - p(m_i)} \right) \quad (2.26)$$

By applying the Bayes filter with static state on the expression in the parenthesis in equation (2.24), the following term results

$$\frac{p(m_i | z_{1:t}, x_{1:t})}{1 - p(m_i | z_{1:t}, x_{1:t})} = \frac{p(m_i | z_t, x_t)}{1 - p(m_i | z_t, x_t)} \frac{p(m_i | z_{1:t-1}, x_{1:t-1})}{1 - p(m_i | z_{1:t-1}, x_{1:t-1})} \frac{1 - p(m_i)}{p(m_i)} \quad (2.27)$$

and by putting it in log odds form, and exploiting the fact that the products of logarithms turns into a sum, the final expression results

$$l(m_i | z_{1:t}, x_{1:t}) = \underbrace{l(m_i | z_t, x_t)}_{\text{inverse sensor model}} + \underbrace{l(m_i | z_{1:t-1}, x_{1:t-1})}_{\text{recursive term}} - \underbrace{l(m_i)}_{\text{prior}} \quad (2.28)$$

or in short

$$l_{t,i} = \text{inv_sensor_model}(m_i, x_t, z_t) + l_{t-1,i} - l_0 \quad (2.29)$$

The expression $\log \left(\frac{p(m_i | z_t, x_t)}{1 - p(m_i | z_t, x_t)} \right)$ is called the *inverse sensor model*. In contrast to the measurement model, the inverse sensor model gives the probability distribution of the map given the measurement and current pose, not the other way around. In other words, the inverse sensor model can be used to generate a map based on measurement and pose only. Equation (2.27) and (2.28) are found in the lecture *Grid Maps* from Cyrill Stachniss [21] as the book *Probabilistic Robotics* [23] did not go too much in detail on how to derive the final expression of *grid mapping with known poses*.

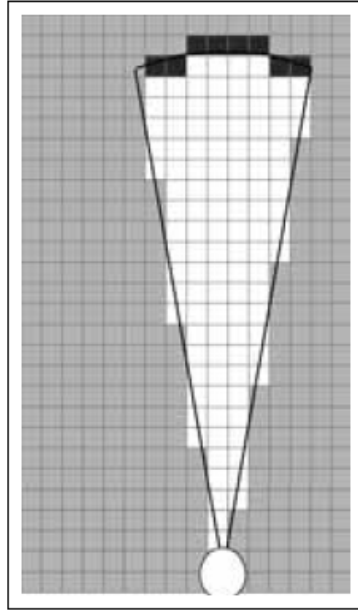


Figure 2.3: Example of an inverse sensor measurement model [23]

```

1:   Algorithm inverse_range_sensor_model( $m_i, x_t, z_t$ ):
2:     Let  $x_i, y_i$  be the center-of-mass of  $m_i$ 
3:      $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$ 
4:      $\phi = \text{atan2}(y_i - y, x_i - x) - \theta$ 
5:      $k = \text{argmin}_j |\phi - \theta_{j,\text{sens}}|$ 
6:     if  $r > \min(z_{\text{max}}, z_t^k + \alpha/2)$  or  $|\phi - \theta_{k,\text{sens}}| > \beta/2$  then
7:       return  $l_0$ 
8:     if  $z_t^k < z_{\text{max}}$  and  $|r - z_t^k| < \alpha/2$ 
9:       return  $l_{\text{occ}}$ 
10:    if  $r \leq z_t^k$ 
11:      return  $l_{\text{free}}$ 
12:    endif

```

Figure 2.4: Algorithm for an inverse measurement model of a range finder [23]

As an example, a simplistic model can be set up for a range finder, as shown in Figure 2.4. Here, α is the width of an obstacle, β is the width of the sensor beam, r is the range to the centre of mass, k is the beam index and ϕ is the heading to the measurement. The inverse sensor model varies depending on which sensor is used. Figure 2.3 shows the effect of an inverse sensor model. Here, the white grid cells represent free area, black represents occupied area and grey represents unexplored area.

2.5 Scan Matching

In reality, motion is quite noisy. This is something that can not be ignored as it leads to the failure of the assumed known poses. This, in turn, leads to a faulty map. The concepts presented here comes from a lecture by Cyrill Stachniss on *Grid Maps* [21]. Usually, sensors are quite precise and can be used to correct the pose estimate. Scan matching tries to incrementally align either two scans, or a map to a scan, without revisiting the past map. The overall goal of the scan matcher is to maximize the likelihood of the current pose based on the previous pose and the map knowledge.

$$x_t^* = \underset{x_t}{\operatorname{argmax}} \{p(z_i | x_{t-1}, m_{t-1}) p(x_t | u_{t-1}, x_{t-1}^*)\} \quad (2.30)$$

There exists a variety of different scan matching techniques. As there are too many methods to describe all of them in this section, some of the different ways to achieve scan matching will be listed instead:

- Iterative closest point (ICP)
- Scan-to-scan
- Scan-to-map
- Map-to-map
- Feature-based
- RANSAC (good for outlier rejection)
- Correlative matching

2.6 Particle Filters

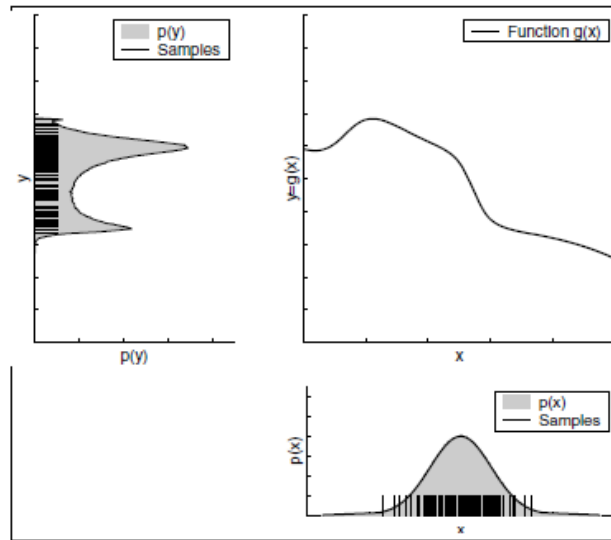


Figure 2.5: Particle distribution after passing through a non-linear function [23]

The idea of the particle filter, abbreviated PF, is to use randomly generated samples to represent the posterior of $bel(x_t)$. Such representation is approximate, but since the PF is non-parametric, it can represent much broader distributions than, for instance, a Gaussian. This can be illustrated in the following way: A robot was placed in an office in a typical office environment. The robot would be unsure of the specific office it was placed in, so it simply assumes it is placed in an office. In a Gaussian distribution the robot had to have a mean, μ , and a covariance, σ . The mean would typically be the exact placement of the robot and the covariance would be the uncertainty. In the Gaussian world, the robot would have to guess at which office it was placed in. However, when using the particle filter, the robot just assumes it is placed in an office. The robot would then drive along until further observations would confirm which exact office the robot was placed in. In other words, based on later observations, the robot could determine where it came from. This gives the robot an edge in repeating environments.

The samples of the posterior of the particle filter is called *particles*, and the particles are an hypothesis of what the world look like at time t . This is shown in Figure 2.5. In the lower right graph, samples are drawn from a Gaussian distribution. These samples passes through a non-linear function in the top right graph resulting in the top left graph of the particles distributed according to the random variable Y . The concepts and ideas presented here are gathered from chapter 4.3 *The Particle Filter* from the book *Probabilistic Robotics* [23] and the lecture *Short Introduction to Particle Filters and Monte Carlo Localization* by Cyrill Stachniss [22].

The particles are represented as a weighted set of samples

$$\chi = \{ \langle x_t^{[j]}, w_t^{[j]} \rangle \}_{j=1, \dots, J} \quad (2.31)$$

where

χ : set of weighted samples
 $x_t^{[j]}$: state hypothesis of sample j
 $w_t^{[j]}$: importance weight of sample j

and the sample posterior is represented as

$$p(x) = \sum_{j=1}^J w_t^{[j]} \delta_x^{[j]} \quad (2.32)$$

where

$\delta_x^{[j]}$: Dirac distribution at sample j

Ideally, the particles which is included in the true distribution should be kept in the set χ_t

$$x_t^{[j]} \sim p(x_t | z_{1:t}, u_{1:t}) \quad (2.33)$$

As a consequence, if there is a large subset of particles populated in a dense area, the more likely it is that the true state also lies here. This is only true if there is a high enough amount of particles to represent the distribution. Ideally there would be an infinite amount of samples to represent the distribution, but this is not realizable in practice.

Next, the importance sampling principle has to be explained. The key idea is that it is possible to use a different distribution g to generate samples from f . To account for the differences between g and f , a weighting is used.

$$w = \frac{f}{g} \quad (2.34)$$

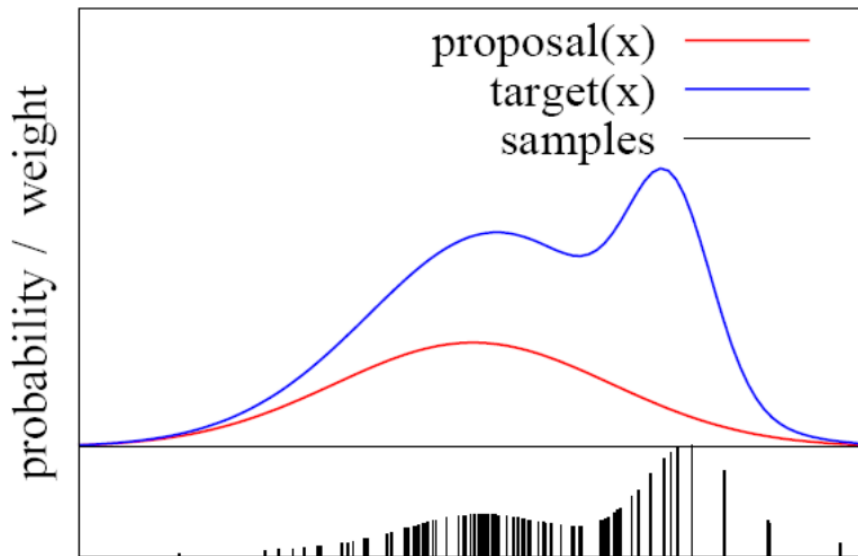


Figure 2.6: Target and proposal distribution with the resulting samples [22]

The purpose of this weighting factor is to correct the mismatch between f and g . This only works if $f(x) > 0 \rightarrow g(x) > 0$. This is a way to represent the *target distribution* $f(x)$ using the *proposal distribution* $g(x)$. This is illustrated in Figure 2.6.

The particle filter algorithm looks like this:

1. Sample particles using the proposal distribution

$$x_t^{[j]} \sim \pi(x_t | \dots)$$

2. Compute the importance weight

$$w_t^{[j]} = \frac{\text{target}(x_t^{[j]})}{\text{proposal}(x_t^{[j]})}$$

3. Resampling: Draw sample i with probability $w_t^{[i]}$ and repeat J times

```

Particle filter( $\mathcal{X}_{t-1}, u_t, z_t$ ):
1:    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
2:   for  $m = 1$  to  $M$  do
3:     sample  $x_t^{[m]} \sim \pi(x_t)$ 
4:      $w_t^{[m]} = \frac{p(x_t^{[m]})}{\pi(x_t^{[m]})}$ 
5:      $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
6:   endfor
7:   for  $m = 1$  to  $M$  do
8:     draw  $i$  with probability  $\propto w_t^{[i]}$ 
9:     add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
10:  endfor
11:  return  $\mathcal{X}_t$ 

```

Figure 2.7: Particle filter algorithm [22]

The particle filter algorithm is shown in Figure 2.7.

As for resampling, two approaches are going to be discussed here. One is the *roulette wheel* approach, and the other is the *stochastic universal sampling*.

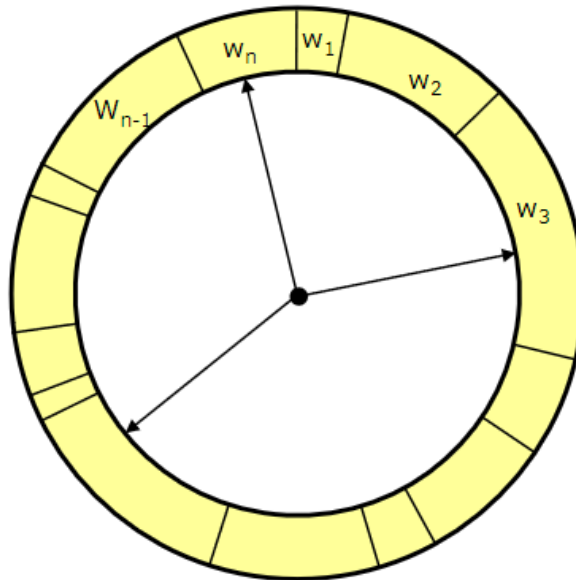


Figure 2.8: Roulette wheel approach to resampling [22]

Figure 2.8 shows an illustration of the roulette wheel approach. This is a binary search algorithm which selects one sample at a time. The higher the weight of the sample, the bigger is the slot

of the "roulette wheel". This is also shown in Figure 2.8 where the area for w_1 is significantly smaller than the area for w_2 . The binary search approach has a complexity of $O(J \log J)$.

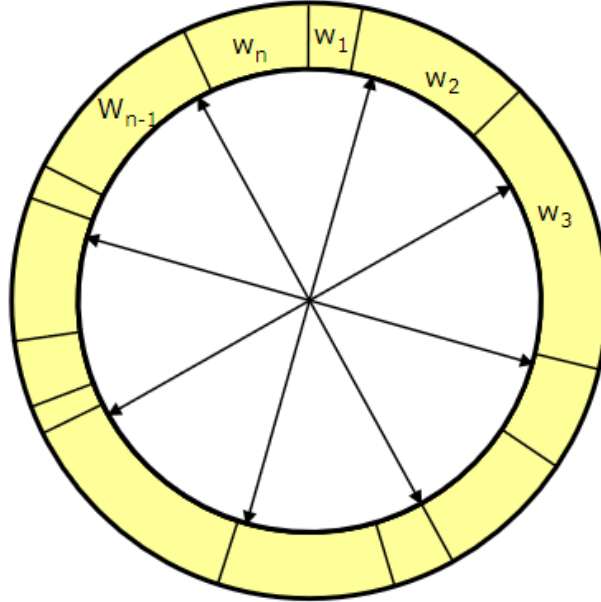


Figure 2.9: Stochastic universal sampling [22]

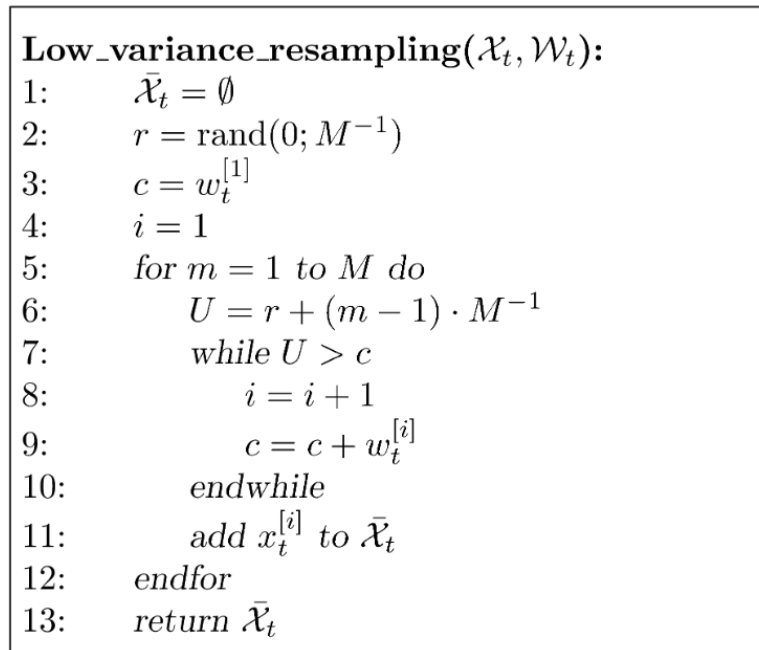


Figure 2.10: Algorithm for low variance resampling [22]

In contrast to the roulette wheel approach of the binary search, the stochastic universal sampling uses low variance sampling. Figure 2.9 illustrates this approach. Once again, the weights of

the samples determine how "large" of an area the sample get at the roulette table. However, instead of selecting one sample at the time, this roulette selects all the samples to be resampled in one turn. This gives this approach a complexity of $O(J)$. This approach has a much lower complexity than the binary search approach. Figure 2.10 shows the low variance resampling algorithm.

2.7 Monte Carlo Localization

Monte Carlo localization, abbreviated MCL, is a popular localization algorithm which uses a particle filter to represent the belief $bel(x_t)$. The MCL algorithm supports multi-modal distributions and is not bound to a limited parametric subset of distributions. This gives it the ability to represent complex multi-modal distributions and even blend them using Gaussian-style distributions. The concepts and ideas presented here are gathered from chapter 8.3 *Monte Carlo Localization* in the book *Probabilistic Robotics* [23].

Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):

- 1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
- 2: for $m = 1$ to M do
- 3: sample $x_t^{[m]} \sim p(x_t | u_t, x_{t-1}^{[m]})$
- 4: $w_t^{[m]} = \frac{p(z_t | x_t^{[m]})}{p(z_t | x_{t-1}^{[m]})}$
- 5: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
- 6: endfor
- 7: for $m = 1$ to M do
- 8: draw i with probability $\propto w_t^{[i]}$
- 9: add $x_t^{[i]}$ to \mathcal{X}_t
- 10: endfor
- 11: return \mathcal{X}_t

Figure 2.11: Monte Carlo localization algorithm [22]

The main difference of the MCL and the standard particle filter is how the proposal distribution and weighting is done. In contrast to the particle filter discussed in the previous section, the proposal distribution of the MCL uses the motion model $p(x_t | x_{t-1}, u_t)$. The weighting factor $w_t^{[j]}$ is calculated by the measurement model $p(z_t | x_t, m)$. Besides these two alterations, the algorithm works in the same way as the particle filter. The Monte Carlo localization algorithm is shown in Figure 2.11. In this figure, the red underline signifies the differences between the particle filter and Monte Carlo localization.

2.8 Kalman Filter

The Kalman filter, abbreviated KF, is a Bayes filter used on linear Gaussian systems. The belief at time t is represented by two parameters; the mean μ_t and the covariance Σ_t . The concepts and ideas presented here are gathered from chapter 3.2 *The Kalman Filter* in the book *Probabilistic Robotics* [23].

The Kalman filter assumes:

1. Gaussian distribution and noise
2. Linear motion and measurement model

$$\begin{aligned}x_t &= A_t x_{t-1} + B_t u_t + \varepsilon_t \\z_t &= C_t x_t + \delta_t\end{aligned}\tag{2.35}$$

In a nutshell, the Kalman filter has two steps; a prediction step and a correction step. The prediction step utilizes a linear variant of the motion model and added Gaussian noise to predict the states of the system.

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t\tag{2.36}$$

where

A_t : $n \times n$ square matrix, n is the dimension of state vector x_t

B_t : $n \times m$ matrix, m is the dimension of control vector u_t

ε_t : Gaussian random vector to model uncertainty in state transition

Equation (2.36) shows the prediction state of the Kalman filter. The random Gaussian state transition vector ε_t is used to model the uncertainty of the state transition and has a covariance, here denoted, R_t , and a mean μ at zero. Gaussian techniques share the idea that the belief can be represented by the multivariate normal distribution

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right\}\tag{2.37}$$

By applying equation (2.36) in the multivariate normal distribution, equation (2.37), the following state transition equation can be derived

$$p(x_t | u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right\}\tag{2.38}$$

As there is now an expression for the prediction state of the Kalman filter, the correction state has to be defined. The linear measurement model is used in the same way to get the final expression of the measurement probability $p(z_t | x_t)$.

$$z_t = C_t x_t + \delta_t \quad (2.39)$$

where

C_t : matrix of size $k \times n$, k is the dimension of measurement vector z_t

δ_t : measurement noise

δ_t is a multivariate Gaussian with zero mean and variance, here denoted, Q_t . Once more, by combining equation (2.39) and the multivariate normal distribution, equation (2.37), the following expression for the correction step is determined:

$$p(z_t | x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (z_t - C_t x_t)^T Q_t^{-1} (z_t - C_t x_t) \right\} \quad (2.40)$$

Finally, to ensure that the final belief is a Gaussian, the initial belief has to be Gaussian.

$$bel(x_0) = p(x_0) = \det(2\pi \Sigma_0)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (x_0 - \mu_0)^T \Sigma_0^{-1} (x_0 - \mu_0) \right\} \quad (2.41)$$

Lastly, the Kalman gain K_t has to be computed. This variable determines how much of the measurement z_t is going to be incorporated in the new state estimate. The Kalman gain is calculated by

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1} \quad (2.42)$$

where

$\bar{\Sigma}_t$: predicted covariance

The new μ_t is calculated using this equation

$$\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t) \quad (2.43)$$

where

$\bar{\mu}_t$: predicted mean

The new covariance is calculated by using

$$\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t \quad (2.44)$$

where

I : identity matrix

```

1:   Algorithm Kalman_filter( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$ ):
2:      $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$ 
3:      $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$ 
4:      $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$ 
5:      $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$ 
6:      $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$ 
7:     return  $\mu_t, \Sigma_t$ 

```

Figure 2.12: Kalman filter algorithm [23]

The full Kalman filter algorithm is shown in Figure 2.12.

2.9 Extended Kalman Filter

The difference between the Kalman filter and the extended Kalman filter, abbreviated EKF, is that the EKF uses local linearisation to handle non-linear distributions. Here, the assumption is that the state transition probability and measurement probability is governed by non-linear functions

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \quad (2.45)$$

$$z_t = h(x_t) + \delta_t \quad (2.46)$$

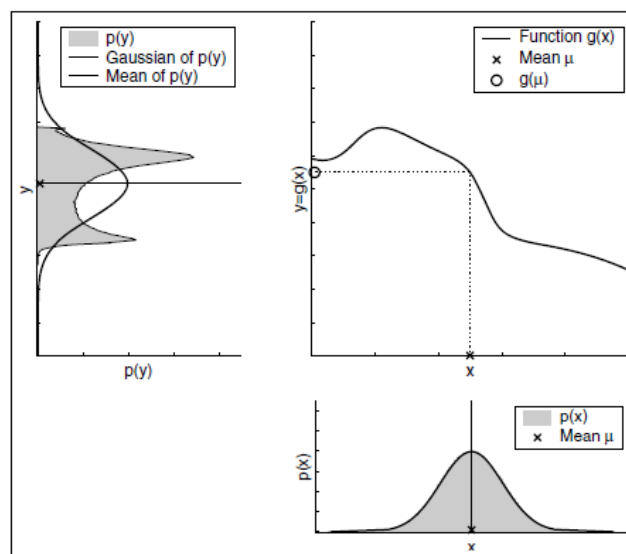


Figure 2.13: Extended Kalman filter linearisation [23]

The key idea of the EKF is to use the first order Taylor expansion to linearise the non-linear function at the mean of the Gaussian. The concepts and ideas presented here are gathered from chapter 3.3 *The Extended Kalman Filter* in the book *Probabilistic Robotics* [23]. This is illustrated in Figure 2.13. Note that the top left figure contains the "true" representation of the posterior of the non-linear distribution in grey. This distribution was constructed using a particle filter. In contrast to this "true" distribution, the Gaussian posterior of the EKF, with its mean, is shown in black.

The Taylor expansion makes a linear approximation of a non-linear function g from the value of the derivative of g , namely g' . g' is found by partially differentiating g

$$g'(u_t, x_{t-1}) = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \quad (2.47)$$

For Gaussians, the most likely state at the linearisation time is the posterior mean μ_{t-1} . g can therefore be approximated to its value μ_{t-1} and the control data u_t , giving

$$\begin{aligned} g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + \underbrace{g'(u_t, \mu_{t-1})}_{=: G_t} (x_{t-1} - \mu_{t-1}) \\ &= g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \end{aligned} \quad (2.48)$$

where

G_t : Jacobian of g

By writing this as a Gaussian, the transition probability is approximated as

$$\begin{aligned} p(x_t | u_t, x_{t-1}) &\approx \\ \det(2\pi R_t)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T \dots \right. & \quad (2.49) \\ \left. \dots R_t^{-1} [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})] \right\} & \end{aligned}$$

The same linearisation process is used to linearise the function h . The only difference is the linearisation point. Now $\bar{\mu}_t$ is used as the linearisation point as it is the most likely state of the robot at the time the linearisation is happening. This gives

$$\begin{aligned} h(x_t) &\approx h(\bar{\mu}_t) + \underbrace{h'(\bar{\mu}_t)}_{=: H_t} (x_t - \bar{\mu}_t) \\ &= h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \end{aligned} \quad (2.50)$$

where

$h'(t) : \frac{\partial h(x_t)}{\partial x_t}$
 H_t : Jacobian of h

giving the final expression

$$p(z_t | x_t) \approx \det(2\pi Q_t)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} [z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t)]^T Q_t^{-1} [z_t - h(\bar{\mu}_t) - H_t(x_t - \bar{\mu}_t)] \right\} \quad (2.51)$$

The Kalman gain is now computed by

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1} \quad (2.52)$$

The Kalman gain is used to calculate the mean and covariance of the Gaussian distribution

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)) \quad (2.53)$$

$$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t \quad (2.54)$$

```

1:   Algorithm Extended_Kalman_filter( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$ ):
2:      $\bar{\mu}_t = g(u_t, \mu_{t-1})$ 
3:      $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$ 
4:      $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$ 
5:      $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$ 
6:      $\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$ 
7:     return  $\mu_t, \Sigma_t$ 

```

Figure 2.14: Extended Kalman filter algorithm [23]

This results in the EKF algorithm shown in Figure 2.14.

2.10 Rao-Blackwellization

The key idea of Rao-Blackwellization is to use factorization to exploit dependencies between variables using the formula for *conditional probability*, which can be found in chapter 2.2 *Basic Concepts in Probability* in the book *Probabilistic Robotics* [23] at the top of page 16. The concepts and ideas presented here are gathered from chapter 13 *The FastSLAM Algorithm* in the book *Probabilistic Robotics* [23] and a lecture by Cyrill Stachniss called *FastSLAM – Feature-Based SLAM with Particle Filters* [19]. The formula for conditional probability states:

$$p(a | b) = \frac{p(a, b)}{p(b)} \quad (2.55)$$

$$\Rightarrow p(a, b) = p(b | a) p(a)$$

By applying this to the SLAM posterior, the following expression is found

$$p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) = \underbrace{p(x_{0:t} \mid z_{1:t}, u_{1:t})}_{\text{path posterior}} \underbrace{p(m_{1:M} \mid x_{0:t}, z_{1:t})}_{\text{map posterior}} \quad (2.56)$$

The map entries are conditionally independent of each other as long as the robot poses are known. This gives the expression

$$p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) = p(x_{0:t} \mid z_{1:t}, u_{1:t}) \prod_{i=1}^M p(m_i \mid x_{0:t}, z_{1:t}) \quad (2.57)$$

This expression can be exploited as the term containing the map posterior can be calculated for each individual map feature. This enables algorithms to update small pieces of the map, in contrast to updating the entire map at once.

2.11 Feature-Based FastSLAM 1.0 and 2.0

Before discussing the FastSLAM algorithm for occupancy grid maps, the feature-based approach is going to be accounted for first. Here, the FastSLAM 1.0 algorithm will be investigated and the differences between FastSLAM 1.0 and FastSLAM 2.0 discussed. The concepts and ideas presented here are gathered from chapter 13 *The FastSLAM Algorithm* in the book *Probabilistic Robotics* [23] and a lecture by Cyrill Stachniss called *FastSLAM – Feature-Based SLAM with Particle Filters* [19].

2.11.1 FastSALM 1.0 with Known Correspondence

The FastSLAM 1.0 utilizes a particle filter for the pose estimate of the robot and two-dimensional EKF's for the map features. The approach described in this section assumes known data association.

$$p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) = \underbrace{p(x_{0:t} \mid z_{1:t}, u_{1:t})}_{\substack{\text{Solved with particle filter} \\ \text{similar to MCL}}} \underbrace{\prod_{i=1}^M p(m_i \mid x_{0:t}, z_{1:t})}_{\substack{\text{Solved with two-dimensional} \\ \text{EKF's}}} \quad (2.58)$$

This gives the particles

$$Y_t^{[k]} = \langle x_t^{[k]}, \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, \dots, \mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]} \rangle \quad (2.59)$$

where

$[k]$: index of particle

$x_t^{[k]}$: path estimate for the robot of particle k

$\mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]}$: mean and covariance of the Gaussian for the n -th feature of particle k

When calculating the particle set Y_t at time t , the previous set Y_{t-1} at time $t-1$ is needed. The particle set from the previous time step Y_{t-1} is used with the new control data u_t , measurement z_t and the feature correspondence c_t to estimate the new particle set. The update for FastSLAM 1.0 is done in several steps

1. Extend the path posterior by sampling a new pose for each individual sample from the motion model

$$x_t^{[k]} \sim p(x_t | x_{t-1}^{[k]}, u_t) \quad (2.60)$$

2. Update features based on observation or no observation. This step is updating the mean $\mu_{n,t-1}^{[k]}$ and the covariance $\Sigma_{n,t-1}^{[k]}$ of the map features. This update has two states which depends on whether the feature is observed, $n = c_t$, or if the feature is unobserved, $n \neq c_t$. If the feature is unobserved, meaning $n \neq c_t$, the mean and covariance remains the same after the update.

$$\langle \mu_{n,t}^{[k]}, \Sigma_{n,t}^{[k]} \rangle = \langle \mu_{n,t-1}^{[k]}, \Sigma_{n,t-1}^{[k]} \rangle \quad (2.61)$$

However, if the feature is observed, meaning $n = c_t$, the feature is updated using the EKF.

$$\begin{aligned} & p(m_{c_t} | x_{1:t}, z_{1:t}, c_{1:t}) = \\ & \eta p(z_t | x_t, m_{c_t}, c_t) p(m_{c_t} | x_{1:t-1}, z_{1:t-1}, c_{1:t-1}) \end{aligned} \quad (2.62)$$

η is here a normalizer. The probability $p(m_{c_t} | x_{1:t-1}, z_{1:t-1}, c_{1:t-1})$ is a Gaussian represented by the mean $\mu_{n,t-1}^{[k]}$ and covariance $\Sigma_{n,t-1}^{[k]}$ which is already known. This, in turn, mean that the only thing needed to be calculated is $p(z_t | x_t, m_{c_t}, c_t)$. This can be approximated to

$$\begin{aligned} h(m_{c_t}, x_t^{[k]}) & \approx \underbrace{h(\mu_{c_t,t-1}^{[k]}, x_t^{[k]})}_{=: \hat{z}_t^{[k]}} + \underbrace{h'(x_t^{[k]}, \mu_{c_t,t-1}^{[k]})}_{=: H_t^{[k]}} (m_{c_t} - \mu_{c_t,t-1}^{[k]}) \\ & = \hat{z}_t^{[k]} + H_t^{[k]} (m_{c_t} - \mu_{c_t,t-1}^{[k]}) \end{aligned} \quad (2.63)$$

By using the same methodology as the EKF, the mean and covariance can be calculated. The Kalman gain has to be computed as to regulate the amount of sensor observation used in the new estimate.

$$K_t^{[k]} = \Sigma_{c_t,t-1}^{[k]} H_t^{[k]T} (H_t^{[k]} \Sigma_{c_t,t-1}^{[k]} H_t^{[k]T} + Q_t)^{-1} \quad (2.64)$$

$$\mu_{c_t,t} = \mu_{c_t,t-1} + K_t^{[k]} (z_t - \hat{z}_t^{[k]}) \quad (2.65)$$

$$\Sigma_{c_t,t}^{[k]} = (I - K_t^{[k]} H_t^{[k]}) \Sigma_{c_t,t-1}^{[k]} \quad (2.66)$$

3. Resample using importance weights. The importance weight factor $w_t^{[k]}$ is

$$w_t^{[k]} = \frac{\text{target}(x_t^{[k]})}{\text{proposal}(x_t^{[k]})} \quad (2.67)$$

The target distribution takes into account the latest measurements z_t and correspondence c_t giving the expression

$$p(x_{1:t}^{[k]} \mid z_{1:t}, u_{1:t}, c_{1:t}) \quad (2.68)$$

Assuming the set of particles Y_{t-1} being distributed as $p(x_{1:t-1} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1})$, the proposal distribution is

$$p(x_{1:t} \mid z_{1:t-1}, u_{1:t}, c_{1:t-1}) = p(x_t^{[k]} \mid x_{t-1}, u_t) p(x_{1:t-1}^{[k]} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) \quad (2.69)$$

This gives an importance weight factor of

$$\begin{aligned} w_t^{[k]} &= \frac{p(x_{1:t}^{[k]} \mid z_{1:t}, u_{1:t}, c_{1:t})}{p(x_{1:t}^{[k]} \mid z_{1:t-1}, u_{1:t}, c_{1:t-1})} \\ &= \eta p(z_t \mid x_t^{[k]}, c_t) \end{aligned} \quad (2.70)$$

The last expression in equation (2.70) is derived from the transformation of the numerator

$$\begin{aligned} &p(x_{1:t}^{[k]} \mid z_{1:t}, u_{1:t}, c_{1:t}) \\ &= \eta p(z_t \mid x_{1:t}^{[k]}, z_{1:t-1}, u_{1:t}, c_{1:t}) p(x_{1:t}^{[k]} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \\ &= \eta p(z_t \mid x_{1:t}^{[k]}, c_{1:t}) p(x_{1:t}^{[k]} \mid z_{1:t-1}, u_{1:t}, c_{1:t}) \end{aligned} \quad (2.71)$$

To calculate $p(z_t \mid x_t^{[k]}, c_t)$ from equation 2.70, further transformations are necessary.

$$\begin{aligned} w_t^{[k]} &= \eta \int p(z_t \mid m_{c_t}, x_t^{[k]}, c_t) p(m_{c_t} \mid x_t^{[k]}, c_t) dm_{c_t} \\ &= \eta \int p(z_t \mid m_{c_t}, x_t^{[k]}, c_t) \underbrace{p(m_{c_t} \mid x_t^{[k]}, c_t)}_{\sim \mathcal{N}(\mu_{c_t,t-1}^{[k]}, \Sigma_{c_t,t-1}^{[k]})} \end{aligned} \quad (2.72)$$

FastSLAM uses a linear approximation of equation 2.72, giving

$$w_t^{[k]} \approx \eta |2\pi Q_t^{[k]}|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (z_t - \hat{z}_t^{[k]})^T Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]}) \right\} \quad (2.73)$$

with covariance

$$Q_t^{[k]} = H_t^{[k]T} \Sigma_{n,t-1}^{[k]} H_t^{[k]} + Q_t \quad (2.74)$$

```

1: Algorithm FastSLAM 1.0_known_correspondence( $z_t, c_t, u_t, Y_{t-1}$ ):
2:   for  $k = 1$  to  $M$  do                                     // loop over all particles
3:     retrieve  $\langle x_{t-1}^{[k]}, \langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]} \rangle, \dots, \langle \mu_{N,t-1}^{[k]}, \Sigma_{N,t-1}^{[k]} \rangle \rangle$  from  $Y_{t-1}$ 
4:      $x_t^{[k]} \sim p(x_t | x_{t-1}^{[k]}, u_t)$                        // sample pose
5:      $j = c_t$                                                // observed feature
6:     if feature  $j$  never seen before
7:        $\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$                    // initialize mean
8:        $H = h'(x_t^{[k]}, \mu_{j,t}^{[k]})$                        // calculate Jacobian
9:        $\Sigma_{j,t}^{[k]} = H^{-1} Q_t (H^{-1})^T$              // initialize covariance
10:       $w^{[k]} = p_0$                                        // default importance weight
11:     else
12:        $\hat{z} = h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$                    // measurement prediction
13:        $H = h'(x_t^{[k]}, \mu_{j,t-1}^{[k]})$                  // calculate Jacobian
14:        $Q = H \Sigma_{j,t-1}^{[k]} H^T + Q_t$                  // measurement covariance
15:        $K = \Sigma_{j,t-1}^{[k]} H^T Q^{-1}$                    // calculate Kalman gain
16:        $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z})$        // update mean
17:        $\Sigma_{j,t}^{[k]} = (I - K H) \Sigma_{j,t-1}^{[k]}$      // update covariance
18:        $w^{[k]} = |2\pi Q|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (z_t - \hat{z}_n)^T Q^{-1} (z_t - \hat{z}_n) \right\}$  // importance factor
19:     endif
20:     for all other features  $j' \neq j$  do                 // unobserved features
21:        $\mu_{j',t}^{[k]} = \mu_{j',t-1}^{[k]}$                  // leave unchanged
22:        $\Sigma_{j',t}^{[k]} = \Sigma_{j',t-1}^{[k]}$ 
23:     endfor
24:   endfor
25:    $Y_t = \emptyset$                                        // initialize new particle set
26:   do  $M$  times                                         // resample  $M$  particles
27:     draw random  $k$  with probability  $\propto w^{[k]}$  // resample
28:     add  $\langle x_t^{[k]}, \langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} \rangle, \dots, \langle \mu_{N,t}^{[k]}, \Sigma_{N,t}^{[k]} \rangle \rangle$  to  $Y_t$ 
29:   endfor
30:   return  $Y_t$ 

```

Figure 2.15: FastSLAM 1.0 algorithm [23]

The FastSLAM 1.0 algorithm is depicted in Figure 2.15.

2.11.2 FastSLAM 2.0

FastSLAM 2.0 uses a lot of the same approaches and techniques as FastSLAM 1.0, but they mainly differ in the proposal distribution. The difference is that FastSLAM 2.0 utilizes the measurement z_t when sampling the pose x_t . This yields good results as the sensors are usually more accurate than the noisy motion model. As a result, FastSLAM 2.0 draws the pose $x_t^{[k]}$ from the posterior

$$x_t^{[k]} \sim p(x_t | x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) \quad (2.75)$$

This distribution takes the measurement z_t and correspondence c_t into consideration for the estimate of the posterior of the pose $x_t^{[k]}$, which leads to an increase in complexity of the mathematical deduction of the distribution. These derivations are based on chapter 13.4 *Improving the Proposal Distribution* from the book *Probabilistic Robotics* [23].

First, the expression in equation (2.75) is rewritten with known distributions, such as the motion and measurement models. This gives the expression

$$\begin{aligned} & p(x_t | x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) \\ \stackrel{\text{Bayes}}{=} & \frac{p(z_t | x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(x_t | x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t})}{p(z_t | x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t})} \\ = & \eta^{[k]} p(z_t | x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(x_t | x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \\ \stackrel{\text{Markov}}{=} & \eta^{[k]} p(z_t | x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(x_t | x_{t-1}^{[k]}, u_t) \\ = & \eta^{[k]} \int p(z_t | m_{c_t}, x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(m_{c_t} | x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) dm_{c_t} \dots \\ & \dots p(x_t | x_{t-1}^{[k]}, u_t) \\ \stackrel{\text{Markov}}{=} & \eta^{[k]} \int \underbrace{p(z_t | m_{c_t}, x_t, c_t)}_{\sim \mathcal{N}(z_t; h(m_{c_t}, x_t), Q_t)} \underbrace{p(m_{c_t} | x_{1:t-1}^{[k]})}_{\sim \mathcal{N}(m_{c_t}; \mu_{c_t, t-1}, \Sigma_{c_t, t-1})} dm_{c_t} \underbrace{p(x_t | x_{t-1}^{[k]}, u_t)}_{\sim \mathcal{N}(x_t; g(x_{t-1}^{[k]}, u_t), R_t)} \end{aligned} \quad (2.76)$$

As it stands, equation 2.76 has no closed form solution. This is due to the non-linear term h . By linearising the expression through a first order Taylor expansion, the terms becomes linear.

$$h(m_{c_t}, x_t) \approx \hat{z}_t^{[k]} + H_m(m_{c_t} - \mu_{c_t, t-1}^{[k]}) + H_x(x_t - \hat{x}_t^{[k]}) \quad (2.77)$$

where

$$\begin{aligned} \hat{z}_t & : h(\mu_{c_t, t-1}^{[k]}, \hat{x}_t) \\ \hat{x}_t & : g(x_{t-1}^{[k]}, u_t) \\ H_m & : \nabla_{m_{c_t}} h(m_{c_t}, x_t) \Big|_{x_t = \hat{x}_t^{[k]}; m_{c_t} = \mu_{c_t, t-1}^{[k]}} \\ H_x & : \nabla_{x_t} h(m_{c_t}, x_t) \Big|_{x_t = \hat{x}_t^{[k]}; m_{c_t} = \mu_{c_t, t-1}^{[k]}} \end{aligned}$$

By using this assumption, equation (2.76) is a Gaussian with the following parameters

$$\Sigma_{x_t}^{[k]} = \left[H_x^T Q_t^{[k]-1} H_x + R_t^{-1} \right] \quad (2.78)$$

$$\mu_{x_t}^{[k]} = \Sigma_{x_t}^{[k]} H_x^T Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]}) + \hat{x}_t^{[k]} \quad (2.79)$$

where

$$Q_t^{[k]} : Q_t + H_m \Sigma_{c_t, t-1}^{[k]} H_m^T$$

Further, the convolution theorem provides a closed form solution for this linearised approximation.

$$\mathcal{N}(z_t; \hat{z}_t^{[k]} + H_x x_t - H_x \hat{x}_t^{[k]}, Q_t^{[k]}) \quad (2.80)$$

Now, equation (2.76) is a product of equation (2.80) and the rightmost term in equation (2.76), the normal distribution $\mathcal{N}(x_t; \hat{x}_t^{[k]}, R_t)$. This product written in Gaussian form gives

$$p(x_t | x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) = \eta \exp \left\{ -P_t^{[k]} \right\} \quad (2.81)$$

$$P_t^{[k]} = \frac{1}{2} \left[(z_t - \hat{z}_t^{[k]} - H_x x_t - H_x \hat{x}_t^{[k]})^T Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]} - H_x x_t - H_x \hat{x}_t^{[k]}) \dots \right. \\ \left. \dots + (x_t - \hat{x}_t^{[k]})^T R_t^{-1} (x_t - \hat{x}_t^{[k]}) \right] \quad (2.82)$$

The mean and covariance of this Gaussian are equivalent to the minimum of $P_t^{[k]}$ and its curvature. They can be found by calculating the first and second derivative of $P_t^{[k]}$.

$$\frac{\partial P_t^{[k]}}{\partial x_t} = -H_x^T Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]} - H_x x_t - H_x \hat{x}_t^{[k]}) + R_t^{-1} (x_t - \hat{x}_t^{[k]}) \quad (2.83) \\ = (H_x^T Q_t^{[k]-1} H_x + R_t^{-1}) x_t - H_x^T Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]} - H_x \hat{x}_t^{[k]}) - R_t^{-1} \hat{x}_t^{[k]}$$

$$\frac{\partial^2 P_t^{[k]}}{\partial x_t^2} = H_x^T Q_t^{[k]-1} H_x + R_t^{-1} \quad (2.84)$$

The covariance $\Sigma_{x_t}^{[k]}$ is obtained by taking the inverse of the second derivative.

$$\Sigma_{x_t}^{[k]} = \left[H_x^T Q_t^{[k]-1} H_x + R_t^{-1} \right]^{-1} \quad (2.85)$$

The mean $\mu_{x_t}^{[k]}$ is found by setting the first derivative to zero.

$$\begin{aligned}
\mu_{x_t}^{[k]} &= \Sigma_{x_t}^{[k]} \left[H_x^T Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]} - H_x \hat{x}_t^{[k]}) + R_t^{-1} \hat{x}_t^{[k]} \right] \\
&= \Sigma_{x_t}^{[k]} H_x^T Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]}) + \Sigma_{x_t}^{[k]} \left[H_x^T Q_t^{[k]-1} H_x + R_t^{-1} \right] \hat{x}_t^{[k]} \\
&= \Sigma_{x_t}^{[k]} H_x^T Q_t^{[k]-1} (z_t - \hat{z}_t^{[k]}) + \hat{x}_t^{[k]}
\end{aligned} \tag{2.86}$$

Now both the mean $\mu_{x_t}^{[k]}$ and covariance $\Sigma_{x_t}^{[k]}$ of the proposal distribution is defined.

When it comes to updating the map features, as with FastSLAM 1.0, unobserved features are not updated. The posterior of these features are simply copied from the previous posterior. When a feature is observed, the update is more intricate as compared to FastSALM 1.0.

$$\begin{aligned}
& p(m_{c_t} \mid x_t^{[k]}, c_{1:t}, z_{1:t}) \\
&= \eta \underbrace{p(z_t \mid m_{c_t}, x_t^{[k]}, c_t)}_{\sim \mathcal{N}(z_t; h(m_{c_t}, x_t^{[k]}), Q_t)} \underbrace{p(m_{c_t} \mid x_{1:t-1}^{[k]}, z_{1:t-1}, c_{1:t-1})}_{\sim \mathcal{N}(m_{c_t}; \mu_{c_t, t-1}^{[k]}, \Sigma_{c_t, t-1}^{[k]})}
\end{aligned} \tag{2.87}$$

The non-linearity of h is once more a problem as it causes the posterior to be a non-Gaussian. This function, however can be linearised giving the expression

$$h(m_{c_t}, x_t) \approx \hat{z}_t^{[k]} + H_m(m_{c_t} - \mu_{c_t, t-1}^{[k]}) \tag{2.88}$$

This leads to the Gaussian

$$\begin{aligned}
& p(m_{c_t} \mid x_t^{[k]}, c_{1:t}, z_{1:t}) \\
&= \eta \exp \left\{ -\frac{1}{2} (z_t - \hat{z}_t^{[k]}) - H_m(m_{c_t} - \mu_{c_t, t-1}^{[k]}) Q_t^{-1} (z_t - \hat{z}_t^{[k]}) - H_m(m_{c_t} - \mu_{c_t, t-1}^{[k]}) \dots \right. \\
&\quad \left. \dots - \frac{1}{2} (m_{c_t} - \mu_{c_t, t-1}^{[k]}) \Sigma_{c_t, t-1} (m_{c_t} - \mu_{c_t, t-1}^{[k]}) \right\}
\end{aligned} \tag{2.89}$$

The new mean and covariance is found by using the standard EKF measurement update equations

$$K_t^{[k]} = \Sigma_{c_t, t-1}^{[k]} H_m^T Q_t^{[k]-1} \tag{2.90}$$

$$\mu_{c_t, t} = \mu_{c_t, t-1} + K_t^{[k]} (z_t - \hat{z}_t^{[k]}) \tag{2.91}$$

$$\Sigma_{c_t, t} = (I - K_t^{[k]} H_m) \Sigma_{c_t, t-1}^{[k]} \tag{2.92}$$

The last thing necessary is to compute the importance weight factor and resample. As with FastSLAM 1.0, the target distribution is

$$p(x_t \mid u_{1:t}, z_{1:t}, c_{1:t}) \quad (2.93)$$

but the proposal distribution is now given by the product

$$p(x_{1:t-1}^{[k]} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1}) p(x_t^{[k]} \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) \quad (2.94)$$

This gives an importance weight factor

$$\begin{aligned} w_t^{[k]} &= \frac{p(x_t \mid u_{1:t}, z_{1:t}, c_{1:t})}{p(x_t^{[k]} \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) p(x_{1:t-1}^{[k]} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1})} \\ &= \frac{p(x_t^{[k]} \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) p(x_{1:t-1}^{[k]} \mid u_{1:t}, z_{1:t}, c_{1:t})}{p(x_t^{[k]} \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) p(x_{1:t-1}^{[k]} \mid z_{1:t-1}, u_{1:t-1}, c_{1:t-1})} \\ &= \frac{p(x_{1:t-1}^{[k]} \mid u_{1:t}, z_{1:t}, c_{1:t})}{p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})} \\ &\stackrel{\text{Bayes}}{=} \eta \frac{p(z_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(x_{1:t-1}^{[k]} \mid u_{1:t}, z_{1:t-1}, c_{1:t})}{p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})} \\ &\stackrel{\text{Markov}}{=} \eta \frac{p(z_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})}{p(x_{1:t-1}^{[k]} \mid u_{1:t-1}, z_{1:t-1}, c_{1:t-1})} \\ &= \eta p(z_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) \end{aligned} \quad (2.95)$$

Further transformations gives the expression

$$\begin{aligned} w_t^{[k]} &= \eta \int p(z_t \mid x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(x_t \mid x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) dx_t \\ &\stackrel{\text{Markov}}{=} \eta \int p(z_t \mid x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(x_t \mid x_{t-1}^{[k]}, u_t) dx_t \\ &= \eta \int \int p(z_t \mid m_{c_t}, x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) p(m_{c_t} \mid x_t, x_{1:t-1}^{[k]}, u_{1:t}, z_{1:t-1}, c_{1:t}) dm_{c_t} \\ &\quad \int p(x_t \mid x_{t-1}^{[k]}, u_t) dx_t \\ &\stackrel{\text{Markov}}{=} \eta \int \underbrace{p(x_t \mid x_{t-1}^{[k]}, u_t)}_{\sim \mathcal{N}(x_t; g(\hat{x}_t^{[k]}, u_t), R_t)} \int \underbrace{p(z_t \mid m_{c_t}, x_t, c_t)}_{\sim \mathcal{N}(z_t; g(\mu_{c_t, t-1}^{[k]}, \Sigma_{c_t, t-1}^{[k]}), Q_t)} \\ &\quad \underbrace{p(m_{c_t} \mid x_{1:t-1}^{[k]}, u_{1:t-1}, z_{1:t-1}, c_{1:t-1})}_{\sim \mathcal{N}(m_{c_t}; \mu_{c_t, t-1}^{[k]}, \Sigma_{c_t, t-1}^{[k]})} dm_{c_t} dx_t \end{aligned} \quad (2.96)$$

This expression can be approximated to a Gaussian over measurements z_t by linearising g . The resulting Gaussian has mean \hat{z}_t and the covariance is

$$L_t^{[t]} = H_x^T Q_t^{-1} H_x + H_m \Sigma_{c_t, t-1}^{[k]} H_m^T + R_t \quad (2.97)$$

The importance weight factor for the k -th particle is given by

$$w_t^{[k]} = |2\pi L_t^{[t]}|^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (z_t - \hat{z}_t)^T L_t^{[t]-1} (z_t - \hat{z}_t) \right\} \quad (2.98)$$

As the importance weight factor now is calculated, the resampling process is done in the same way as FastSLAM 1.0. Despite the added complexity of FastSLAM 2.0, the advantages are better accuracy and fewer particles needed to represent and recreate the observed environment.

2.12 Grid-Based FastSLAM

```

1:   Algorithm FastSLAM_occupancy_grids( $\mathcal{X}_{t-1}, u_t, z_t$ ):
2:      $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
3:     for  $k = 1$  to  $M$  do
4:        $x_t^{[k]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[k]})$ 
5:        $w_t^{[k]} = \text{measurement\_model\_map}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$ 
5:        $m_t^{[k]} = \text{updated\_occupancy\_grid}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$ 
6:        $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[k]}, m_t^{[k]}, w_t^{[k]} \rangle$ 
7:     endfor
8:     for  $k = 1$  to  $M$  do
9:       draw  $i$  with probability  $\propto w_t^{[i]}$ 
10:      add  $\langle x_t^{[i]}, m_t^{[i]} \rangle$  to  $\mathcal{X}_t$ 
11:    endfor
12:    return  $\mathcal{X}_t$ 

```

Figure 2.16: FastSLAM algorithm for occupancy grid maps [23]

The grid-based FastSLAM uses the ideas from occupancy grid mapping, Monte Carlo localization and FastSLAM 1.0 and 2.0, which all have been discussed previously. The big advantage of the grid-based approach is that there is no need to predefine landmark features. The grid-based approach can model arbitrary types of environments, using sensor input. This algorithm is illustrated in Figure 2.16. One of the key ideas of this approach is that each particle contains its own map. Subsequently, each particle has to update their map using the *mapping with known poses* approach [23].

However, to improve the performance of the algorithm, a couple of approaches can be used. To improve the pose estimate, a scan matcher can be applied before the particle filter. This

method would maximize the likelihood of the current pose and map relative to the previous pose and map, giving

$$x_t^* = \underset{x_t}{\operatorname{argmax}} \{p(z_t | x_t, m_{t-1}) p(x_t | u_t, x_{t-1}^*)\} \quad (2.99)$$

Further improvements can be yielded by using a better proposal distribution. The paper *Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters* by Grisetti et al. [4] discusses such an optimal proposal distribution. The key idea here is that the sensors are usually more precise than the motion model. This was the same basic idea as FastSLAM 2.0 used when implementing the measurement z_t into the proposal distribution. The following distribution is a result of this:

$$p(x_t | m_{t-1}^{[i]}, x_{t-1}^{[i]}, z_t, u_t) = \frac{p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t)}{p(z_t | m_{t-1}^{[i]}, x_{t-1}^{[i]}, u_t)} \quad (2.100)$$

The co-author of the paper [4], Cyrill Stachniss, elaborates in more detail on how to sample from this distribution in the lecture *Grid-Based FastSLAM* given at University of Freiburg [20]. By defining the term $\tau(x_t)$ as the numerator of equation (2.100):

$$p(x_t | m_{t-1}^{[i]}, x_{t-1}^{[i]}, z_t, u_t) = \frac{\overbrace{p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t)}^{\tau(x_t)}}{p(z_t | m_{t-1}^{[i]}, x_{t-1}^{[i]}, u_t)} \quad (2.101)$$

and given

$$p(z_t | m_{t-1}^{[i]}, x_{t-1}^{[i]}, u_t) = \int p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t) dx_t \quad (2.102)$$

the following expression is found by combining equation (2.101) and (2.102)

$$p(x_t | m_{t-1}^{[i]}, x_{t-1}^{[i]}, z_t, u_t) = \frac{\tau(x_t)}{\int_{\{x_t | \tau(x_t) > \epsilon\}} \tau(x_t) dx_t} \quad (2.103)$$

where

$$\tau(x_t) : p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t)$$

By approximating a Gaussian for $\tau(x_t)$, $\tau(x_t) \simeq \mathcal{N}(\mu^{[i]}, \Sigma^{[i]})$, the following parameters of the Gaussian is found:

$$\mu^{[i]} = \frac{1}{\eta} \sum_{j=1}^K x_j \tau(x_t) \quad (2.104)$$

$$\Sigma^{[i]} = \frac{1}{\eta} \sum_{j=1}^K (x_j - \mu^{[i]})(x_j - \mu^{[i]})^T \tau(x_t) \quad (2.105)$$

where

x_j : points sampled around the result of the scan matcher

The integral can be approximated to the sum of discrete elements.

$$\int_{\{x_t | \tau(x_t) > \epsilon\}} \tau(x_t) dx_t \simeq \sum_{j=1}^K \tau(x_j) \quad (2.106)$$

As for calculating the importance weight factor $w_t^{[i]}$, it is given as

$$\begin{aligned} w_t^{[i]} &= \frac{\text{target}(x_i)}{\text{proposal}(x_i)} \propto \frac{p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t)}{\pi(x_t^{[i]} | m_{t-1}, x_{t-1}^{[i]}, z_t, u_t)} \dots \\ &\quad \dots \frac{p(x_{1:t-1} | z_{1:t-1}, u_{1:t-1})}{\pi(x_{1:t-1} | z_{1:t-1}, u_{1:t-1})} \\ &= \frac{p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t)}{\int p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t) dx_t} w_{t-1}^{[i]} \\ &= w_{t-1}^{[i]} \int p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t) dx_t \end{aligned} \quad (2.107)$$

By doing some approximations, the following is found

$$\begin{aligned} w_t^{[i]} &= w_{t-1}^{[i]} \int p(z_t | m_{t-1}^{[i]}, x_t) p(x_t | x_{t-1}^{[i]}, u_t) dx_t \\ &\simeq w_{t-1}^{[i]} \int_{\{x_t | \tau(x_t) > \epsilon\}} \tau(x_t) dx_t \\ &\simeq w_{t-1}^{[i]} \sum_{j=1}^K \tau(x_j) \end{aligned} \quad (2.108)$$

The next problem that needs to be handled is the problem with *particle depletion*. This problem consists of "useful" particles being taken away during resampling. The consequence of this, is that the "memory" of the particle filter is greatly decreased. However, the paper by Grisetti et al. [4] also covers this problem, and proposes an elegant solution. This is done by introducing *effective sample size*, η_{eff} .

$$\eta_{eff} = \frac{1}{\sum_{i=1}^N (\tilde{w}^{[i]})^2} \quad (2.109)$$

where

$\tilde{w}^{[i]}$: normalized weight of particle i

Resampling is only done if η_{eff} drops below a certain threshold, for instance $\frac{N}{2}$. A decrease of η_{eff} is caused by a big difference in the particle weight. In other words, resampling is only done when the weights of the particles differ significantly.

Chapter 3

Method

Now the necessary theory needed to understand the problems faced in this thesis has been presented. This section of the thesis serves to explain the working method of the project and justify the selection of SLAM algorithms. In addition, the set-up of both the Robot Operating System workspace, and all necessary software, and the SPURV robot is going to be accounted for. Lastly, the different tests performed to verify the performance of the SLAM algorithms are detailed. This includes the main objective of each individual test and how to set up and execute the tests.

3.1 Working Method

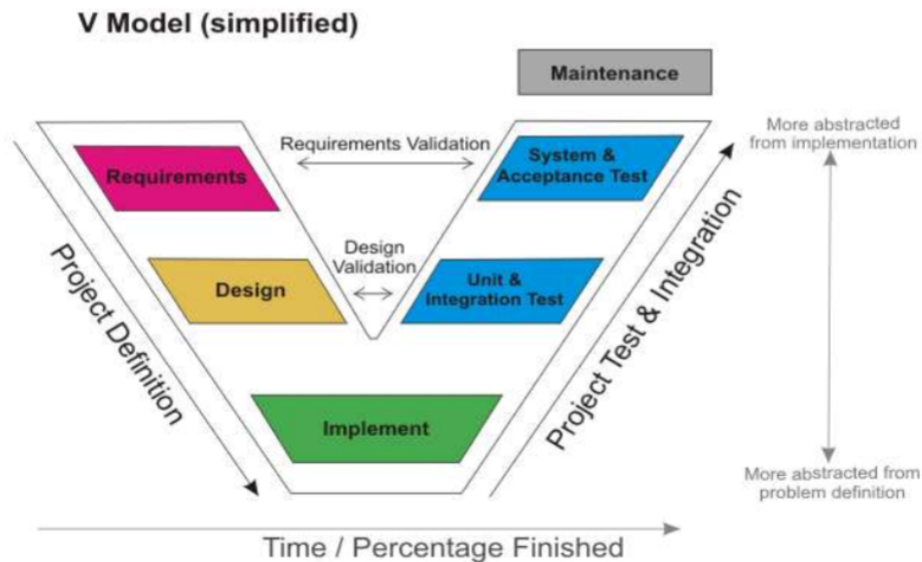


Figure 3.1: Illustration of the V-model working method [11]

In this thesis, the V-model working method is going to be used. This approach is illustrated in Figure 3.1. The method consists of five different stages during the duration project. In the

early stage, the requirements of the project is discussed and settled upon. These are going to be the basis of what the final product is evaluated by. Next, the design phase starts. Here, the systems and subsystems are researched and developed. This is followed by the implementation process, where the designed systems are going to be implemented on the hardware. Lastly, the entire system is going to be tested and evaluated based on the requirements settled upon in the requirement phase. It is important to use sufficient time for testing at the end of the project to ensure the quality of the developed solution.

3.2 Selection of SLAM Approach

To be able to make an informed decision of which SLAM algorithm to implement, several different approaches were researched and evaluated. Concerning the repeating environment, multi-modal assumptions are favourable and for best possible localization, non-linear motion models were highly desired. Multi-modal properties gives the ability to assume the placement of the robot in several places at once. This is advantageous in repeating environments where location ambiguities are likely to occur. In addition, the filter needed to be able to handle large maps and preferably using grid maps to represent arbitrary environments.

The reasoning for using grid maps is that there would be few indistinguishable landmarks or features in a tunnel. Constructing a map using these features, and distinguishing between them, would be infeasible due to the sheer amount of features. Especially in tunnels with jagged walls. Here there would be a lot of features and the robot would have to re-observe the feature from the approximate same height each time. The big advantage of grid based maps over feature based maps is that the grid based maps can use the raw sensor input coming from the laser range finder to construct the map. The feature based maps would need some kind of pre-work to define, then locate, the different landmarks in the environment. Typical landmarks used in this form of mapping would be corners and jump edges. Jump edges are points in which the measurement value jumps or decreases significantly in magnitude from the previous measurement point to the next. Additionally, corners might be difficult to distinguish from each other when there are a lot of them in a small area.

The three main filter types considered was the Bayes filters, particle filters and graph-based filters. They all have strengths and weaknesses, which are going to be presented.

In the Bayes filter family, the Kalman filter, extended Kalman filter, unscented Kalman filter, extended information filter and sparse extended information filter were considered. The filters are classified as a family as they all assume a Gaussian distribution and shares the basic idea that the belief can be represented by multivariate normal distributions. Multivariate distributions are not suited to handle at large maps, with the exception of the sparse extended information filter, due to the growth of the covariance matrix. These filters are easy to implement and are computationally cheap compared to other filters. However, none of these filters are supporting grid based mapping well nor support multi-modal assumptions. As a result of this, they were all discarded as possible solutions.

The next set of filters evaluated were the particle filter family. In this family, both feature maps and grid maps could be constructed. The FastSLAM 1.0 and FastSLAM 2.0 were considered, but ultimately came short compared to a technique using Rao-Blackwellized particle filter

and an improved proposal distribution. This approach, as described by the paper *Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters* [4], seemed to address all of the desired properties which were requested.

Lastly, the graph-based approaches were investigated. In this approach, constraints between the different robot poses were set up, and least squares minimization was used to determine the optimal pose estimate. After the poses are known, the map could be constructed using *mapping with known poses* and the measurement data. Utilizing *robust least squares SLAM with max-mixtures*, outliers in the data set could be handled properly and multi-modal assumptions could be supported.

In the end, there were two obvious solutions for the the SLAM in repeating environments problem; the Rao-Blackwellized particle filter and the robust least squares with max-mixture. Both methods supported grid based mapping, multi-modal assumptions and non-linearities. The particle filter, using sensor data to generate an improved proposal distribution, gave good results in a featureless corridor, as shown in the paper [4]. In this illustration, the robot was driving down a straight featureless corridor and the particle distribution was spread alongside the main direction of the corridor. This feature would be advantageous when navigating in a tunnel, which can be quite similar to a long featureless corridor. On the basis of this, and the aforementioned properties, the particle filter was chosen as the best solution to the SLAM in repeating environments problem.

3.3 Software

The main development platform in this thesis was *Robot Operating System*, abbreviated ROS. The *Kinetic* version of ROS was utilized in this thesis. ROS runs primarily on Linux based systems. As a consequence, *Ubuntu 16.04 LTS* was installed as a dual boot operating system, in conjunction with Windows 10, to be able to run ROS properly.

3.4 GMapping

The source code for the SLAM approach detailed in the paper from Grisetti et al. [4] was available at OpenSLAM.org [3] called *GMapping*. However, this code was developed for the CARMEN system and would not work with the ROS system. An open source project of GMapping was made for ROS at GitHub.com [2] [5]. **GMapping** utilizes the Rao-Blackwellized particle filter as described in the Theory chapter. To be able to create a local copy of the `slam_gmapping` files, the command

```
$ git clone https://github.com/ros-perception/slam_gmapping.git <my_path>
```

was utilized. By inserting this command in a terminal in Ubuntu, a local clone of the repository was created. Here, the command "git clone https://github.com/ros-perception/slam_gmapping.git" clones the repository found in the URL to a local folder. The extension "<my_path>" is optional, and used if the user wants to clone the repository to a certain folder. The

folder is specified by replacing the entire term "<my_path>" with the path of the desired folder.

Similarly, to get the `openslam_gmapping` files, the same command was used only the url was changed:

```
$ git clone https://github.com/ros-perception/openslam_gmapping.git <my_path>
```

Now, as both `openslam_gmapping` and `slam_gmapping` were available, a ROS package could be developed. Lastly, for managing and saving maps, the `map_server` package was used. The git clone approach was used once again, giving the command

```
$ git clone https://github.com/ros-planning/navigation.git <my_path>
```

3.5 Hector SLAM

The `hector_slam` [8] approach is described by the paper *A flexible and scalable SLAM system with full 3D motion estimation* [7] by Kohlbrecher et al. This technique utilizes a laser range finder and has no need for odometry data. This enables the algorithm to work with only a laser range finder. In contrast to `GMapping`, the `hector_slam` uses only a scan matcher for the 2D pose correction. On the other side, this simplicity makes the algorithm quick, enabling it to make use of the quick update rate of newer Lidar systems. However, as the paper [7] states, the algorithm is only useful in small scale scenarios and in cases where large loops does not have to be closed. This algorithm can be obtained by using the command

```
$ sudo apt-get install ros-kinetic-hector-slam
```

in a terminal. As the `hector_slam` package was installed, the `hector_mapping` package was used for SLAM, `hector_geotiff` was used to save and store the maps and `hector_trajectory_server` was used to save the trajectory of the SPURV.

3.6 Why GMapping and Hector SLAM?

It was decided to compare the performance of `GMapping` and `hector_slam` in areas with repeating elements and few features based on their popularity amongst the ROS community. In addition, it was interesting to investigate how only a scan matcher, which `hector_slam` is based on, would fare against the complex Rao-Blackwellized particle filter utilized by `GMapping`.

3.7 The SPURV Robot

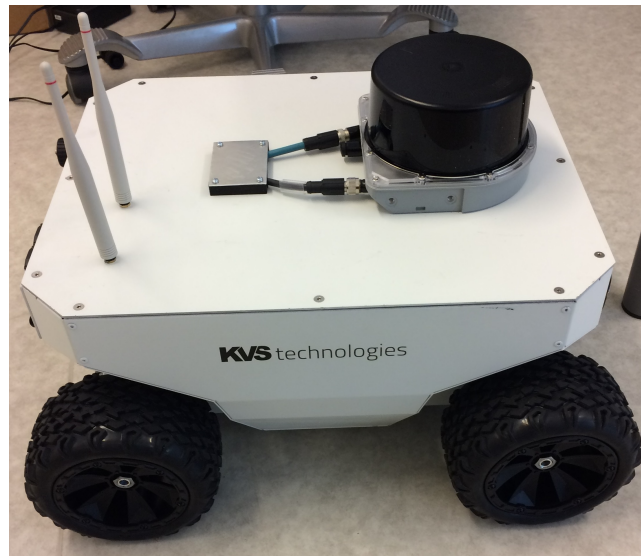


Figure 3.2: SPURV Research used in thesis

The SPURV Research robot used in this thesis is shown in Figure 3.2. This comes equipped with a SICK MRS1000 Lidar, which can be seen as the black cylinder on the top of the robot in the figure. It is also equipped with one front facing and one rear facing camera. The SPURV has an on-board router, which can broadcast its own wireless network using the antennae seen at the top rear of the SPURV in Figure 3.2.

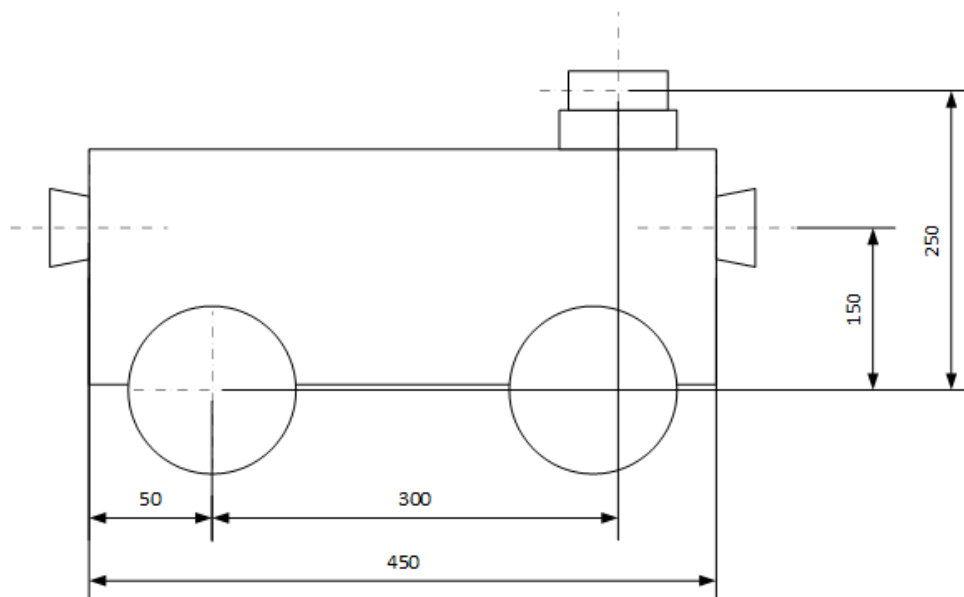


Figure 3.3: Sketch showing the placement of sensors on the SPURV

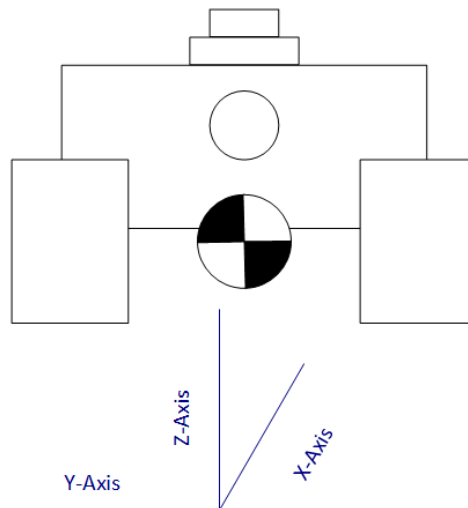


Figure 3.4: Sketch showing the coordinate system used on the SPURV, as seen from the rear

The placement of the sensors on the SPURV is shown in Figure 3.3. Here, both front and rear cameras are in the same height of 150 mm over the rear differential. The rear differential is used as the basis of the coordinate system, as this is seen as the most sensible placement of the coordinate frame when concerning Ackerman steered vehicles. The x-axis is placed along the driving axis and the z-axis is placed orthogonal to the ground plane, pointing up. This is illustrated in Figure 3.4 which shows the SPURV from the rear.

3.8 Test 1 - Initial Testing

3.8.1 Test Plan for Test 1

The first test was designed to test different approaches for simultaneous localization and mapping. The two main algorithms that were to be tested was `GMapping` and `hector_slam`, which have both been previously described. The testing environment consisted of large corridors found on campus Grimstad at the University of Agder. The main idea of this test was to drive the SPURV down long, relative featureless corridors which were connected in a loop. This would test both the SLAM capabilities as well as the loop closing capabilities of each algorithm. Additionally, this test would verify whether the algorithms could handle larger maps.

The SPURV was going to be controlled by a remote computer over the wireless network broadcasted by the SPURV. A wireless Xbox 360 controller was going to be used to control the motion of the SPURV. A wireless adapter connected to the remote computer could be used to transfer the signals from the wireless controller to the computer. The test was going to be conducted on a weekend. This was to reduce the chance of people walking around in the testing area. Several walking people introduce dynamic obstacles during testing, which was undesired. The SPURV was supposed to drive several laps in the corridor loop. This would make it easier to identify whether the algorithms were able to close the loop and identify the previously discovered area or if the maps would seem to "spiral" as the robot turning in the corridors would cause a drift in the mapping process.

To ensure that each algorithm got the same set of data, the **rosvbag** feature in ROS was going to be used. This feature records all the data produced by the SPURV while driving the laps. The recording could be played back on a later date and would produce the same data as if the SPURV was driving live. The advantage of this approach is that the SPURV did not have to be present after recording the data set. Each of the algorithms would perceive the data coming from the **rosvbag** as if it was happening in real time.

3.8.2 Execution of Test 1



(a) Starting Point for Test 1



(b) Long Hallway, accessed by turning left from starting point



(c) Continuation of the Long Hallway



(d) Narrow Hallway, accessed by turning left three quarters of the way through Long Hallway

Figure 3.5: Depiction of the route used to perform Test 1



(a) Open Area 1, accessed by turning left in Narrow Hallway

(b) Open Area 2, accessed by continuing driving straight from Open Area 1



(c) End Point of the route, accessed by driving to the end of Open Area 2 and turning left

Figure 3.6: Continuation of the route used in Test 1

Figure 3.5 and Figure 3.6 depicts the route used in Test 1. The SPURV robot would start at the Starting Point, Figure 3.5 a), and driving straight ahead and then turning left. This would lead the SPURV robot down the Long Hallway shown in Figure 3.5 b) and Figure 3.5 c). At three quarters of the way through the Long Hallway the SPURV would turn left, entering the Narrow Hallway shown in Figure 3.5 d). Here, by driving the SPURV to the end of the Narrow

Hallway and then turning left, the Open Area would be accessed, as shown in Figure 3.6 a). By driving to the end of the Open Area, Figure 3.6 a) and Figure 3.6 b), the End Point was reached by turning to the left at the end of Open Area, as shown in Figure 3.6 c). The End Point and the Starting Point is the same place, which means the robot could take several laps of the same route if desired.

The test was performed using the wireless network produced by the SPURV to transmit data from the sensors on board the SPURV. The remote computer was placed in the cubicle visible to the right in Figure 3.5 a). The wireless Xbox controller was connected to the remote computer using a wireless adapter and used to control the motion of the SPURV.

```
# spurv broadcasting by it self
function spurv_master_in_wild() {
    export SPURV=10.1.8.5
    export ROS_MASTER_URI=http://$SPURV:11311
    export ROS_IP="$(hostname -I | awk '{print $1}')"
    export ROS_HOSTNAME=$ROS_IP
}
```

Figure 3.7: Function in .bashrc to set correct the IP-addresses to the SPURV

The connection to the SPURV was achieved by connecting to the network "spurv", which would start up as the SPURV robot was turned on. Once connected to the "spurv" network, the function "spurv_master_in_wild" was used in every terminal to set the correct IP-addresses. This function can be seen in Figure 3.7. To check if a connection was established, the command

```
$ rostopic list
```

was entered in a terminal. This would show all produced topics. Topics are one way ROS publishes and subscribes to data. If the topics coming from the SPURV showed up, a successful connection had been established. To be able to use the Xbox controller, the program `xboxdrv` was used. This had to be started in a separate terminal to get access to the signals coming from the wireless controller. This was done by running the command

```
$ sudo xboxdrv
```

in a separate terminal. As the connection to the SPURV and the controller was set up, the program "joystick_example.py" was used to control the SPURV via the wireless Xbox controller. This program came with the SPURV and was copied from the SPURV to the remote computer. The program was started using the command

```
$ roslaunch spurv_examples joystick_example.launch
```

in a new terminal.

The remote computer was set up to record the data produced by the SPURV as it was driving along the test route. The sensor data was recorded using the `rosbag` package in ROS. This was done at the remote computer using the commands

```
$ mkdir -p bagfiles/test_1
```

to create the folder "bagfiles" with the sub-folder "test_1". Then the command

```
$ cd bagfiles/test_1
```

was used to get to the desired directory, and then the command

```
$ rosbag record -a
```

resulted in a data set, which is called "bag" in ROS, being recorded with all the data produced by the SPURV in the desired directory.

3.9 Test 2 - Controlling SPURV, Identify Topics

3.9.1 Test Plan for Test 2

The main goal of the second test was similar to the test described in chapter 3.8.1. However, where the test differed from the previous test was in how the SPURV was going to be driven and controlled. Another aspect of the test was how the data recorded by the `rosbag` was done. In the first test, the SPURV was controlled with a wireless Xbox controller for the motion commands and a camera gave primary visual feedback of the scene. As the camera feedback proved to be inefficient at greater distances, it was concluded that the operator was going to be walking behind the SPURV while it performed the required test route. This would be realized by putting the remote computer, which controls the SPURV, in a backpack and walking behind the SPURV. It was crucial that the operator did not enter the range of the laser scanner, as this would influence the data gathered from the test, possibly altering the end result.

Additionally, when recording the desired data, or "topics" which they are called in ROS, it was important to select the topics which were relevant for the testing purpose. This could be done by setting restrictions when recording the `rosbag`. By utilizing the command

```
$ rosbag record /topic1 /topic2 <etc.>
```

, the topics "topic1" and "topic2" and any additional desired topics would be recorded. For this test, the topics needed to to play back the testing route in `rviz` were going to be discovered. The package `pointcloud_to_laserscan` should be tested to transform the data from the point cloud topic to the laser scan topic as both `GMapping` and `hector_slam` required the topic `"/scan"` to work.

Besides these two changes from the first test, the test was going to be performed in the same environment as the first test. The goal was ultimately to verify the performance of **GMapping** and **hector_slam** in a repeating environment using data collected in the **rosbag**. The sub goals of the test was to reliably control the SPURV robot in the narrow hallways and use **pointcloud_to_laserscan** to transform the `"/cloud"` topic into the `"/scan"` topic. Additionally, the topics needed to play back the recorded data in **rviz** were to be discovered.

3.9.2 Execution of Test 2

In the second test, the route of the first test was reused. This route is described in more detail in chapter 3.8.2 and Figure 3.5 and Figure 3.6. Contrary to the first test, in the second test, the remote computer was placed in a backpack before driving the SPURV with the wireless Xbox controller.

The connection to the SPURV was achieved by connecting to the network "spurv", which would start up as the SPURV robot was turned on. Once connected to the "spurv" network, the function "spurv_master_in_wild" was used in each terminal who needed to communicate with the SPURV to set the correct IP-addresses. To check if the connection to the SPURV was established, the command

```
$ rostopic list
```

was ran. This would show all topics available, and if the topics coming from the SPURV showed up, a successful connection had been established. To be able to use the Xbox controller, the program **xboxdrv** was used. This had to be started in a separate terminal to get access to the signals coming from the wireless controller. This was done by running the command

```
$ sudo xboxdrv
```

in a new terminal. As the connection to the SPURV and the controller was set up, the program "joystick_example.py" was used to control the SPURV via the wireless Xbox controller. This program came with the SPURV, and had to be copied from the SPURV to the remote computer. The program was started using the command

```
$ roslaunch spurv_examples joystick_example.launch
```

in a new terminal.

To find which topics were necessary to reproduce the test run in **rviz**, several different runs of the route was performed. To check the results in **rviz**, several panels had to be added. The following panels were added to see validate the results of the test: "Map" panel with "Topic" set to `"/map"`, "TF" panel, "PointCloud2" with "Topic" set to `"/cloud"` and "LaserScan" panel with "Topic" set to `"/scan"`. In the first run, the topics `"/cloud"`, `"/odom"` and `"/tf"` were recorded. This configuration of **rviz** was saved and used in subsequent tests.

The data collected for the testing in **rviz** was done by using the commands

```
$ mkdir -p bagfiles/test_2
```

```
$ cd bagfiles/test_2
```

```
$ rosbag record -O first_run /cloud /odom /tf
```

in a terminal. These commands would create the folder "test_2", change the directory to "~/bagfiles/test_2" and then record the topics "/cloud", "/odom" and "/tf" to this directory. The "-O first_run" sets the name of the bagfile to "first_run". These data were checked in `rviz` to check if the necessary topics were present by starting `rviz`

```
$ rviz
```

and then playing the bag

```
$ rosbag play first_run
```

In the second run, the topics "/cloud", "/odom", "/tf" and "/tf_static" were recorded using the command

```
$ rosbag record -O second_run /cloud /odom /tf /tf_static
```

and a third control run was recorded using

```
$ rosbag record -O third_run /cloud /odom /tf /tf_static
```

For each run, once the scripts to control the SPURV with the wireless controller and the `rosbag` recording was started, the lid of the computer was shut close. The computer was quickly placed in the backpack and the SPURV was driven through the testing route. As the front facing camera was not used during the test, the resolution was set low to a resolution of 40×60. This was to minimize use of unnecessary bandwidth of the wireless network produced by the SPURV on transferring data coming from the camera feed.

```
# restore connection to your computer
function spurv_local_power() {
    export ROS_MASTER_URI=http://127.0.0.1:11311
    export ROS_IP=127.0.0.1
    export ROS_HOSTNAME=$ROS_IP
}
```

Figure 3.8: Function in `.bashrc` to set the IP addresses back to the local computer

When the necessary data from the SPURV was collected, the rest of the testing was done on the remote computer. To be able to run ROS once the SPURV was turned off, the function "spurv_local_power" had to be run. This function set the ROS IPs back to the local IPs of the computer, and is shown in greater detail in Figure 3.8. Additionally, the ROS master had to be activated in a separate terminal using the command

```
$ roscore
```

The data collected in the `rosbags` were used to test out the `pointcloud_to_laserscan` algorithm. This algorithm transforms the `/cloud` topic into the `/scan` topic, which both `GMapping` and `hector_slam` needed. This algorithm was obtained by using the command

```
$ sudo apt-get install ros-kinetic-pointcloud_to_laserscan
```

```
<?xml version="1.0"?>
<launch>

  <!-- run pointcloud_to_laserscan node -->
  <node pkg="pointcloud_to_laserscan" type="pointcloud_to_laserscan_node" name="pointcloud_to_laserscan">

    <remap from="cloud_in" to="/cloud"/> #$(arg camera)/depth_registered/points_processed
    <remap from="scan" to="/scan"/> #$(arg camera)/scan
    <rosparam>
      target_frame: laser # Leave disabled to output scan in pointcloud frame, original "camera_link"
      transform_tolerance: 0.01
      min_height: 0.0
      max_height: 2.0

      angle_min: -1.5708 # -M_PI/2, original -1.5708
      angle_max: 1.5708 # M_PI/2, original 1.5708
      angle_increment: 0.004363 # M_PI/360.0
      scan_time: 0.02
      range_min: 0.2
      range_max: 60.0
      use_inf: true

      # Concurrency level, affects number of pointclouds queued for processing and number of threads used
      # 0 : Detect number of cores
      # 1 : Single threaded
      # 2->inf : Parallelism level
      concurrency_level: 1
    </rosparam>
  </node>
</launch>
```

Figure 3.9: Launch file used in test 2 for `pointcloud_to_laserscan` algorithm in ROS

The launch file for this algorithm was located in the directory `/opt/ros/kinetic/share/pointcloud_to_laserscan/launch` and the launch file "sample_node.launch" was altered to the one shown in Figure 3.9. The advantage of using a launch file was that the parameters of the script were stored in the file. When starting up with the launch file, these parameters were loaded as well.

The `pointcloud_to_laserscan` package had to be started before `GMapping` and `hector_slam` could be used. This was due to the topic `/cloud` had to be transformed into the `/scan` topic beforehand. `pointcloud_to_laserscan` was started using the command

```
$ roslaunch pointcloud_to_laserscan sample_node.launch
```

in a new terminal.

```

<launch>
  <arg name="scan_topic" default="/scan" />
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="odom_frame" value="odom"/>
    <param name="base_frame" value="base_link"/>
    <param name="map_frame" value="map"/>

    <!-- Process 1 out of every this many scans (set it to a higher number to skip more scans) -->
    <param name="throttle_scans" value="1"/>

    <param name="map_update_interval" value="1.0"/> <!-- default: 5.0 -->

    <!-- The maximum usable range of the laser. A beam is cropped to this value. -->
    <param name="maxUrange" value="30.0"/>

    <!-- The maximum range of the sensor. If regions with no obstacles within the range of the sensor should appear as free space in the map, set maxUrange to this value. -->
    <param name="maxRange" value="60.0"/>

    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="minimumScore" value="0.0"/>
    <!-- Number of beams to skip in each scan. -->
    <param name="lskip" value="0"/>

    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>

    <!-- Process a scan each time the robot translates this far -->
    <param name="linearUpdate" value="0.1"/>

    <!-- Process a scan each time the robot rotates this far -->
    <param name="angularUpdate" value="0.05"/>

    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>

    <!-- Number of particles in the filter. default 30 -->
    <param name="particles" value="10"/>

  <!-- Initial map size -->
  <param name="xmin" value="-50.0"/>
  <param name="ymin" value="-50.0"/>
  <param name="xmax" value="50.0"/>
  <param name="ymax" value="50.0"/>

  <!-- Processing parameters (resolution of the map) -->
  <param name="delta" value="0.02"/>

  <param name="lssamplerange" value="0.01"/>
  <param name="lssamplestep" value="0.01"/>
  <param name="lasamplerange" value="0.005"/>
  <param name="lasamplestep" value="0.005"/>

</node>
</launch>

```

Figure 3.10: Launch file used in test 2 for GMapping in ROS

The launch file used to test **GMapping** is shown in Figure 3.10. The launch file was based on the *jackal gmapping launch file* [6], and was named "slam_gmapping_experimental.launch". The initial map size was set to 100×100 meters with a resolution of 2 cm per grid cell. To start **GMapping**, the command

```
$ rosparam set use_sim_time true
```

was executed followed by the command

```
$ roslaunch gmapping slam_gmapping_experimental.launch
```

in a new terminal. Once the **rosvbag** was completed, the commands

```
$ mkdir map
```

```
$ cd map
```

and then the command

```
$ rosrn map_server map_saver -f test_2
```

was run in a new terminal to save the generated map with the name "test_2" in the directory `~/map`. After this was done, the terminal running **GMapping** had to be shut down using `ctrl+c` and restarted using the command

```
$ roslaunch gmapping slam_gmapping_experimental.launch
```

or simply scrolling through previously used commands using the up and down arrow keys. This was needed if another **rosbag** was to be tested. The map from the next **rosbag** could be saved using the same **map_server** command as previously mentioned.

3.10 Fixing the Odometry of the SPURV

The second test reviled an error with the odometry of the SPURV. The error was evident when further error testing was made. When the SPURV was driving forwards, the odometry claimed it was driving backwards. It was suspected that the error was due to a set up error in the motor controller, or that the wrong cables were connected to the wrong poles of the motor. This error was in reality one and the same, and the only question was whether to fix it in software or in hardware. To further test this hypothesis the motion command

```
$ rostopic pub -r 10 /commands/motor/speed std_msgs/Float64 "data: 1200.0"
```

was given to the SPURV. This command would publish a velocity command, forcing the motor of the SPURV to produce 1200 rpm. The velocity command was published ten times per second, as given by the "-r 10" term. If the command resulted in the SPURV was driving forwards, the error would lie elsewhere. However, the SPURV ended up driving backwards, strengthening the belief that the error was in the motor controller.

As previously mentioned, the most likely source of the error was the motor controller or the connection to the motor. As the necessary tools to access the motor controller was not available, the connections to the motor was switched. As the motor was a 3-pole motor, by switching any two poles, the motor would rotate in the opposite direction.

To validate that the change had an effect on the error, the command

```
$ rostopic pub -r 10 /cmd_vel geometry_msgs/Twist
    "linear:
      x: 0.4
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0"
```

to force the SPURV to drive with a velocity of 0.4 meters per second. The reason for using the `/cmd_vel` topic instead of `/commands/motor/speed` topic was due to the odometry. The odometry would not be started unless something was published on the `/cmd_vel` topic. The odometry readings were accessed by listening to the `/odom` topic with the command

```
$ rostopic echo /odom
```

After changing the poles of the motor, the SPURV was driving forwards when given the command to drive with a positive velocity. Additionally, the odometry also claimed the SPURV was driving in positive x-direction. Further testing was done using the wireless Xbox controller, accessed with the command

```
$ sudo xboxdrv
```

in a new terminal, and the control algorithm, accessed with the command

```
$ roslaunch spurv_examples joystick_example.launch
```

in a new terminal. The test was formed to test the odometry. First, by driving along the x-axis, which was defined from where the SPURV was started, the odometry for the x-direction was tested. The readings was increasing when the SPURV was driving in positive x-direction and decreasing when driving in negative x-direction. As the readings from the x-direction of the odometry was sensible, the SPURV was rotated 90° clockwise to investigate the y-axis. By driving the SPURV forwards, the odometry should give out negative numbers, and reversing should result in positive increase. This was also the case. As the odometry now gave sensible readings, the hypothesis that the error was with the motor controller/connections was strengthened.

3.11 Test 3 - Verify Odometry, Driving Speed

3.11.1 Test Plan for Test 3

The third test should test the changes done to the odometry of the SPURV. This would be done by using the same testing area as the first and second test, detailed in chapter 3.8.2, Figure 3.5 and Figure 3.6. The SPURV should take one lap in the test area and log the data using **rosvbag**. A wireless Xbox controller was going to be used. The remote computer sending the control information to the SPURV should also be recording the necessary topics. The computer was going to be placed in a backpack during the test drive. The logging of the data should be done using only the necessary topics discovered from the second test. Additionally, the influence of the driving speed was to be investigated.

GMapping was going to be used to map the area using the logged data. During the mapping process, **rviz** would verify whether the odometry was corresponding with the ground truth or not. Additionally, the quality of the map produced by **GMapping** in this test should be compared with the map produced in the second test. By comparing the maps, any improvements of quality could be a result of better odometry.

3.11.2 Execution of Test 3

In the third test, the route of the first test was reused. This route is described in more detail in chapter 3.8.2 and Figure 3.5 and Figure 3.6. Similarly to the second test, in the third test, the remote computer was placed in a backpack before driving the SPURV with the wireless Xbox controller.

The connection to the SPURV was achieved by connecting to the network "spurv", which would start up as the SPURV robot was turned on. Once connected to the "spurv" network, the function "spurv_master_in_wild" was used in each terminal who needed to communicate with the SPURV to set the correct IP-addresses. To check if the connection to the SPURV was established, the command

```
$ rostopic list
```

was ran. This would show all topics available, and if the topics coming from the SPURV showed up, a successful connection had been established. To be able to use the Xbox controller, the program **xboxdrv** was used. This had to be started in a separate terminal to get access to the signals coming from the wireless controller. This was done by running the command

```
$ sudo xboxdrv
```

in a new terminal. As the connection to the SPURV and the controller was set up, the program "joystick_example.py" was used to control the SPURV via the wireless Xbox controller. This program came with the SPURV, and had to be copied from the SPURV to the remote computer. The program was started using the command


```
$ roslaunch spurv_examples joystick_example.launch
```

in a new terminal. Due to the success of this method of connecting to and controlling the SPURV, as described above, this approach was decided to be used in subsequent tests when driving the SPURV. Additionally, it was decided that the approach had been detailed well, and that it did not need to be re-explained in every subsequent test.

As with the second test, the folder "test_3" had to be created and the data needed to be recorded in this folder. This was done by running the commands

```
$ mkdir -p bagfiles/test_3
```

```
$ cd bagfiles/test_3
```

```
$ rosbag record -O first_run /cloud /odom /tf /tf_static
```

in a terminal. These commands would create the folder "test_3" in "bagfiles", change the directory to "*sim*/bagfiles/test_3" and then record the topics /cloud, /odom and /tf and /tf_static to this directory. The "-O first_run" sets the name of the bagfile to "first_run". To investigate how the speed of the SPURV would influence the mapping process, a slow run of the testing route was performed. The data from this run was stored in "test_3" by using the command

```
$ rosbag record -O second_run_slow /cloud /odom /tf /tf_static
```

in the terminal already with the directory `~/bagfiles/test_3`.

For each run, once the scripts to control the SPURV with the wireless controller and the **rosbag** recording was started, the lid of the computer was shut close. The computer was quickly placed in the backpack and the SPURV was driven through the testing route. As the front facing camera was not used during the test, the resolution was set low to a resolution of 40×60. This was to minimize use of unnecessary bandwidth of the wireless network produced by the SPURV on transferring data coming from the camera feed.

When the necessary data from the SPURV was collected, the rest of the testing was done on the remote computer. To be able to run ROS once the SPURV was turned off, the function "spurv_local_power" had to be run. This function set the ROS IPs back to the local IPs of the computer. Additionally, the ROS master had to be activated using the command

```
$ roscore
```

in a separate terminal.

```

<?xml version="1.0"?>
<launch>

  <!-- run pointcloud_to_laserscan node -->
  <node pkg="pointcloud_to_laserscan" type="pointcloud_to_laserscan_node" name="pointcloud_to_laserscan">

    <remap from="cloud_in" to="/cloud"/> #$(arg camera)/depth_registered/points_processed
    <remap from="scan" to="/scan"/> #$(arg camera)/scan
    <roscparam>
      target_frame: laser # Leave disabled to output scan in pointcloud frame, original "camera_link"
      transform_tolerance: 0.01
      min_height: 0.0
      max_height: 2.0

      angle_min: -2.35619 # -M_PI/2, original -1.5708
      angle_max: 2.35619 # M_PI/2, original 1.5708
      angle_increment: 0.004363 # M_PI/360.0
      scan_time: 0.02
      range_min: 0.2
      range_max: 60.0
      use_inf: true

      # Concurrency level, affects number of pointclouds queued for processing and number of threads used
      # 0 : Detect number of cores
      # 1 : Single threaded
      # 2->inf : Parallelism level
      concurrency_level: 1
    </roscparam>

  </node>
</launch>

```

Figure 3.11: Launch file used in test 3 for `pointcloud_to_laserscan` with highlighted alterations

The launch file for this algorithm was located in the directory `/opt/ros/kinetic/share/pointcloud_to_laserscan/launch` and the launch file "sample_node.launch" was altered to the one shown in Figure 3.11. The advantage of using a launch file was that the parameters of the script were stored in the file. When starting up with the launch file, these parameters were loaded as well. The differences between the launch file used in the second test and the one used in the third test are highlighted in Figure 3.11. The field of view angle was changed from $\pm 90^\circ$ to $\pm 135^\circ$, given in radians as ± 2.35619 . This was to get the full benefit of the entire scanning range of the Lidar.

The `pointcloud_to_laserscan` package had to be started before **GMapping** could be used. This was due to the topic `/cloud` had to be transformed into the `/scan` topic beforehand. `pointcloud_to_laserscan` was started using the command

```
$ roslaunch pointcloud_to_laserscan sample_node.launch
```

in a new terminal.

```

<launch>
  <arg name="scan_topic" default="/scan" />
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="odom_frame" value="odom"/>
    <param name="base_frame" value="base_link"/>
    <param name="map_frame" value="map"/>

    <!-- Process 1 out of every this many scans (set it to a higher number to skip more scans) -->
    <param name="throttle_scans" value="1"/>

    <param name="map_update_interval" value="1.0"/> <!-- default: 5.0 -->

    <!-- The maximum usable range of the laser. A beam is cropped to this value. -->
    <param name="maxUrange" value="30.0"/>

    <!-- The maximum range of the sensor. If regions with no obstacles within the range of the sensor should appear as free space in the map, set maxUrange to this value. -->
    <param name="maxRange" value="60.0"/>

    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="minimumScore" value="0.0"/>
    <!-- Number of beams to skip in each scan. -->
    <param name="lskip" value="0"/>

    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>

    <!-- Process a scan each time the robot translates this far -->
    <param name="linearUpdate" value="0.1"/>

    <!-- Process a scan each time the robot rotates this far -->
    <param name="angularUpdate" value="0.05"/>

    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>

    <!-- Number of particles in the filter. default 30 -->
    <param name="particles" value="10"/>

  <!-- Initial map size -->
  <param name="xmin" value="-50.0"/>
  <param name="ymin" value="-50.0"/>
  <param name="xmax" value="50.0"/>
  <param name="ymax" value="50.0"/>

  <!-- Processing parameters (resolution of the map) -->
  <param name="delta" value="0.02"/>

  <param name="lssamplerange" value="0.01"/>
  <param name="lssamplestep" value="0.01"/>
  <param name="lasamplerange" value="0.005"/>
  <param name="lasamplestep" value="0.005"/>
</node>
</launch>

```

Figure 3.12: Launch file used in test 3 for GMapping in ROS

The launch file used to test **GMapping** is shown in Figure 3.12. The initial map size was set to 100×100 meters with a resolution of 2 cm per grid cell and 10 particles in the particle filter. To start **GMapping**, the command

```
$ rosparam set use_sim_time true
```

was executed followed by the command

```
$ roslaunch gmapping slam_gmapping_experimental.launch
```

in a new terminal. To play the **rosvbag**, the command

```
$ rosbag play <name_of_bag>
```

Was used. Once the **rosbag** was completed, the commands

```
$ mkdir -p map/test_3
```

```
$ cd map/test_3
```

and then the command

```
$ rosrun map_server map_saver -f first_run
```

was run in a new terminal to save the generated map with the name "first_run" in the directory `~/map/test_3`. After this was done, the terminal running **GMapping** had to be shut down using `ctrl+c` and restarted using the command

```
$ roslaunch gmapping slam_gmapping_experimental.launch
```

or simply scrolling through previously used commands using the up and down arrow keys. This was needed if another **rosbag** was to be tested. The map from the next **rosbag** was saved using the command

```
$ rosrun map_server map_saver -f second_run_slow
```

in the same terminal.

3.12 Test 4 - GMapping Parameter Test

3.12.1 Test Plan for Test 4

Due to the resulting maps from the third test, it was decided to perform a new test on the same data set as was used in the third test. The only difference between the tests was changing the parameters of **GMapping**. The overall goal of the fourth test was to investigate the effects of the parameters of **GMapping**. The parameters which were going to be investigated were the map resolution and number of particles in the particle filter. Previously these values were a map resolution of 5 cm/grid cell and 10 particles in the particle filter. The test was designed to investigate the impact of these two parameters. If the resulting maps had a better overall quality, the test would be considered a success. The hypothesis to be tested was that more particles in the particle filter should result in better pose estimates, as explained in the Theory chapters about particle filters.

3.12.2 Execution of Test 4

First, the ROS master had to be started. This was done by using the command

```
$ roscore
```

in a new terminal.

The `pointcloud_to_laserscan` package had to be started before **GMapping** could be used. This was due to the topic `/cloud` had to be transformed into the `/scan` topic beforehand. `pointcloud_to_laserscan` was started using the command

```
$ roslaunch pointcloud_to_laserscan sample_node.launch
```

in a new terminal.

```

<!-- Number of particles in the filter. default 30 -->
<param name="particles" value="30"/>

<!-- Initial map size -->
<param name="xmin" value="-50.0"/>
<param name="ymin" value="-50.0"/>
<param name="xmax" value="50.0"/>
<param name="ymax" value="50.0"/>

<!-- Processing parameters (resolution of the map) -->
<param name="delta" value="0.05"/>

<param name="lssamplerange" value="0.01"/>
<param name="lssamplestep" value="0.01"/>
<param name="lasamplerange" value="0.005"/>
<param name="lasamplestep" value="0.005"/>

</node>
</launch>

```

Figure 3.13: Launch file used in test 3 for GMapping with highlighted alterations

The launch file used to test **GMapping** is shown in Figure 3.13. The initial map size was set to 100×100 meters with a resolution of 5 cm per grid cell and 30 particles in the particle filter. To start **GMapping**, the command

```
$ roslaunch gmapping slam_gmapping_experimental.launch
```

was used in a new terminal. To play the **rosvbag**, the command

```
$ rosvbag play <name_of_bag>
```

Was used. Once the **rosvbag** was completed, the commands

```
$ mkdir -p map/test_3
```

```
$ cd map/test_3
```

and then the command

```
$ roslaunch map_server map_saver -f params_first
```

was run in a new terminal to save the generated map with the name "first_run" in the directory `~/map/test_3`. After this was done, the terminal running **GMapping** had to be shut down using `ctrl+c` and restarted using the command

```
$ roslaunch gmapping slam_gmapping_experimental.launch
```

or simply scrolling through previously used commands using the up and down arrow keys. This was needed if another **rosvbag** was to be tested. The map from the next **rosvbag** was saved using the command

```
$ roslaunch map_server map_saver -f params_second_slow
```

in the same terminal.



Figure 3.14: Debris found in the Long Hallway during Test 4

It was determined to collect additional data with the SPURV. The same test route was used to collect the data, however there were some debris in the way in the Long Hallway. This is shown in Figure 3.14. The same approach for controlling the SPURV and gathering and storing the data set, which previous tests have detailed, was utilized. The data from the SPURV was saved in the directory `~/bagfiles/test_4` and the maps were stored in the directory `~/map/test_4`.

Table 3.1: Testing parameters investigated in Test 4

Resolution of Map	Number of Particles
5 cm/grid cell	30
10 cm/grid cell	30
5 cm/grid cell	60
5 cm/grid cell	100

During the test run, the SPURV was driven at a slow walking speed. In every sharp corner, the SPURV was slowed down further. This was done to get good sensor data in the transitions between the different hallways. The map resolution and number of particles parameters were to be further tested to see their influence on the produced maps. Table 3.1 shows how the parameters was changed during testing. This was done by changing the parameters in the launch file before starting **GMapping**. Each parameter configuration was tested twice. This was to try to eliminate random results and thereby reinforcing the results.

3.13 Test 5 - Odometry Investigation

3.13.1 Test Plan for Test 5

During the fourth test, a trend concerning a spiralling effect on the maps produced by **GMapping** was discovered. It was decided to investigate in what way the odometry of the SPURV was playing a part of this spiralling effect. To test this, the test route described in chapter 3.8.2 was going to be used. In contrast to previous test, the test route was going to be driven in the opposite direction. If the spiral effect indeed was caused by the odometry, the spiral would change direction. The hypothesis tested was that the sharp turns of the hallways would be contributing to a bad odometry estimate. If this was the case, by turning the SPURV in the opposite direction in each corner, the spiral should also change direction, reflecting the change in turning direction. After collecting the data from the test route, **GMapping** was going to be used to validate if the spiral had changed direction or not.

3.13.2 Execution of Test 5



Figure 3.15: Debris found in the Long Hallway during Test 5

In the fifth test, the testing route described in chapter 3.8.2 was once again used. In contrast to previous test, the route was driven in the opposite direction, meaning the SPURV started in the starting point, then driving to Open Area 2, Open Area 1, Narrow Hallway, Long Hallway before ending up at the starting point again. In other words, as seen from the starting point, the SPURV would be driving in a clockwise direction. The only difference from this test and prior test was the debris shown in Figure 3.15. The collection of data was done in a similar fashion as described in the previous tests. The parameters used by **GMapping** for map resolution and number of particles in the particle filter were 10 cm/grid cell and 30 particles.

Due to the results of the first run of Test 5, the test was decided to be split into three sub tests. In part 1 of the test, the SPURV was to be driven slowly through the testing route in a clockwise direction. In part 2 of the test, the SPURV was to be driven fast through the testing route in a clockwise direction. This was done to ensure that **GMapping** would break, revealing the direction of the spiral. Lastly, in part 3 of the test, the SPURV was to be driven slowly through the original testing direction, meaning counter-clockwise. This was done as a control.

3.14 Test 6 - Hector SLAM

3.14.1 Test Plan for Test 6

The results of the third test revealed that the odometry problems from Test 2 was fixed. However, the odometry seemed to struggle with sharp corners. For that reason, it was decided to test **hector_slam** with the data collected in all the previous tests. The results of **hector_slam** should be compared with the maps produced by **GMapping** from the previous tests. The **hector_slam** approach was interesting to investigate as it did not utilize the odometry data. As the SPURV struggled to give accurate odometry readings in the sharp corners used in the testing area, it was suspected that **hector_slam** might yield a better results. The test consisted of altering the launch file of **hector_slam** to make it work with the data recorded by the SPURV in the previous tests. Next, the maps produced by **hector_slam** and **GMapping** were going to be compared. This was done to find the best mapping approach based on the testing area and collected data.

3.14.2 Execution of Test 6

```

<?xml version="1.0"?>
<launch>
  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_footprint"/>
  <arg name="odom_frame" default="nav"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>
  <node pkg="tf2_ros" type="static_transform_publisher" name="base_link_to_laser" args="0.3 0 0.25 0 0 0 base_link laser" />
  <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen">
    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="base_link" /> #$(arg base_frame)
    <param name="odom_frame" value="odom" /> #$(arg odom_frame)

    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/>

    <!-- Map size / start point -->
    <param name="map_resolution" value="0.100"/>
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />

    <!-- Map update parameters -->
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.4"/>
    <param name="map_update_angle_thresh" value="0.06" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />

    <!-- Advertising config -->
    <param name="advertise_map_service" value="true"/>

    <param name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>

    <!-- Debug parameters -->
    <!--
    <param name="output_timing" value="false"/>
    <param name="pub_drawings" value="true"/>
    <param name="pub_debug_output" value="true"/>
    -->
    <param name="tf_map_scanmatch_transform_frame_name" value="$(arg tf_map_scanmatch_transform_frame_name)" />
  </node>
  <!--<node pkg="tf" type="static_transform_publisher" name="map_nav_broadcaster" args="0 0 0 0 0 0 map nav 100"/>-->
</launch>

```

Figure 3.16: Launch file used in test 6 for hector_slam algorithm in ROS

It was decided to use the **rosbags** from Test 2, Test 3, Test 4 and Test 5 to check the performance of **hector_slam**. The original launch file "mapping_default.launch" for **hector_mapping**, found in the directory `/opt/ros/kinetic/share/hector_mapping/launch`, was altered with the entries highlighted in Figure 3.16. The two changes consisted of adding a static transformation detailing the placement of the Lidar sensor compared with the reference coordinate system and changing the resolution of the map from 5 cm/grid cell to 10 cm/grid cell.

The tests were performed by first starting the rosmaster using the command

```
$ roscore
```

in a new terminal. Then the `pointcloud_to_laserscan` node had to be started. This was done by using the command

```
$ roslaunch pointcloud_to_laserscan sample_node.launch
```

in a new terminal. Next, **hector_slam** was started using the command

```
$ roslaunch hector_slam_launch tutorial.launch
```

in a new terminal. The latest command would start up **hector_mapping**, **rviz** and **hector_geotiff**. The **hector_geotiff** was used to save, store and recall maps produced by **hector_slam**. Lastly, the command

```
$ rosbag play <name_of_bag> --clock
```

was used to play the recorded **rosbags** from the previous tests. For each new **rosbag**, the **hector_slam** was stopped with **ctrl+c** and restarted before playing the next bag. **hector_slam** was restarted using the same command

```
$ roslaunch hector_slam_launch tutorial.launch
```

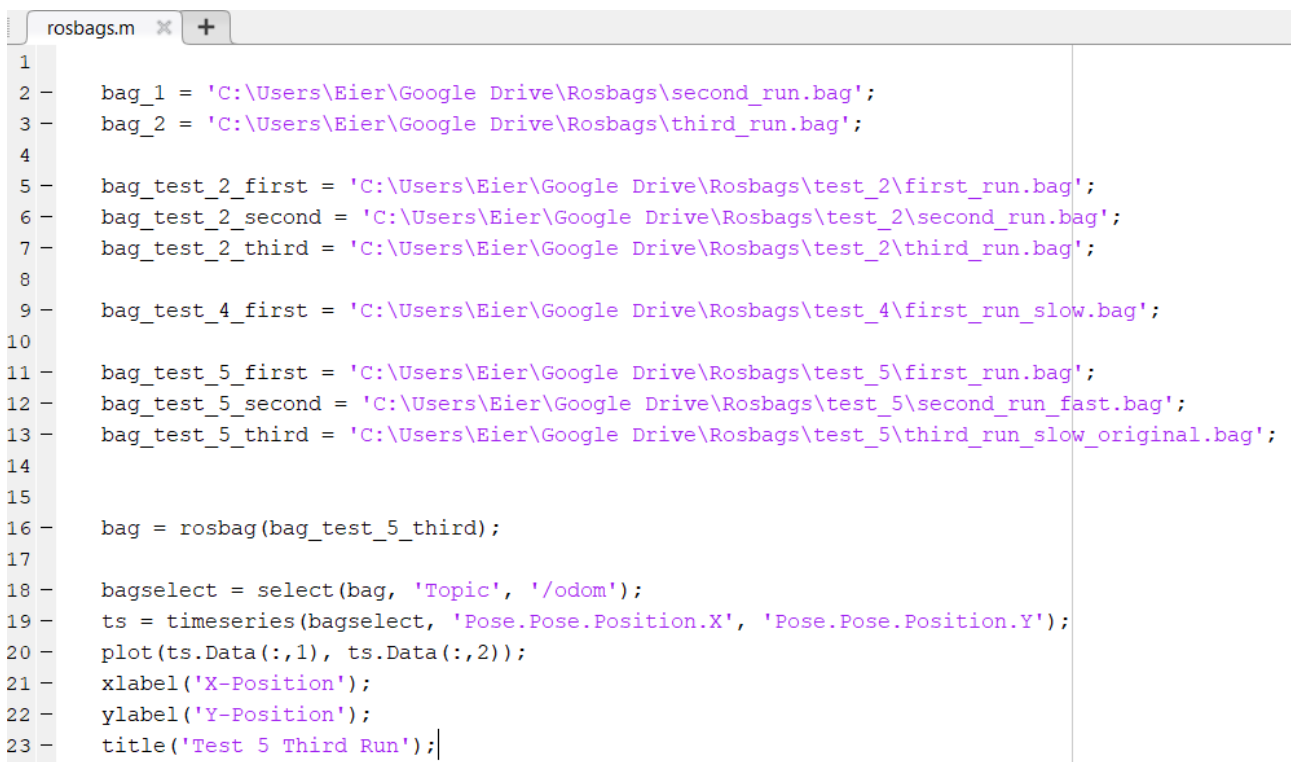
in the terminal it was originally used in.

3.15 Test 7 - MATLAB Plot of Odometry

3.15.1 Test Plan for Test 7

Test 7 was designed to verify whether the odometry of the SPURV was accurate or not. It was decided to plot the odometry topic coming from the different **rosbags** in MATLAB. MATLAB would be used to plot the output of the `/odom` topic and the results of these plots would show the overall odometry of the SPURV. If these plots did not corresponded with the ground truth, it would be concluded that the odometry of the SPURV had to be improved.

3.15.2 Execution of Test 7



```
1
2 - bag_1 = 'C:\Users\Eier\Google Drive\Rosbags\second_run.bag';
3 - bag_2 = 'C:\Users\Eier\Google Drive\Rosbags\third_run.bag';
4
5 - bag_test_2_first = 'C:\Users\Eier\Google Drive\Rosbags\test_2\first_run.bag';
6 - bag_test_2_second = 'C:\Users\Eier\Google Drive\Rosbags\test_2\second_run.bag';
7 - bag_test_2_third = 'C:\Users\Eier\Google Drive\Rosbags\test_2\third_run.bag';
8
9 - bag_test_4_first = 'C:\Users\Eier\Google Drive\Rosbags\test_4\first_run_slow.bag';
10
11 - bag_test_5_first = 'C:\Users\Eier\Google Drive\Rosbags\test_5\first_run.bag';
12 - bag_test_5_second = 'C:\Users\Eier\Google Drive\Rosbags\test_5\second_run_fast.bag';
13 - bag_test_5_third = 'C:\Users\Eier\Google Drive\Rosbags\test_5\third_run_slow_original.bag';
14
15
16 - bag = rosbag(bag_test_5_third);
17
18 - bagselect = select(bag, 'Topic', '/odom');
19 - ts = timeseries(bagselect, 'Pose.Pose.Position.X', 'Pose.Pose.Position.Y');
20 - plot(ts.Data(:,1), ts.Data(:,2));
21 - xlabel('X-Position');
22 - ylabel('Y-Position');
23 - title('Test 5 Third Run');
```

Figure 3.17: MATLAB code used to plot `/odom` topic from **rosbags**

The script shown in Figure 3.17 shows the script used to realize the plotting of the **rosbags** in MATLAB. The add on "Robotics System Toolbox" had to be downloaded and installed to MATLAB before the code could function. The full path to the **rosbag** had to be provided to the function **rosbag()** for MATLAB to get access to the **rosbag**. Once MATLAB had access to the **rosbag**, the **timeseries()** function was used to obtain the x- and y-position of the SPURV. After this, the pose was plotted and the axis of the plot was labelled before giving the plot a suitable title.

3.16 Test 8 - Repeating, Featureless Environment

3.16.1 Test Plan for Test 8

The eighth and final test was designed to test the performance of `GMapping` and `hector_slam` in the area of the testing route with the least amount of features. In this test, it was decided to use the Long Hallway. This was the area with the least features in the testing route. The test was split into two separate sub-tests. In the first part of the test, the SPURV was supposed to drive down the entirety of the Long Hallway and return to the starting point. In the second part of the test, the SPURV was driven several laps in front of an area without any significant features before returning to the starting point. In both test, the result would be evaluated based on the overall quality of the maps and whether the starting point and end point of the maps would correspond or not.

3.16.2 Execution of Test 8



(a)



(b)



(c)



(d)



(e)

Figure 3.18: Testing area used in the first part of Test 8

In the first part of Test 8, the testing area shown in Figure 3.18 was used. The SPURV was driven down the hallway with the wireless Xbox controller and the remote computer was collecting the data produced by the SPURV. The SPURV was driven through the hallway, as shown in Figure 3.18 a) to d). This was the end of the hallway, and the SPURV was driven back to the original starting point. On the way back, the SPURV would pass the area shown in Figure 3.18 e) before turning right, ending up at the starting area. In both parts of the test, the SPURV was driven at a slow walking speed. Once the data was collected, **GMapping** and **hector_slam** was used to produce maps and MATLAB was used to check the odometry coming from the SPURV. These approaches has been detailed in previous tests.



Figure 3.19: Featureless area used in the second part of Test 8

In the second part of the test, portions the Long Hallway was reused. In contrast to the first part of the test, the SPURV was to take several laps in front of the area with the least features of the Long Hallway. This area is shown in Figure 3.19. The SPURV was driven to this area and then it took three laps in front of the featureless area before returning to the starting area. Once again, **GMapping** and **hector_slam** was used to produce maps and MATLAB was used to check the odometry coming from the SPURV. These approaches has been detailed in previous tests.

Chapter 4

Results

In this section, the results from the testing of algorithms and the SPURV are going to be presented and commented on.

4.1 Test 1 - Initial Testing



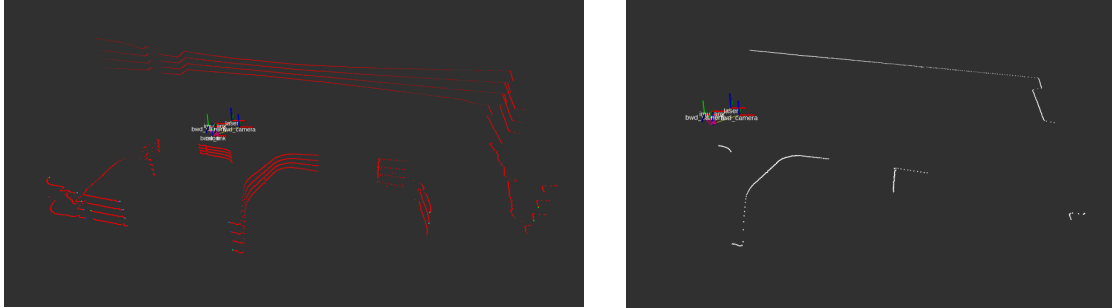
Figure 4.1: Map generated using GMapping from the second test

The SPURV was not able to perform the required route which was defined by Test 1. Around the doorway of the Long Hallway, shown in Figure 4.1, the data transfer speed of the SPURV became too slow to safely control the SPURV with camera feedback only. Despite this, two sets of data was recorded from the start point to the middle of the Long Hallway. These data sets revealed that there was an error with the laser scan data. There seemed to be a transformation that was missing.

Additionally, it was discovered that by changing the resolution of the front facing camera in the set up file did not retain the full image, only with poorer quality, as one would expect. Instead, the image was "cropped", resulting in only the top left part of the image remaining after decreasing the resolution. The remaining area was dependant on what the resolution was set to, naturally.

4.2 Test 2 - Controlling SPURV, Identify Topics

Walking behind the SPURV while it was recording data worked well. Additionally, the topics needed to recreate the data in `rviz` was discovered. The topics needed was `/cloud`, `/odom`, `/tf` and `/tf_static`. Once the `roscap` was set up to record these topics, the recording process as well as driving and manoeuvring of the SPURV was trivial.



(a) Starting area of the second test with `/cloud` topic (b) Starting area of the second test with `/scan` topic

Figure 4.2: Transition between `/cloud` topic to `/scan` topic

The `roscaps` collected provided the necessary topics to be reproduced in `rviz`, as shown in Figure 4.2 a). Further, the `pointcloud_to_laserscan` worked to generate the `/scan` topic, which can be seen in Figure 4.2 b). The screen-shots were taken in the same general time frame and area to make them as representative of the transformation as possible.

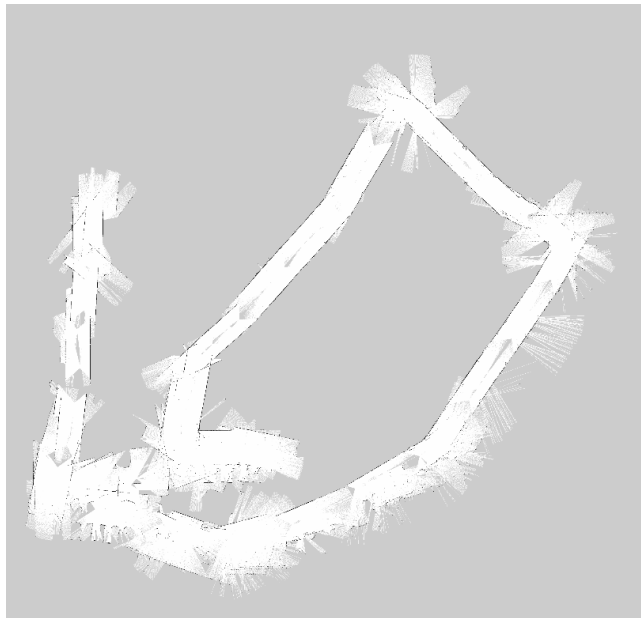
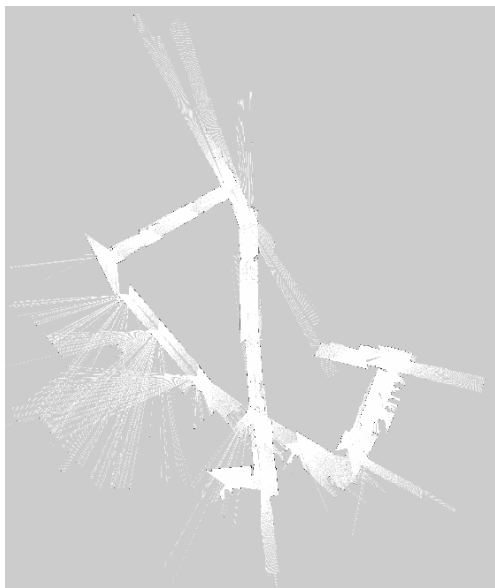


Figure 4.3: Map generated using GMapping from the second test

By using the `roscap`, `pointcloud_to_laserscan` and GMapping, a map of the test route was created, as shown in Figure 4.3.

4.3 Test 3 - Verify Odometry, Driving Speed



(a) Result of GMapping at brisk walking speed



(b) Result of GMapping at slow walking speed

Figure 4.4: Maps generated using GMapping from the third test

The results shown in Figure 4.4 are from the third test. Here, GMapping was used with 10 particles in the particle filter and a map resolution of 2 cm/grid cell. The figure shows the testing area at campus Grimstad as described in chapter 3.8.2. The map shown in Figure 4.4 b) was cut short due to technical difficulties during the test run. The end point was not reached and the map stops in the middle of Open Area 2, as described in chapter 3.8.2. This was not seen as a problem, as the function of the second part of the third test was to see how much the speed of the SPURV influenced the mapping quality of **GMapping**.

4.4 Test 4 - GMapping Parameter Test



(a) Result of GMapping at brisk walking speed



(b) Result of GMapping at slow walking speed

Figure 4.5: Maps generated using GMapping from the fourth test

The results shown in Figure 4.5 are from the third test. Here, GMapping was used with 30 particles in the particle filter and a map resolution of 5 cm/grid cell. The figure shows the testing area at campus Grimstad as described in chapter 3.8.2. The map shown in Figure 4.5 b) was cut short due to technical difficulties during the test run. The end point was not reached and the map stops in the middle of Open Area 2, as described in chapter 3.8.2.

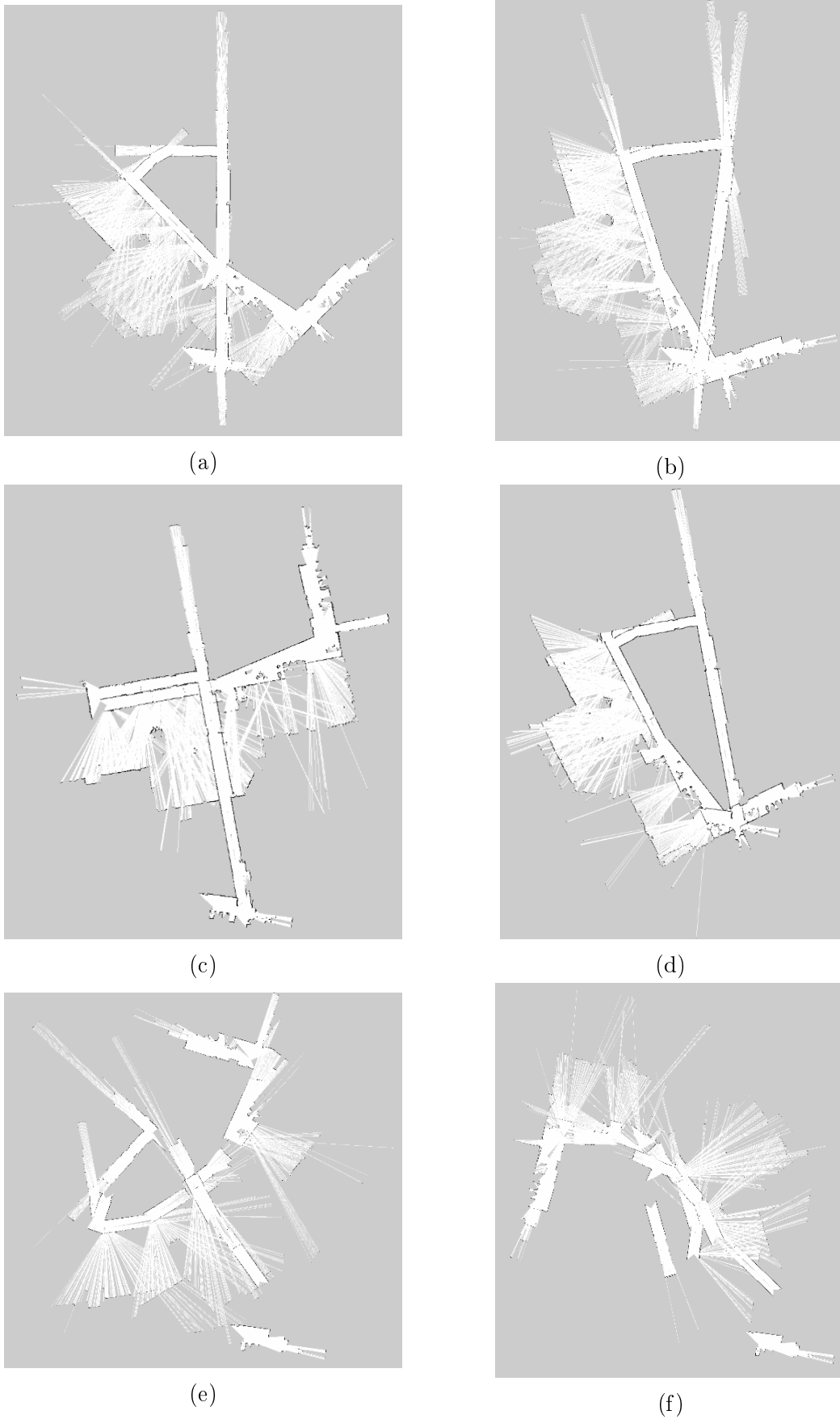


Figure 4.6: Maps generated using GMapping from the parameter test

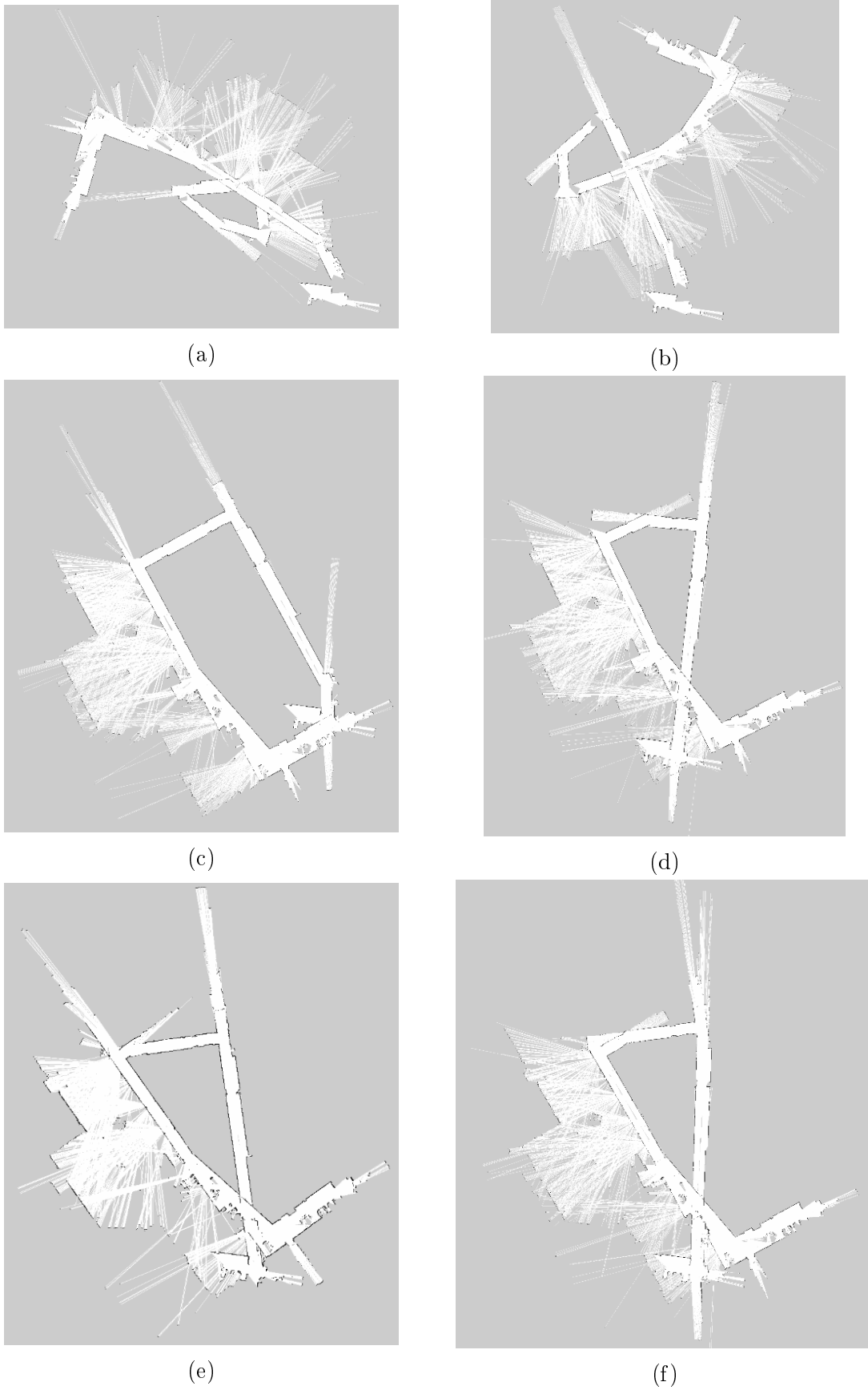


Figure 4.7: Maps generated using GMapping from the parameter test continuation

Table 4.1: Testing parameters used in Test 4

Resolution of Map	Number of Particles	Corresponding Figure
5 cm/grid cell	30	Figure 4.6 a) and b)
10 cm/grid cell	30	Figure 4.6 c) and d)
5 cm/grid cell	100	Figure 4.6 e) and f)
5 cm/grid cell	60	Figure 4.7 a) and b)
5 cm/grid cell	30	Figure 4.7 c) and d)
10 cm/grid cell	30	Figure 4.7 e)
5 cm/grid cell	30	Figure 4.7 f)

After collecting a new data set with the SPURV, the testing parameters shown in Table 4.1 was used with **GMapping**. The maps shown in Figure 4.7 c) and d) was created as a control after the poor quality of Figure 4.6 e) and f) and Figure 4.7 a) and b). Figure 4.7 e) and f) was created to check the two best solutions one more time to check their performance.

4.5 Test 5 - Odometry Investigation

4.5.1 Results of Test 5 Part 1

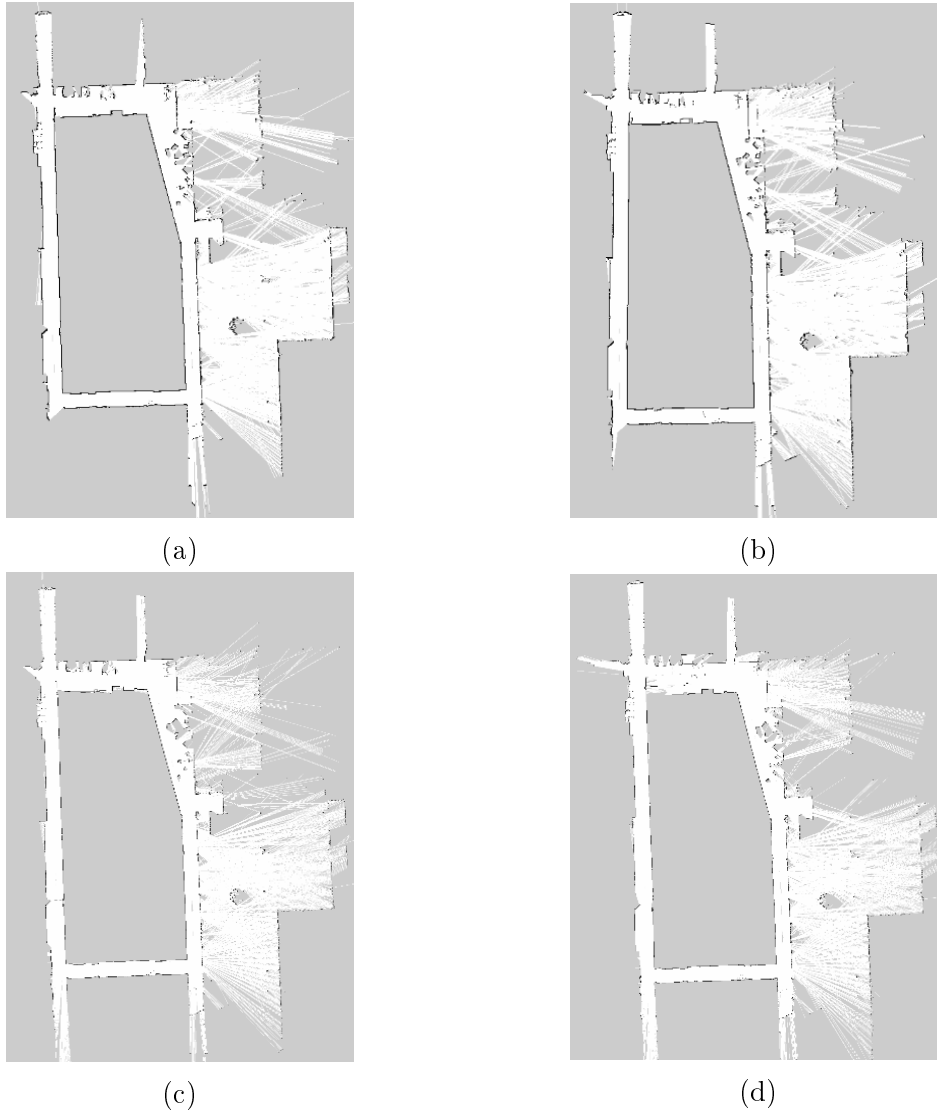


Figure 4.8: Maps generated using GMapping resulting from Test 5 part 1

Table 4.2: Parameters used in Test 5 part 1

Resolution of Map	Number of Particles	Corresponding Figure
10 cm/grid cell	30	Figure 4.8 a) and b)
5 cm/grid cell	30	Figure 4.8 c) and d)

The results of the first part of Test 5 gave good coherent maps. These maps are shown in Figure 4.8. The maps are continuous and the starting point and end point matches up well, as can be seen as the top part of the maps are connected.

4.5.2 Results of Test 5 Part 2

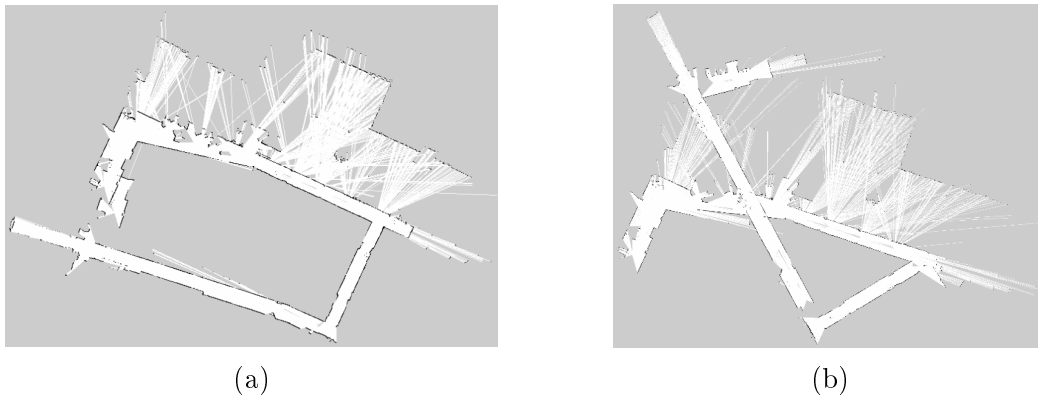


Figure 4.9: Maps generated using GMapping resulting from Test 5 part 2

Table 4.3: Parameters used in Test 5 part 2

Resolution of Map	Number of Particles	Corresponding Figure
10 cm/grid cell	30	Figure 4.9 a)
5 cm/grid cell	30	Figure 4.9 b)

In the second part of Test 5, the maps created by **GMapping** would start to break. The maps shown in Figure 4.9 was selected to represent the best outcome, Figure 4.9 a), and worst outcome, Figure 4.9 b), of the second part of Test 5. It was performed ten tests, which are all represented in Appendix A. In five of the ten tests, the maps would break in a similar fashion as shown in Figure 4.9 b). In the remaining five cases, the maps would look more closely to the map shown in Figure 4.9 a).

4.5.3 Results of Test 5 Part 3

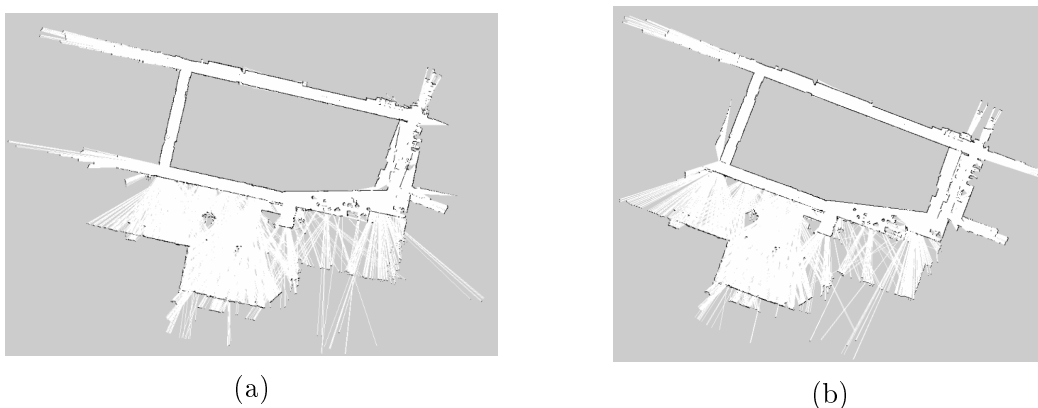


Figure 4.10: Maps generated using GMapping resulting from Test 5 part 3

Table 4.4: Parameters used in Test 5 part 3

Resolution of Map	Number of Particles	Corresponding Figure
10 cm/grid cell	30	Figure 4.10 a) and b)

In the third and final part of Test 5, the test route was driven in the original counter-clockwise direction. The maps shown in Figure 4.10 were the results of this test. Even though this part of the test was performed while driven the SPURV very slowly, the starting point and end point of the route did not match up particularly well. This can be seen in Figure 4.10 and looking to the rightmost part of the maps. Compared to the maps produced in the first part of Test 5, shown in Figure 4.8, the maps produced by part three of Test 5 were of a poorer quality.

4.6 Test 6 - Hector SLAM

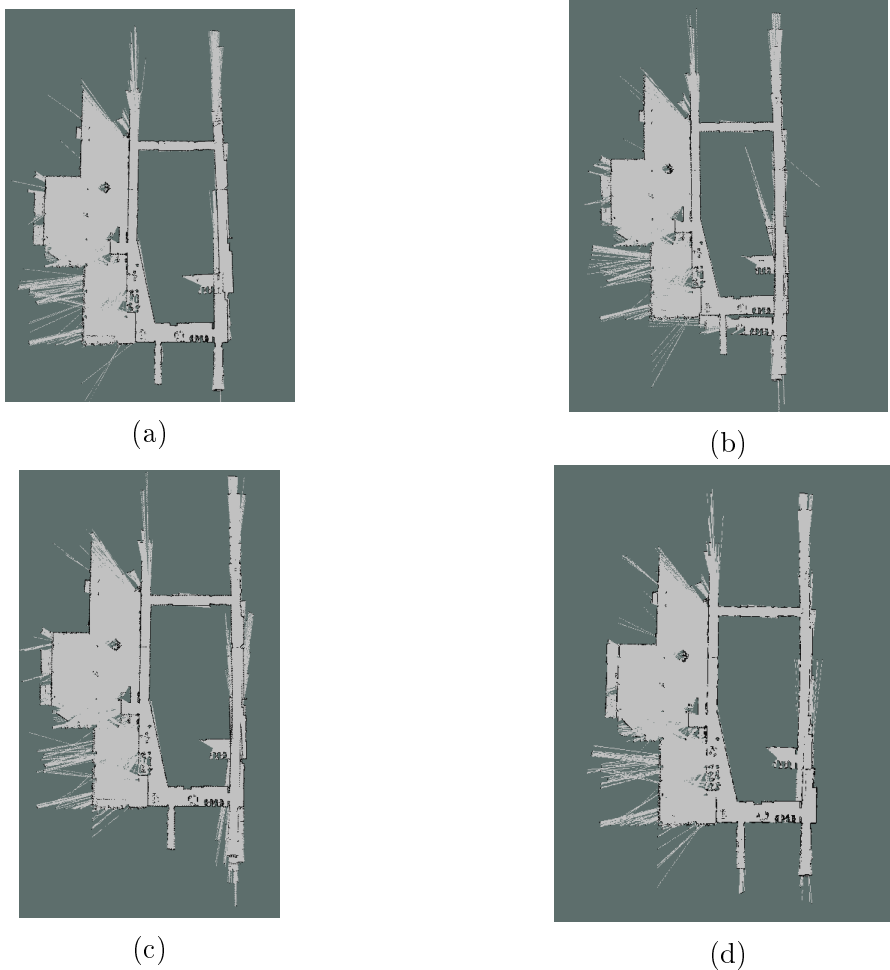
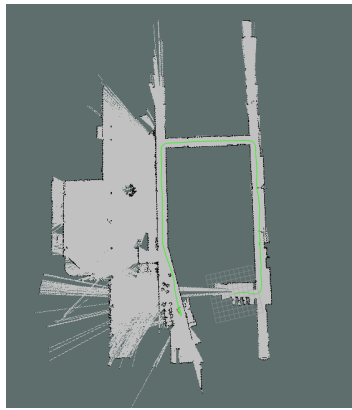


Figure 4.11: Maps generated using Hector SLAM



(a)



(b)



(c)



(d)



(e)

Figure 4.12: Maps generated using Hector SLAM continuation

Table 4.5: Hector SLAM results

Associated Figure	Associated Test	Associated rosbag
Figure 4.11 a)	Test 2	first_run
Figure 4.11 b)	Test 2	second_run
Figure 4.11 c)	Test 2	third_run
Figure 4.11 d)	Test 3	first_run
Figure 4.12 a)	Test 3	second_run_slow
Figure 4.12 b)	Test 4	first_run_slow
Figure 4.12 c)	Test 5 Part 1	first_run
Figure 4.12 d)	Test 5 Part 2	second_run_fast
Figure 4.12 e)	Test 5 Part 3	third_run_slow_original

The results shown in Figure 4.11 and Figure 4.12 are the results of using the `hector_slam` algorithm on the `rosbags` coming from Test 2, Test 3, Test 4 and Test 5. The associated test and `rosbag` to each figure are detailed in Table 4.5. All of the maps resulting from a test where the SPURV was driving counter-clockwise around the testing area struggles to connect the starting point and end point of the test route. This can easily be seen in Figure 4.11 b) and Figure 4.12 e). In the bottom of both figures, the start point and end point does not connect at all. By looking closely at the bottom of Figure 4.11 b), it can be seen that the same end point is repeating three times. In the bottom of Figure 4.12 e) the same area is repeated twice. In contrast, in Figure 4.12 b), Figure 4.12 c) and Figure 4.12 d), the maps were all coherent and well constructed.

4.7 Test 7 - MATLAB Plot of Odometry

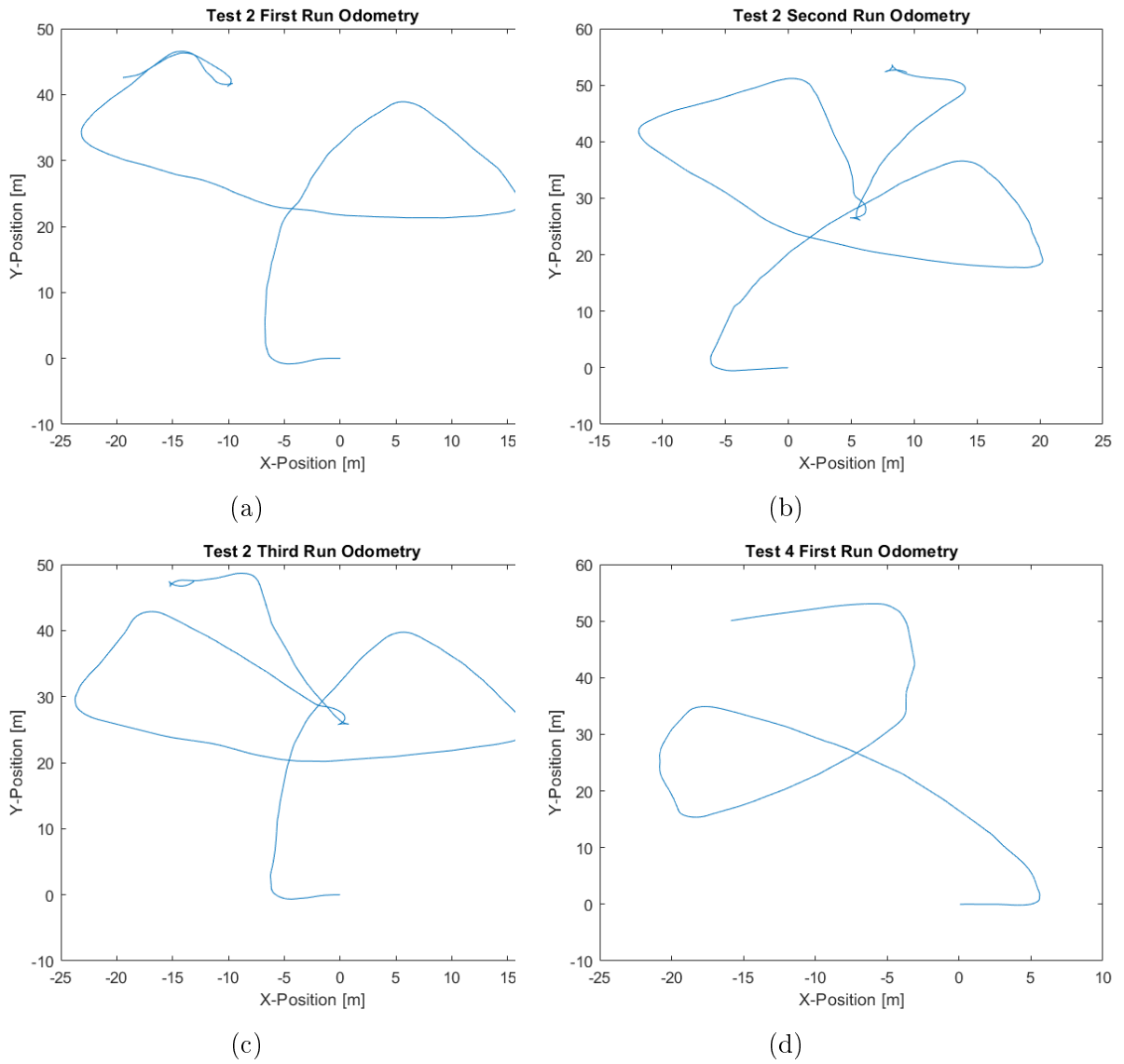


Figure 4.13: Plot of the odometry of the SPURV during testing from MATLAB

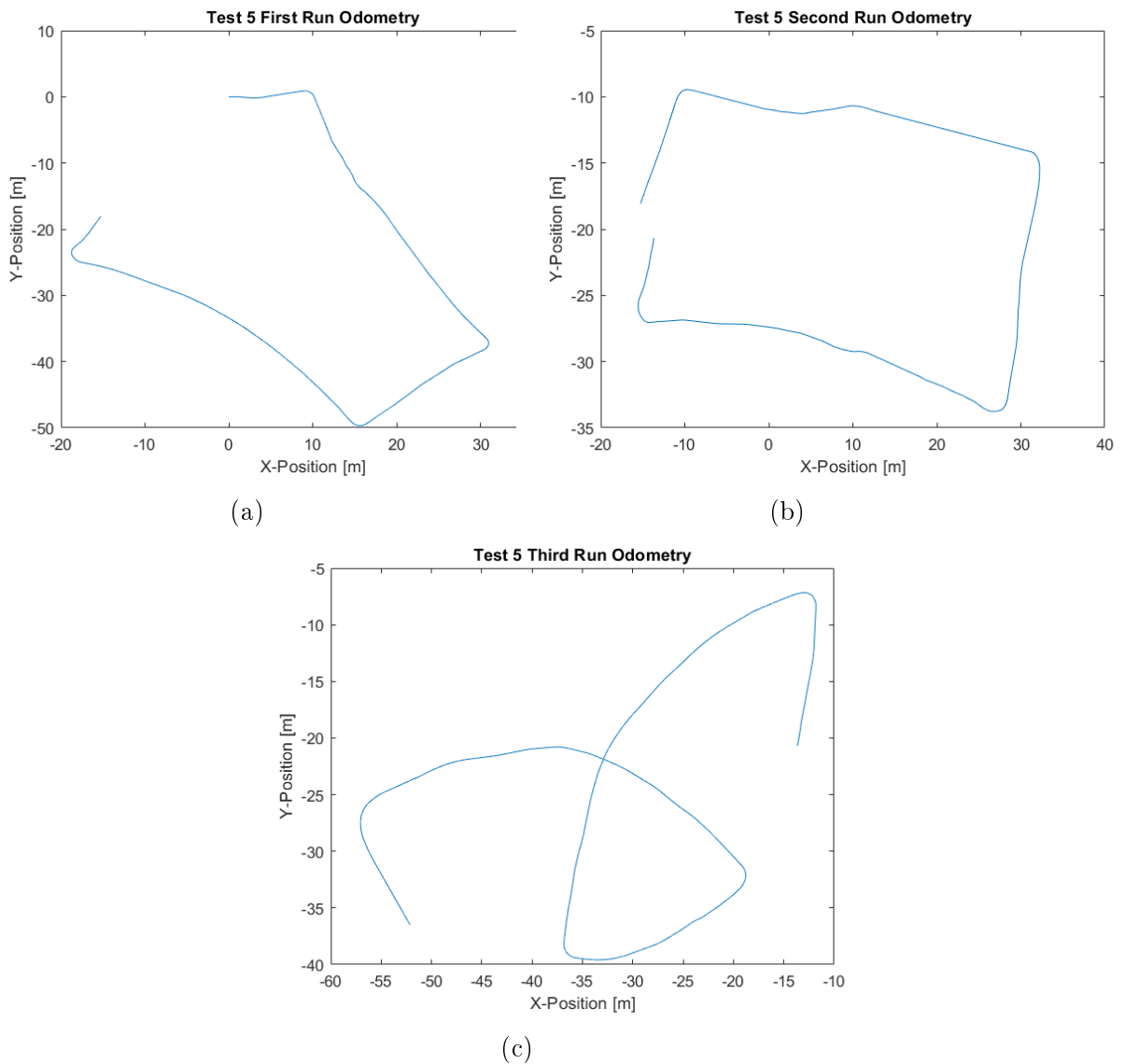


Figure 4.14: Plot of the odometry of the SPURV during testing from MATLAB continuation

Table 4.6: Corresponding data to the odometry plots

Associated Figure	Associated Test	Associated rosbag	Test Direction
Figure 4.13 a)	Test 2	first_run	CCW
Figure 4.13 b)	Test 2	second_run	CCW
Figure 4.13 c)	Test 2	third_run	CCW
Figure 4.13 d)	Test 4	first_run	CCW
Figure 4.14 a)	Test 5 Part 1	first_run	CW
Figure 4.14 b)	Test 5 Part 2	second_run_fast	CW
Figure 4.14 c)	Test 5 Part 3	third_run_slow_original	CCW

Figure 4.13 and Figure 4.14 shows the results of plotting the odometry of the SPURV in MATLAB. Table 4.6 gives further information to the figures. In the instances where the SPURV

was performing the test route in a counter-clockwise direction, the odometry seemed to not correspond with the ground truth. This is evidenced by looking at Figure 4.13 a), Figure 4.13 b), Figure 4.13 c), Figure 4.13 d), Figure 4.14 b) and Figure 4.14 c). All of these figures has one thing in common, the testing route was all driven in a counter-clockwise direction. Figure 4.14 a) and Figure 4.14 b) both showcases a decent odometry. In both of these instances, the robot was driving in a clockwise direction.

Table 4.7: Error between starting point and end point from Test 7

Associated Figure	Start Point (x,y)	End Point (x,y)	Error [m] (x,y) (EP-SP)	Error [m] $\left(\sqrt{x^2 + y^2}\right)$
Figure 4.13 a)	(0,0)	(-20,45)	(-20,45)	49
Figure 4.13 b)	(0,0)	(10,50)	(10,50)	51
Figure 4.13 c)	(0,0)	(-14,48)	(-14,48)	50
Figure 4.13 d)	(0,0)	(-16,50)	(-16,50)	52
Figure 4.14 a)	(0,0)	(-16,-19)	(-16,-19)	25
Figure 4.14 b)	(-16,-19)	(-15,-21)	(1,-2)	2
Figure 4.14 c)	(-15,-21)	(-52,-37)	(-37,-16)	40

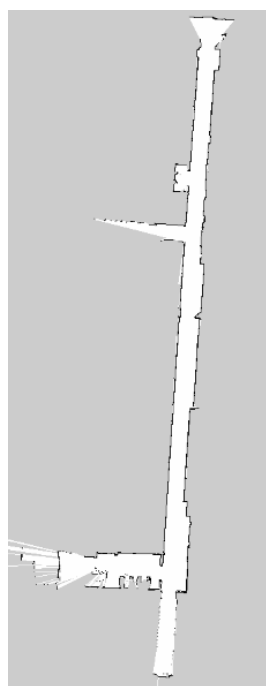
The error shown in Table 4.7 was defined as End Point – Start Point. The coordinates of the Start Point and End Point was gathered by visual approximation of the coordinate value. This was deemed valid as the error between the Start Point and End Point was far greater than the error from the visual approximation. In all of the tests, the SPURV was driven to the approximate same spot at the end of the test as where it started, give or take a few meters. Even though this approach was imprecise, it does not cause errors of 10 to 37 meters in x-direction and 45 to 50 meters in y-direction. The errors from the position estimate from the odometry of the SPURV can be seen in Table 4.7 in the "Error" column.

4.8 Test 8 - Repeating, Featureless Environment

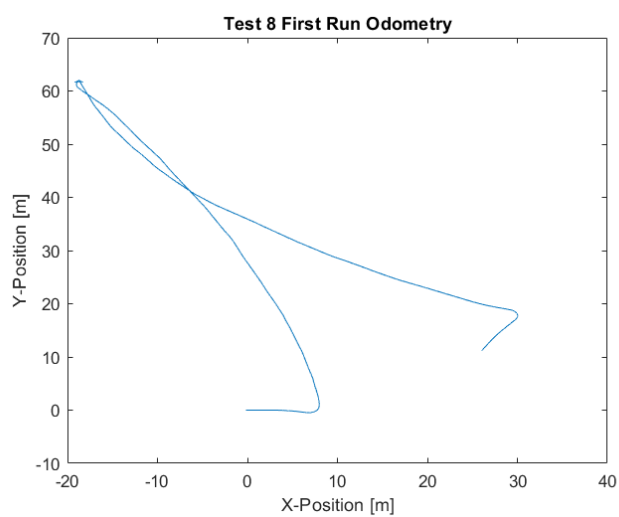
4.8.1 Results of Test 8 Part 1



(a)



(b)



(c)

Figure 4.15: Results of Test 8 Part 1 with Hector SLAM, GMapping and odometry

The results coming from Test 8 Part 1 is shown in Figure 4.15. Both maps produced by **GMapping** and **hector_slam** were of good quality. In addition, the starting point and end point of the maps lined up in both of the instances. This is shown in Figure 4.15 a) and Figure 4.15 b). Figure 4.15 c) shows the odometry coming from the SPURV during the test. In the figure, it can be seen that the odometry estimate of the position of the SPURV starts off good, then progressively turns worse as the SPURV drives further away from the starting area.

Table 4.8: Error between starting point and end point from Test 8 Part 1

Associated Figure	Start Point (x,y)	End Point (x,y)	Error [m] (x,y) (EP-SP)	Error [m] $\left(\sqrt{x^2 + y^2}\right)$
Figure 4.13 a)	(0,0)	(26,11)	(26,11)	28

Table 4.8 shows the error from the position estimate coming from the odometry of the SPURV. In this test, the error in the position estimate of the SPURV was quite severe. The error in position was 26 meters in x-direction and 11 meters in y-direction. Despite this, the map generated in **GMapping** was quite good. This map can be seen in Figure 4.15 b).

4.8.2 Results of Test 8 Part 2

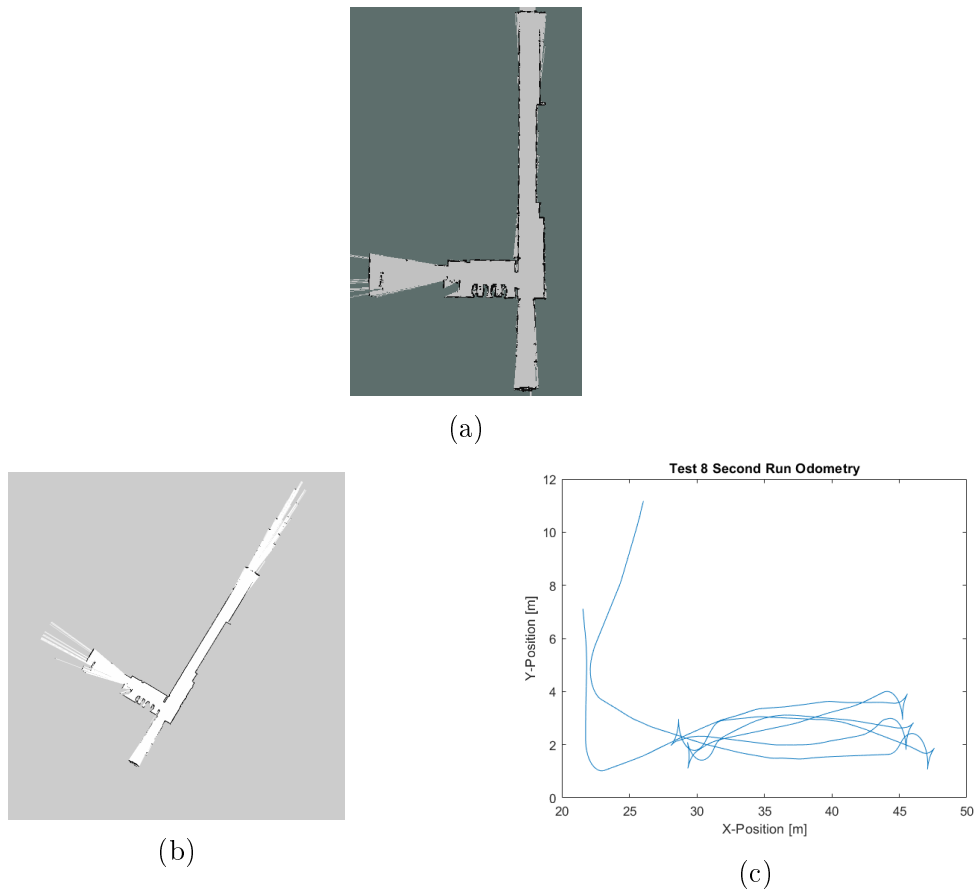


Figure 4.16: Results of Test 8 Part 2 with Hector SLAM, GMapping and odometry

The results from Test 8 Part 2 were also good. These can be seen in Figure 4.16. Both **GMapping** and **hector_slam** produced good quality maps where both the starting point and end point of the testing area lined up. This is shown in Figure 4.16 a) and Figure 4.16 b). Figure 4.16 c) shows the position estimate coming from the odometry of the SPURV. In this instance, the odometry estimate was not too bad, almost lining up with the ground truth.

Table 4.9: Error between starting point and end point from Test 8 Part 2

Associated Figure	Start Point (x,y)	End Point (x,y)	Error [m] (x,y) (EP-SP)	Error [m] $\left(\sqrt{x^2 + y^2}\right)$
Figure 4.13 a)	(26,11)	(22,7)	(-4,-4)	6

Table 4.9 shows the error from the position estimate coming from the odometry of the SPURV. In this test, the error in the position estimate was -4 meters in x-direction and -4 meters in y-direction. This error was more justified based on the imprecise placement of the starting point and end point during testing. The map generated in **GMapping** can be seen in Figure 4.16 b).

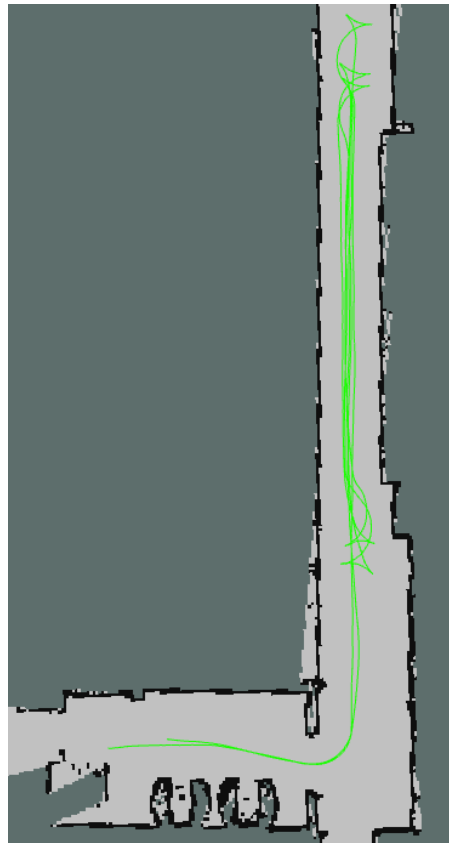


Figure 4.17: Position estimate from Hector SLAM during Test 8 Part 2

Figure 4.17 shows the position estimate coming from **hector_slam**. This position estimate was more or less spot on to the ground truth and is shown as the green line in Figure 4.17.

Chapter 5

Discussion

In this section, the results coming from the different test are going to be discussed. This means that the results from the individual tests as well as different tests might be compared to each other to form a statement regarding the test results. The statement should be a reflection of the goals described in the test plan for the test in question.

5.1 Test 1 - Initial Testing

Even though Test 1 did not yield the expected results, it made for an excellent learning opportunity. It was discovered that the SPURV was not able to transfer both the camera feed and the sensor data over vast distances. However, at short distances, this was not a problem. Additionally, another discovery was made. The **rosvbag** was set to record all data produced by the SPURV. This resulted in huge amounts of unwanted data. Future tests could be improved by recording the desired data only, instead of recording all produced data.

Another improvement which could be carried out in subsequent tests was to walk behind the SPURV robot when it was driving, with the remote computer in a backpack. This would increase the ease of controlling the SPURV in the narrow hallways. Additionally, this would make it easier to transfer the sensor data, as it was discovered that this worked well over short distances.

As it was discovered that changing the resolution did not result in the same image with grainier image quality, driving with camera feedback was discarded as a valid way of controlling the SPURV. This problem would also be solved by walking behind the SPURV when it was operating.

There was an error with the data coming from the laser scanner. The transformation seemed to be missing. This was also confirmed by looking at the description of the **sick_scan** [10] package. This package was the one used by the Lidar. The description stated the following:

"The LaserScan data should only be used for debugging purposes. They provide the raw data for each scan plane in a different coordinate frame. Due to the geometry of the scanning planes of the MRS1104 there is no coordinate transformation between

the scan planes that can be described by tf messages, therefore no tf messages are published. That is why the PointCloud2 data should be used." [10]

In the subsequent tests, the data from the point cloud of the Lidar was going to be used and converted to laser scan using `pointcloud_to_laserscan` package in ROS. The testing of `GMapping` and `hector_slam` was not performed due to the errors in the laser scan data.

5.2 Test 2 - Controlling SPURV, Identify Topics

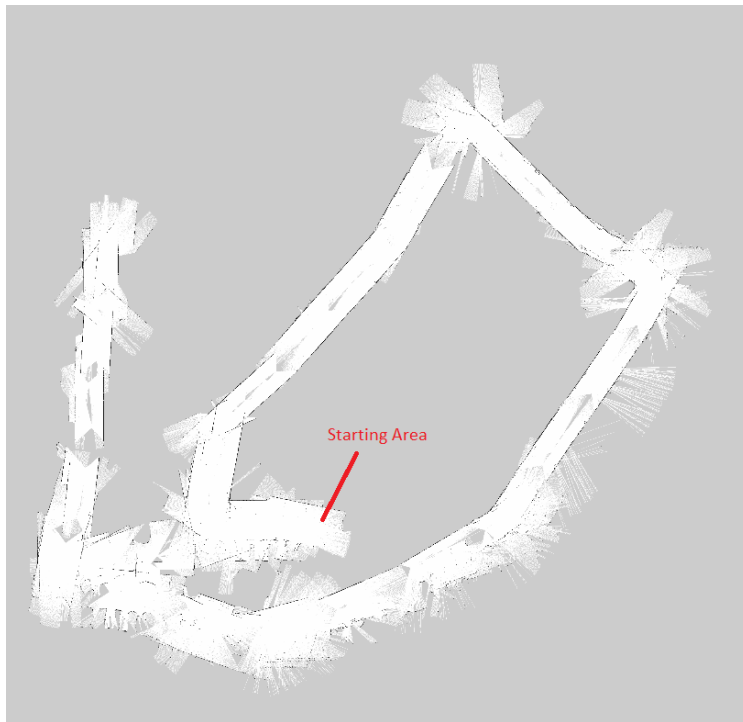


Figure 5.1: Map generated using GMapping from the second test

The map generated by `GMapping` in the second test, as shown in Figure 5.1, was of no particularly good quality, but had some potential. While generating the map, an critical error was discovered. The odometry of the robot was set up wrong. By further testing, it was discovered that by driving the SPURV forwards, the data coming from the odometry claimed the robot was driving backwards. This resulted in faulty odometry, which was one potential reason for the poor quality of the resulting map. On the basis of this, it was decided to not test `hector_slam` before the odometry was fixed.

Despite the poor quality of the generated map, the test also showed that the `pointcloud_to_laserscan` transformation worked. This was proven as the transformation was good enough for the `GMapping` algorithm to work properly without crashes. Further, it was discovered that to be able to reproduce the driven route with the necessary data, only the topics `/cloud`, `/odom`, `/tf` and `/tf_static` was needed. This, in turn, would lessen the load put on the network coming from the SPURV, as only the required topics were transmitted and recorded.

Walking behind the SPURV while driving it through the hallways at the campus worked exceptionally well. The data was transferred without significant delay, and the added control gained by walking behind the SPURV made it much easier to control through the testing route. This was especially apparent in the narrow parts of the hallways and when people were walking around. At one time during testing, a door was suddenly opened. This door would not have been avoided if the operator was not placed directly in the environment, resulting in a potential collision.

5.3 Test 3 - Verify Odometry, Driving Speed

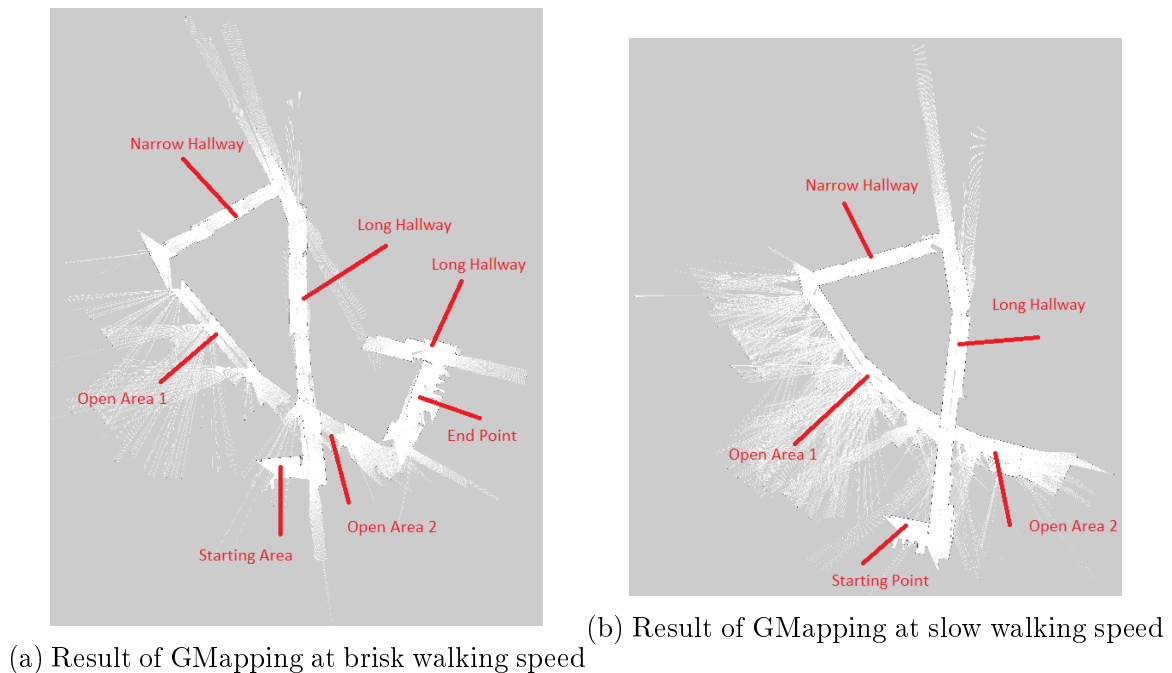
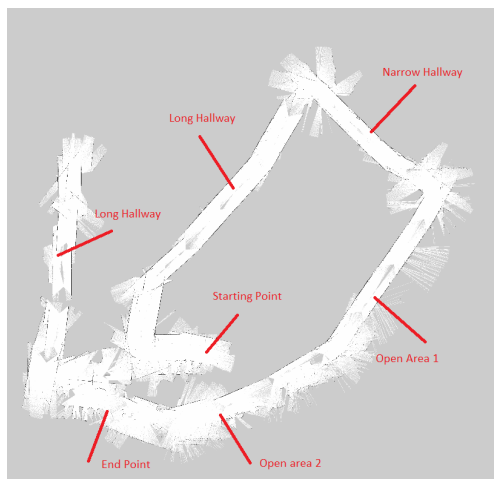
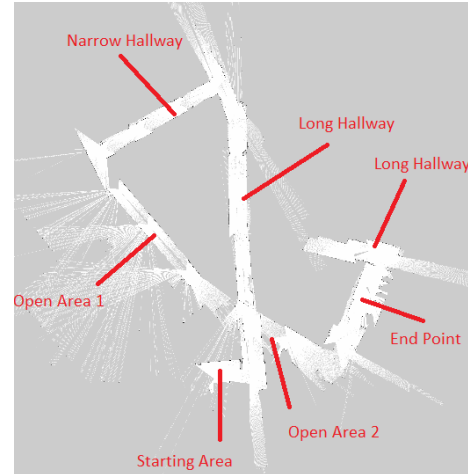


Figure 5.2: Maps generated using GMapping from the third test

The goal of the third test was to see if the changes done to the SPURV had fixed the odometry. If the results of the mapping done with **GMapping** was corresponding to the movements performed in the test, the odometry error was most likely linked to the set up of the motor and motor controller. Figure 5.2 was the result from the testing. In both Figure 5.2 a) and Figure 5.2 b) the corresponding names was given to the hallways as described in chapter 3.8.2.



(a) Result of GMapping from Test 2



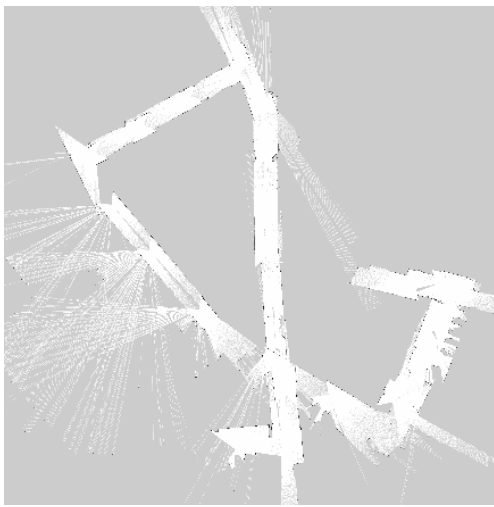
(b) Result of GMapping from Test 3 with brisk walking speed

Figure 5.3: Comparison of maps from Test 2 and Test 3

Figure 5.3 shows the difference in the maps produced by the second and third test. In both tests, the parameters of **GMapping** was the same. The field of view was altered in the third test from $\pm 90^\circ$ to $\pm 135^\circ$, given in radians as ± 2.35619 . In addition, the poles of the motor were changed between the second and third test. Figure 5.3 a) shows the result of the second test. In this figure, the entire testing route was flipped. Also, in every sharp turn there would be weird distortions. Figure 5.3 b) shows the result of the third test. Here, the hallways are much straighter and the distortions in the sharp turns are gone. In addition, the map corresponds to the true execution of the testing route. Based on the quality of the map of the third test, it was concluded that the odometry was fixed.

As Figure 5.2 shows, the odometry of SPURV struggles when taking sharp turns. This was apparent as the end point and start point did not align, neither did the Long Hallway, as shown in Figure 5.2 a). Additionally, the quality of the map shown in Figure 5.2 b) was much better than the map map shown in Figure 5.2 a). The main difference between these was the speed at which the SPURV had driven while performing the testing route. Based on the results shown in Figure 5.2, it was concluded that driving speed had a significant influence on the quality of the maps produced by **GMapping**.

5.4 Test 4 - GMapping Parameter Test



(a) Result of GMapping at brisk walking speed from Test 3



(b) Result of GMapping at brisk walking speed from Test 4

Figure 5.4: Comparison of maps generated by GMapping at brisk walking speed from Test 3 and Test 4



(a) Result of GMapping at brisk walking speed from Test 4



(b) Result of GMapping at slow walking speed

Figure 5.5: Comparison of maps generated by GMapping at slow walking speed from Test 3 and Test 4

In both cases where the parameters were changed to 30 particles in the particle filter and map resolution of 5 cm/grid cell, as shown in Figure 5.4 b) and Figure 5.5 b), from the parameters

of 10 particles in the particle filter and map resolution of 2 cm/grid cell, as shown in Figure 5.4 a) and Figure 5.5 a), the resulting maps were better. The maps in Figure 5.4 b) and Figure 5.5 b) were both more coherent and the walls more pronounced compared to the results from the third test.

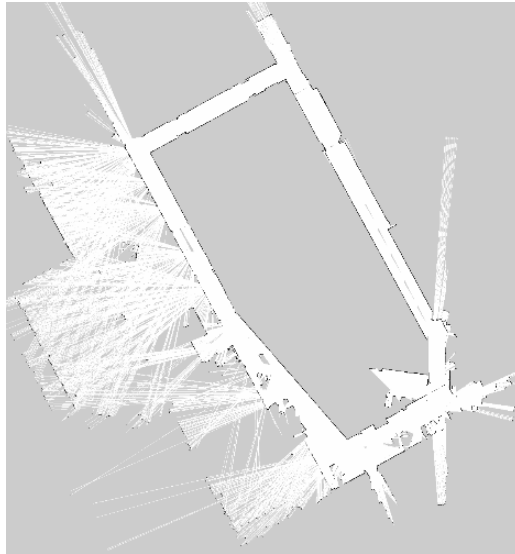


Figure 5.6: Best resulting map using GMapping from Test 4

Figure 5.6 shows the best resulting map from Test 4. This map captured the true testing route well. However, the loop closer did not seem to recognise that the start point and end point of the testing route was the same location. This can be seen to the lower right in Figure 5.6. The increase in particles did not result in better maps. On the contrary, the maps were all disjointed and incoherent. By decreasing the resolution to 10 cm/grid cell the maps got "sharper" and, without one notable exception, performed as well as the maps with the resolution of 5 cm/grid cell. Based on the results, if further tuning of **GMapping** were to be performed, the number of particles in the filter should remain at 30 while the resolution of the map could be altered.



Figure 5.7: Map showcasing the "spiral" tendency from Test 4

A trend in all of the maps produced in both Test 3 and Test 4 seemed to be apparent. Most of the maps had a tendency to "spiral" in a counter clockwise motion. This spiral tended to result in a "snapping" of the Narrow Hallway, which can be seen in Figure 5.7. It was suspected that the cause of this effect originated from either the odometry of the SPURV or bad loop closing from **GMapping**. It was estimated that the most likely source of the errors came from the odometry. This was determined due to the consistency of the spiral effect going in the same direction each time. Odometry has a tendency to drift over time, leading to errors in the pose estimates propagating over time as the SPURV would keep on driving. By navigating the SPURV in sharp corners, the error in rotation would most likely propagate as well. Additionally, the points where the maps would intersect seemed to be inconsistent and illogical places for the loop closer to close the loops. It was therefore decided to further investigate the odometry to determine if it was the cause of the spiral effect in the maps.

5.5 Test 5 - Odometry Investigation

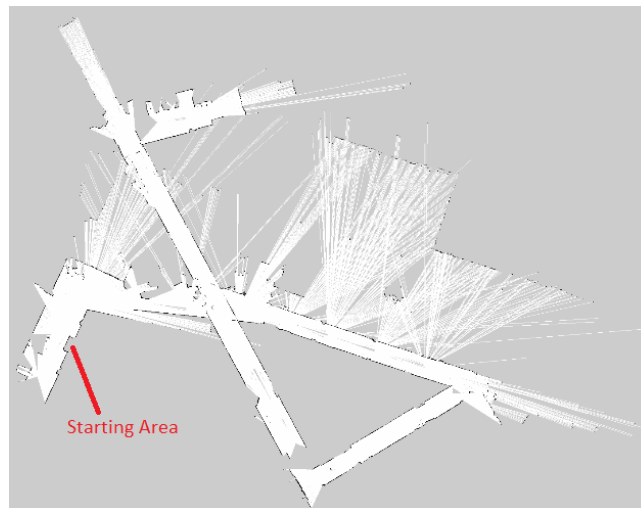


Figure 5.8: Map showcasing the "spiral" tendency from Test 5

In the fifth test, it was decided to test how the odometry was affecting the results of **GMapping**. By driving the SPURV in the opposite direction, contrary to previous test, it was suspected that the spiralling effect seen in Figure 5.7 would change direction. The spiral shown in Figure 5.7 had a counter-clockwise direction. It was suspected that this was a result of the SPURV having to take several left turns, and an error in the odometry would propagate over time, resulting in the counter-clockwise spiral. The spiral effect shown in Figure 5.8 shows the spiral going in a clockwise direction. This gave further belief that the odometry could be improved, and that it might be affecting the results of **GMapping**.

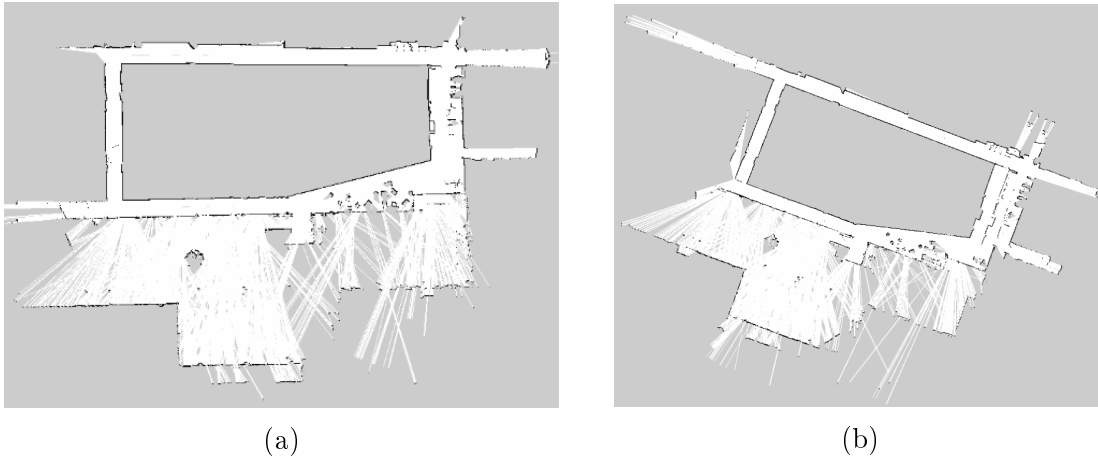
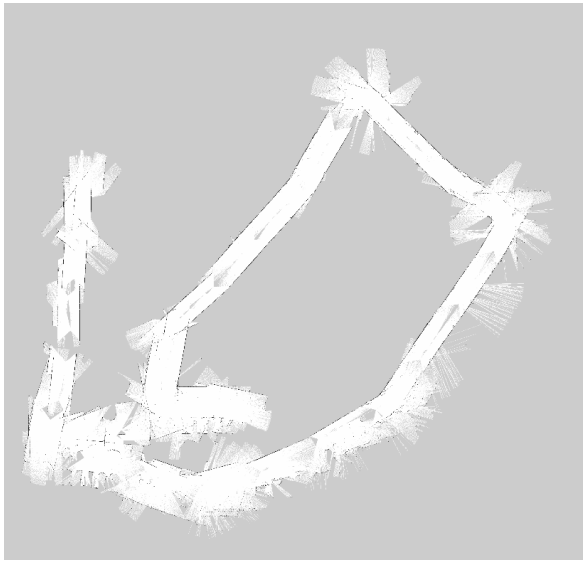


Figure 5.9: Maps generated using GMapping resulting from Test 5

The map shown in Figure 5.9 a) was produced from the first part of Test 5. In this part of the test, the SPURV was driven slowly through the testing route in a clockwise direction. The map shown in Figure 5.9 b) was produced from third part of Test 5. In this part of the test, the SPURV was driven slowly through the testing route in a counter-clockwise direction. In both maps, it was interesting to see how the maps would conjoin when the SPURV would re-enter the starting point. This area can be seen in both Figure 5.9 a) and Figure 5.9 b) to the right in both figures. In Figure 5.9 a), this area is coherent with no significant signs of overlapping areas. However, in Figure 5.9 b) this area is overlapping. The rest of both maps are virtually similar. The main difference between the tests that resulted in the maps seen in Figure 5.9 was the driving direction. This gives further claim that the odometry might not be ideal and that it negatively affects the results of **GMapping**.

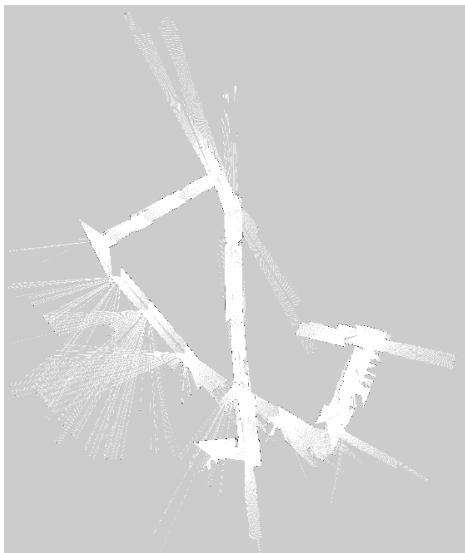
5.6 Test 6 - Hector SLAM



(a)



(b)



(c)



(d)

Figure 5.10: Comparison of maps generated from GMapping (left) vs Hector SLAM (right)

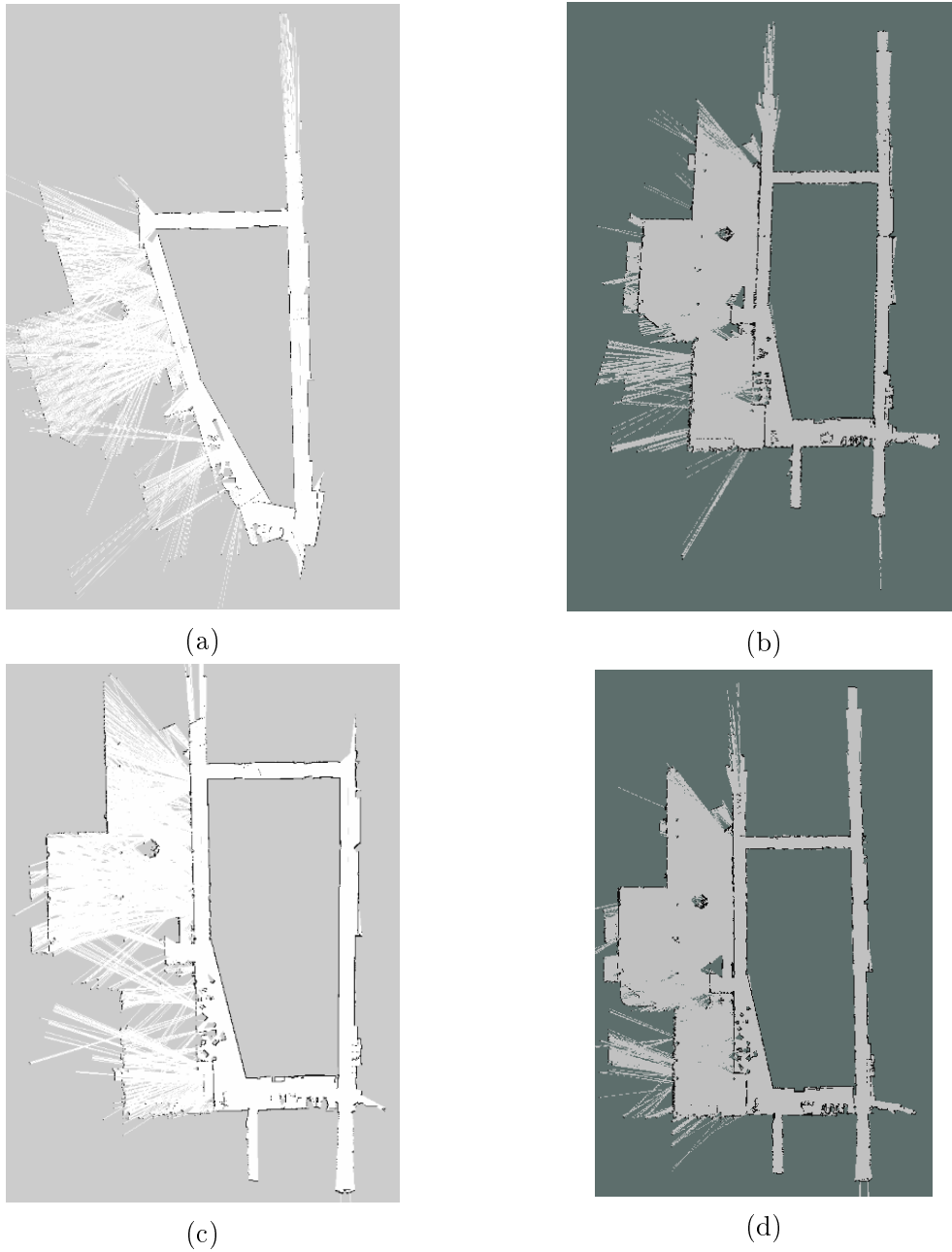


Figure 5.11: Comparison of maps generated from GMapping (left) vs Hector SLAM (right) continuation

Table 5.1: GMapping vs Hector SLAM

Associated Figure	Associated Test	Associated rosbag	Speed	Test Direction
Figure 5.10 a) and b)	Test 2	third_run	Fast	CCW
Figure 5.10 c) and d)	Test 3	first_run	Fast	CCW
Figure 5.11 a) b)	Test 4	first_run_slow	Slow	CCW
Figure 5.11 c) and d)	Test 5 Part 1	first_run	Slow	CW

The results shown in Figure 5.10 and Figure 5.11 are the comparisons of the maps produced by **GMapping**, which can be seen in the figures to the left, and **hector_slam**, which can be seen in the figures to the right. The table shown in Table 5.1 shows in which test the maps have been gathered from and some of the important test parameters involved. There seemed to be a trend where walking speed and test direction had an influence over the overall quality of the maps produced by both **GMapping** and **hector_slam**. Further, it is evident that when the odometry is false or poor, the **hector_slam** approach yields better results. This can be seen in all of the figures in Figure 5.10.

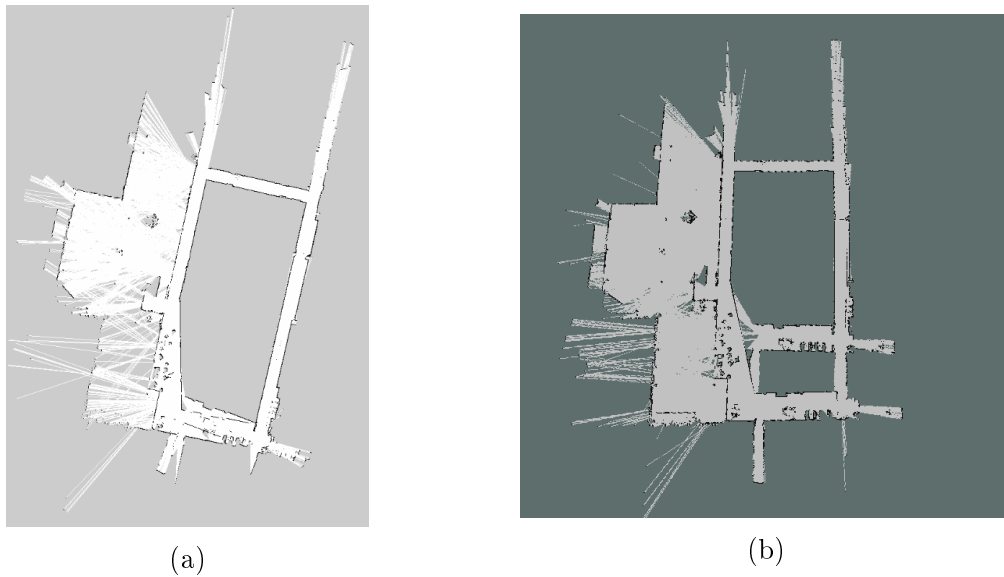


Figure 5.12: Comparison of maps generated by GMapping (left) and Hector SLAM (right)

Table 5.2: GMapping vs Hector SLAM

Associated Figure	Associated Test	Associated rosbag	Speed	Test Direction
Figure 5.12 a) and b)	Test 5 Part 3	third_run_slow_-original	Slow	CCW

One exception can be seen in Figure 5.12. Here, the **GMapping** solution seems to be the better representation of the ground truth than the **hector_slam** solution. Figure 5.12 b) shows that the **hector_slam** solution generates a repeated area of the test route, which can be seen at the bottom of Figure 5.12 b). This repeated area is more clearly perceptible once compared with the result from **GMapping**, as seen in Figure 5.12 a). The table shown in Table 5.2 shows from which test the maps have been gathered from and some of the important test parameters involved.

5.7 Test 7 - MATLAB Plot of Odometry

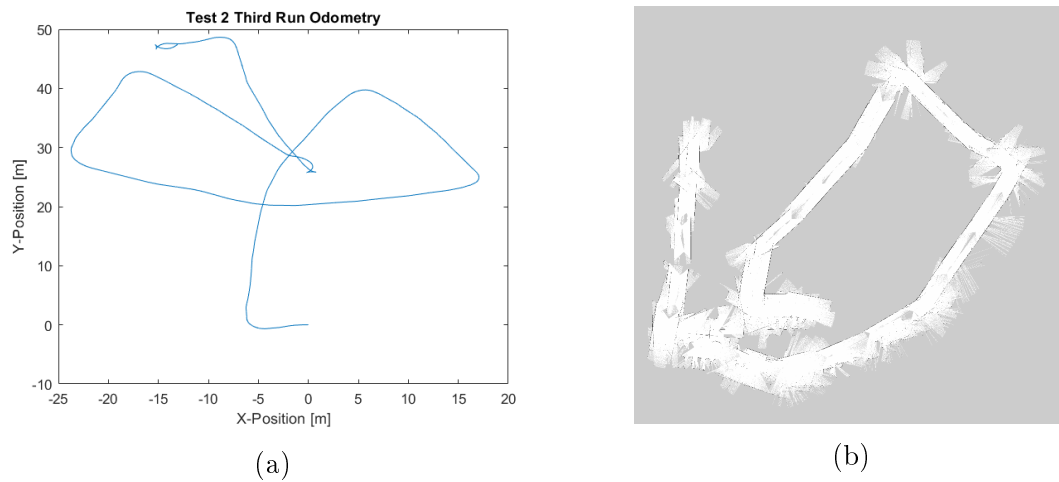


Figure 5.13: Plot of odometry vs map generated in GMapping from Test 2

Figure 5.13 shows the odometry and generated map resulting from Test 2. The test was performed by driving in a counter-clockwise direction through the testing area. It was discovered in this test that the odometry data was false, giving the opposite readings of the ground truth. This was further confirmed by looking at Figure 5.13 a). In the test the SPURV would drive forwards and turn left and then preceding to turn left at each corner. By looking in the bottom of Figure 5.13 a) at coordinates $X = 0, Y = 0$ it is evident that the odometry of the SPURV interpreted the motion as going in reverse and then turn right. This can be seen as the the SPURV starts in $X = 0, Y = 0$ the moves to approximately $X = -5, Y = 0$ before going in a positive Y -direction.

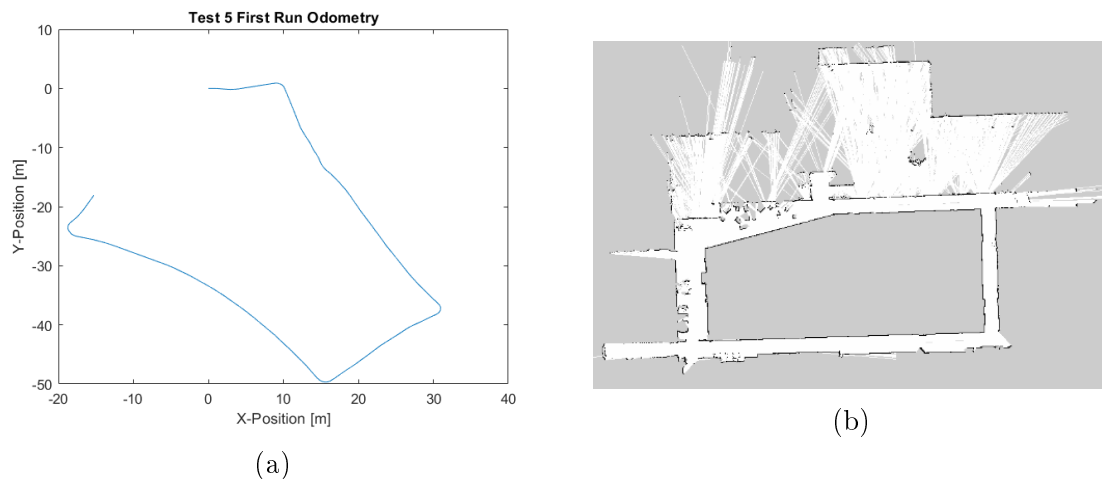


Figure 5.14: Plot of odometry vs map generated in GMapping from Test 5 Part 1

By contrast, when the SPURV was performing Test 5 Part 1, the mapping result from **GMapping** was good. Figure 5.14 a) shows the odometry from the test. In this instance, the odometry was

much better than the odometry shown in Figure 5.13 a). By looking at the starting coordinates of the SPURV at $X = 0, Y = 0$, the general trend of the odometry plot seemed to fit the ground truth. Additionally, the resulting map was excellent, especially when compared to the map in Figure 5.13 b).

By looking at the results from Test 7, it was quite apparent that the driving direction during testing influenced the resulting odometry. From the results presented, in every instance the SPURV was driving counter-clockwise, the odometry was not a good representation of the ground truth. When the SPURV was driving clockwise, the odometry was generally better. Based on these results, it was concluded that the odometry was not fully optimized, leading to errors in the position estimate of the SPURV. Further, this error was more prominent when driving in a counter-clockwise direction as opposed to driving in a clockwise direction. It was also suspected that the odometry error may have influenced the quality of the maps coming from **GMapping**.

5.8 Test 8 - Repeating, Featureless Environment

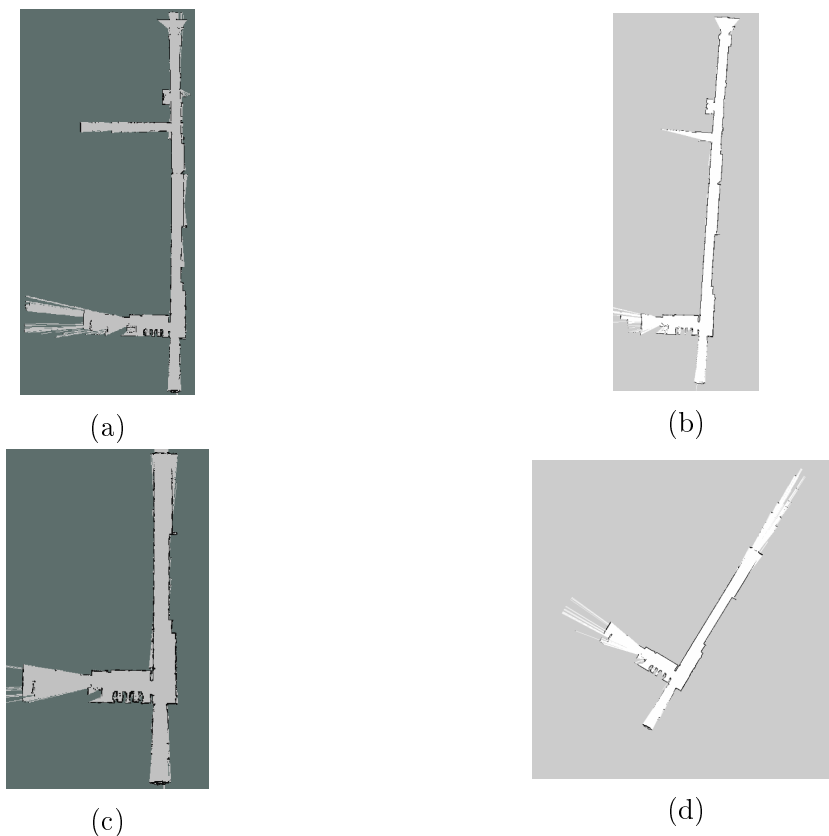


Figure 5.15: Results of Test 8 with Hector SLAM and GMapping

The main objective of Test 8 was to investigate how **GMapping** and `hector_slam` would perform in an environment with few features. The Long Hallway was chosen as this was the most featureless part of the testing area. Figure 5.15 a) and b) shows the results from Test 8

Part 1 and Figure 5.15 c) and d) shows the results of Test 8 Part2. In both parts of the test, the maps produced by **GMapping** and **hector_slam** were of high quality and a good representation of the mapped area. In addition, in all of the maps, the starting point and end point of the testing route were coherent and without any overlap. Even in Test 8 Part 2 when the SPURV was driving several laps in front of the area with the least features of the testing area, the maps were coherent and of high quality.

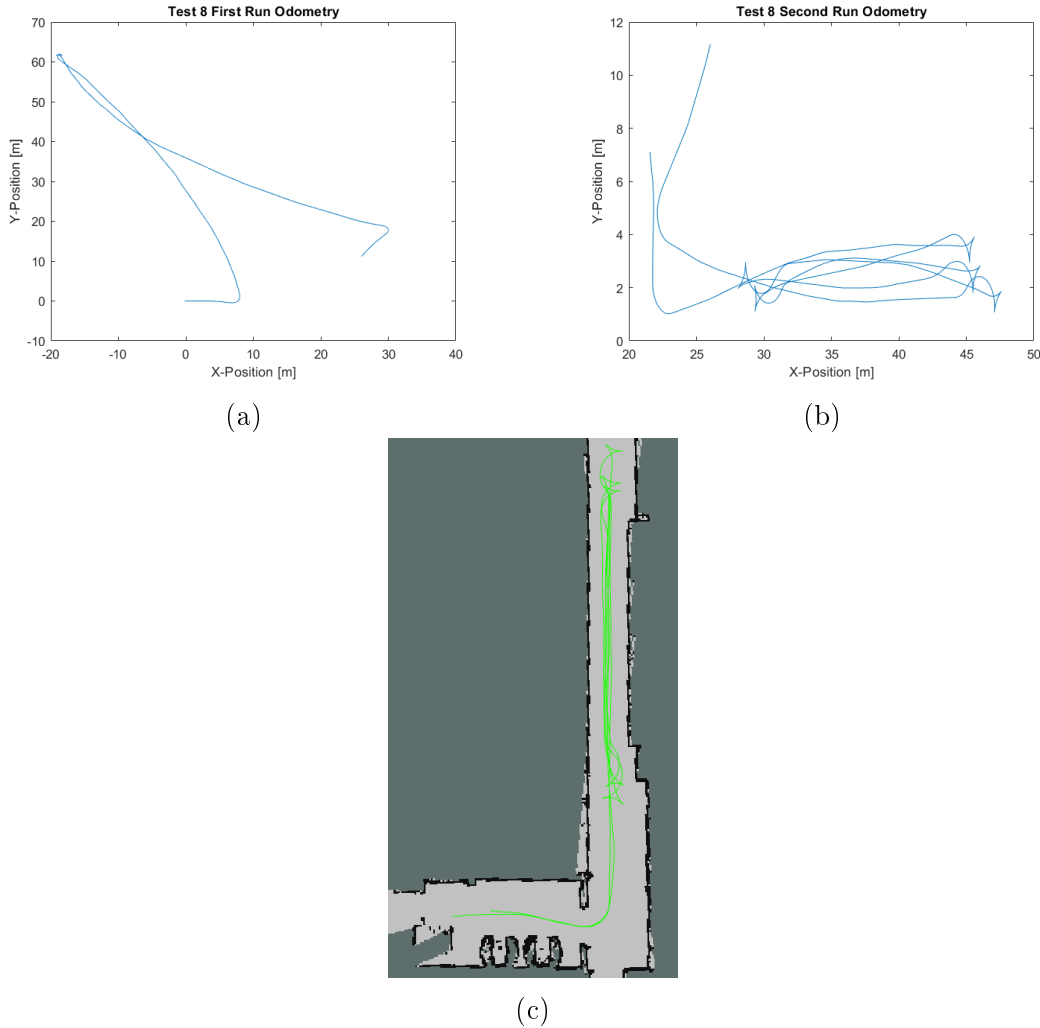


Figure 5.16: Position estimates coming from Test 8 from odometry and Hector SLAM

Figure 5.16 a) and b) shows the position estimate coming from the SPURV. Figure 5.16 c) shows the position estimate coming from **hector_slam**. The position estimate shown in Figure 5.16 a) comes from Test 8 Part 1 where the SPURV was driving down the entirety of the Long Hallway. In this figure, a drift in position can be seen. This is most obvious as the starting point and end point of the plot are nowhere near each other. In Figure 5.16 b) the position estimate from the odometry of the SPURV seems far better. Here, the starting point and end point are in the same proximity. In contrast, Figure 5.16 c) shows the position estimate coming from **hector_slam** in Test 8 Part 2. This estimate is almost the exact ground truth and corresponds well to the path the SPURV was driven in Test 8 Part 2.

5.9 Evaluation of the Testing Method

The tests performed in this thesis could have been improved to give more quantifiable results. One way this could have been done was to mark the position and orientation where the SPURV started on each individual test. Subsequently, when the test was completed, the SPURV could be remotely controlled to the exact position and orientation where the SPURV started. As a consequence, this could possibly yield a difference of centimetres when comparing the starting point and end point. As such, by comparing the position estimate of the SPURV based on the odometry, a more conclusive and precise result could be gathered. Due to the fact that the starting point and end point of the test would be at the exact same place, give or take a couple of centimetres and degrees, any deviation from this would be caused by the interpretation of the data from the SPURV. Slippage of the wheels might also contribute to some of the errors, as odometry estimates usually assumes no slip during position estimation.

Even though this testing method was not carried out during the tests in this thesis, the results coming from this thesis still have validity. With deviations of up to 40 meters in x-direction and 50 meters in y-direction between the starting point and end point of the testing route, the small amount of inaccuracies of the exact stopping point of the SPURV is negligible. These numbers can be further inspected in chapter 4.7 *Test 7 - MATLAB Plot of Odometry*. In this chapter the result of the deviation between starting point and endpoint based on the position estimate from the odometry of the SPURV have been presented in tables.

Chapter 6

Conclusions and Recommendations

In this thesis, two different SLAM algorithms have been tested in the most repeating and featureless testing environment found on the University of Agder Campus Grimstad. Both solutions were compatible with Robot Operating System and were both able to create maps of the testing area. Both solutions were able to save and store the generated maps. None of the algorithms needed GPS signals to work. Both algorithms worked with a fast and slow walking speed, but it was discovered that both solutions produced better maps during slow walking speed. It was discovered that each of the algorithms had strengths and shortcomings which are going to be accounted for now.

The **GMapping** solution of the SLAM problem was able to produce good quality maps. However, the solution was heavily reliant on decent position estimates coming from the odometry. When the position estimates were accurate, the resulting maps were coherent and continuous. When the position estimates were poor, the maps were distorted. Even when the maps were distorted, they were a better representation of the environment than just the position estimate coming from the odometry.

The **hector_slam** solution of the SLAM problem was also able to produce good quality maps. This solution was only reliant on the laser range finder to estimate the position of the SPURV and generate the map. Compared with **GMapping**, the **hector_slam** algorithm was far less complicated and thereby able to process the data coming from the SPURV faster. However, if the SPURV was travelling in vast open areas, the **hector_slam** algorithm might be struggling with position estimation. As **hector_slam** only relies on the data coming from the laser range finder and the previous mapped area to estimate the position, the algorithm might be unable to estimate the position when there are no features to correlate the latest laser scans with. This was not tested in the thesis as the primary operating area of the SPURV was in road tunnels. Here, the SPURV would be encapsulated by walls which should make the problem with open areas improbable.

Based on the results shown and discussed in this thesis, both solutions worked in the testing area at Campus Grimstad. However, both solutions were heavily reliant on the driving speed of the SPURV. It was discovered that lower speeds yielded better results than fastest speeds. By further improving the position estimates from the odometry of the SPURV, the resulting maps of **GMapping** would most likely be improved. It was shown in Test 5 and Test 7 how the

odometry influenced the maps produced by **GMapping**. **GMapping** with improved odometry might be outperforming **hector_slam** as several of the results from Test 6 showed that the starting area and end point of the testing route did not align with the **hector_slam** approach. This resulted in a duplication of the area, which was not present in the testing area or in the results from **GMapping**.

In the end, by looking at the results from Test 8, it was shown that both **GMapping** and **hector_slam** were able to produce high quality maps when driving through a long hallway with repeating areas and few features found on Campus Grimstad. On the basis of the results and tests performed in the thesis, it was concluded that both **GMapping** and **hector_slam** would be suitable solutions regarding the SLAM problem in repeating areas with few features.

To further improve the odometry of the SPURV, it is suggested to put encoders on each wheel. These encoders could be placed on the shafts going out of the differentials to each wheel. Here, the encoders would be able to measure the true rotation of the individual wheels. In addition, by placing the encoders near the differentials they would be easy to clean. This would keep dirt and grime out of the encoders. As an additional design improvement, handles could be placed on the SPURV, making it easier to carry around. Alternatively, a backpack solution for transporting the SPURV could be developed. This would make the SPURV easier to transport for the operator, and the backpack solution could be designed in such a way that most critical and expensive sensors would be protected.

Chapter 7

Further Work

As for further work, the `/scan` topic coming from the `sick_scan` package needs to be suppressed for the algorithms to work properly on the SPURV. This topic would influence the signals coming to both `GMapping` and `hector_slam`. As the topic coming from `sick_scan` is only used in debugging circumstances, it can safely be removed. Further, either `GMapping` and `hector_slam` needs to be installed to the SPURV and the launch files of the SPURV has to be modified to start the required packages. In either case, the parameters of the selected SLAM algorithm could be tuned to further improve the performance. Lastly, tests needs to be performed in road tunnels to verify the performance of the chosen SLAM algorithm. This is important as the performance of the algorithm has to be verified in the environment it is meant to be used.

Bibliography

- [1] filewatcher.com. Download mirrors for basic_localization_stage.bag (8.16 mb). http://www.filewatcher.com/m/basic_localization_stage.bag.8554022-0.html. Accessed: March 2018.
- [2] Brian Gerkey. slam_gmapping. https://github.com/ros-perception/slam_gmapping. Accessed: February 2018.
- [3] Giorgio Grisetti et al. Gmapping. <http://www.openslam.org/gmapping.html>. Accessed: February 2018.
- [4] Giorgio Grisetti et al. Improved techniques for grid mapping with rao-blackwellized particle filters. <http://ieeexplore.ieee.org/document/4084563/>. Accessed: February 2018.
- [5] Giorgio Grisetti et al. openslam_gmapping. https://github.com/ros-perception/openslam_gmapping. Accessed: March 2018.
- [6] Jackal. gmapping.launch. https://github.com/jackal/jackal/blob/indigo-devel/jackal_navigation/launch/include/gmapping.launch. Accessed: April 2018.
- [7] Stefan Kohlbrecher et al. A flexible and scalable slam system with full 3d motion estimation. <http://ieeexplore.ieee.org/document/6106777/>. Accessed: April 2018.
- [8] Stefan Kohlbrecher and Johannes Meyer. hector_slam. http://wiki.ros.org/hector_slam. Accessed: April 2018.
- [9] KVS Technologies. Norwegian engineers develop robots to improve fire safety in road tunnels. <http://kvstech.no/en/2017/06/21/norske-ingeniorer-utvikler-roboter-som-skal-bedre-brannsikkerheten-i-lange-veitunneler-teknisk-ukeblad/>. Accessed: January 2018.
- [10] Michael Lehning et al. sick_scan. http://wiki.ros.org/sick_scan. Accessed: April 2018.
- [11] Morten Ottestad. Mechatronics_thesis_2017.pdf. <https://uia.instructure.com/courses/883/files?preview=69018>. Accessed: May 2018.
- [12] ros.org. About ros. <http://www.ros.org/about-ros/>. Accessed: April 2018.
- [13] ros.org. Creating a workspace for catkin. http://wiki.ros.org/catkin/Tutorials/create_a_workspace. Accessed: March 2018.
- [14] ros.org. History. <http://www.ros.org/history/>. Accessed: April 2018.

-
- [15] ros.org. How to build a map using logged data. http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData. Accessed: March 2018.
- [16] ros.org. Tools. <http://wiki.ros.org/Tools>. Accessed: April 2018.
- [17] Isaac I. Y. Saito. Apis. <http://wiki.ros.org/APIs>. Accessed: April 2018.
- [18] Roland Siegwart et al. *Introduction to Autonomous Mobile Robots*. MIT Press, 2nd edition, 2011.
- [19] Cyrill Stachniss. Fastslam: Feature-based slam with particle filters. <http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam12-fastslam.pdf>. Accessed: February 2018.
- [20] Cyrill Stachniss. Grid-based fastslam. <http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam13-gridfastslam.pdf>. Accessed: February 2018.
- [21] Cyrill Stachniss. Grid maps. <http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam10-gridmaps.pdf>. Accessed: February 2018.
- [22] Cyrill Stachniss. Short introduction to particle filters and monte carlo localization. <http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/pdf/slam09-particle-filter.pdf>. Accessed: February 2018.
- [23] Sebastian Thrun et al. *Probabilistic Robotics*. MIT Press, 2006.

Appendix A

All Results from Test 5 Part 2

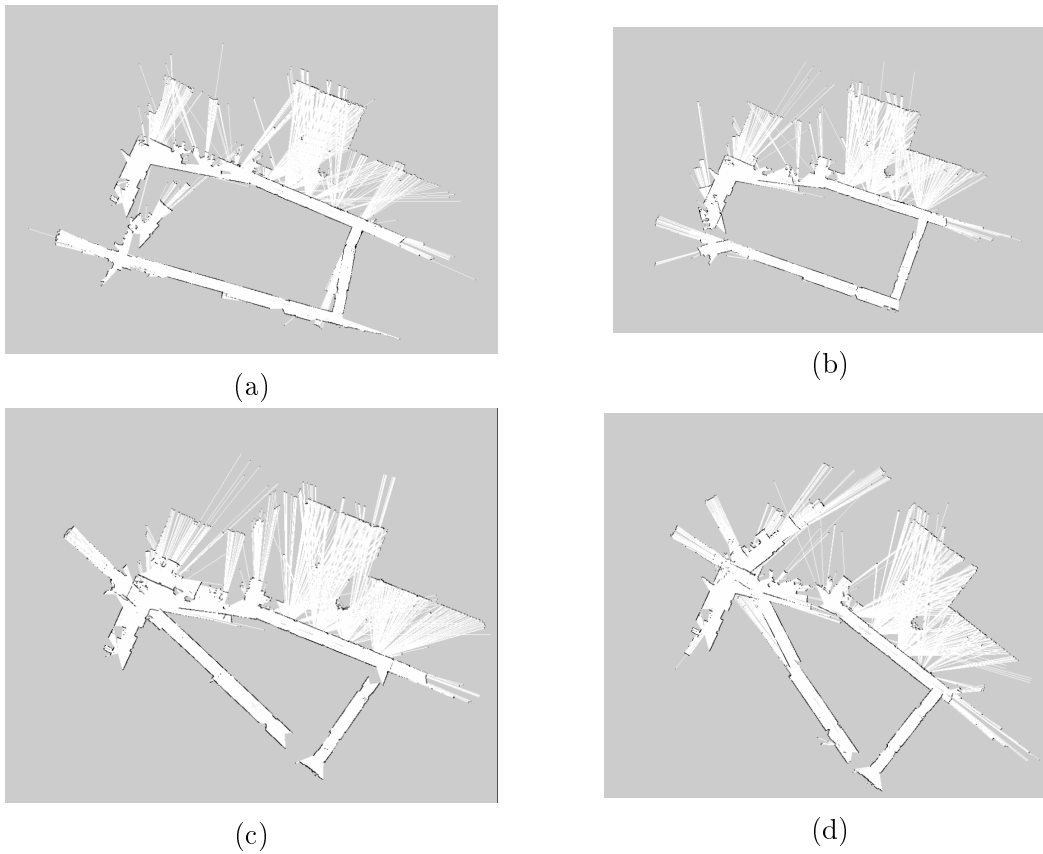
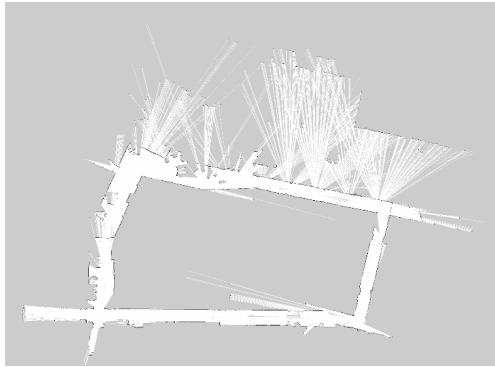


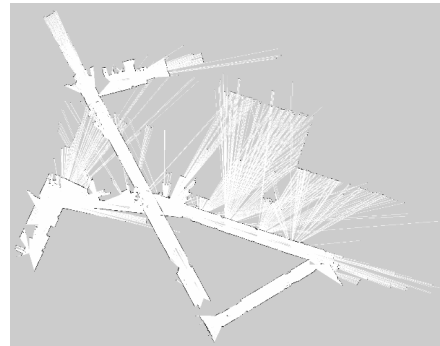
Figure A.1: All test results from Test 5 Part 2

Table A.1: Test Parameters used in Test 5 Part 2

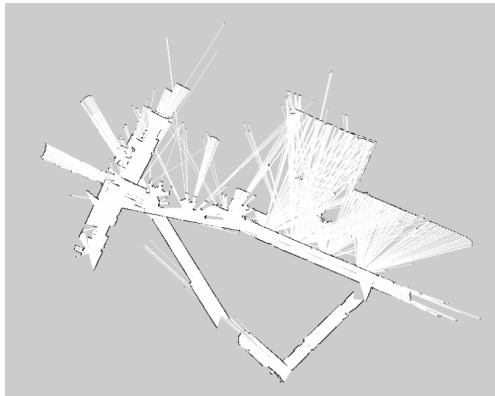
Resolution of Map	Number of Particles	Corresponding Figure
5 cm/grid cell	30	Figure A.2 a) and b)
10 cm/grid cell	30	Rest of the figures



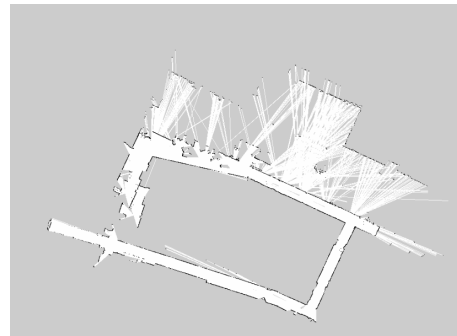
(a)



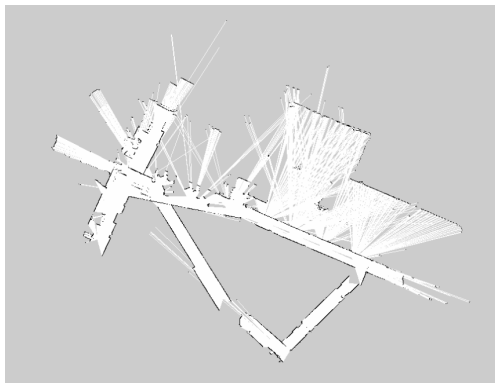
(b)



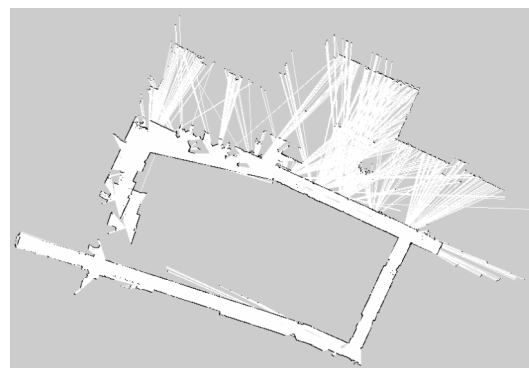
(c)



(d)



(e)



(f)

Figure A.2: All test results from Test 5 Part 2 continuation

Appendix B

Setting Up ROS Workspace "gmapping_ws"

To be able to work in the ROS environment, a catkin workspace had to be set up. This approach is shown in the tutorials at the ROS wiki [13]. This section is going to showcase how this was done, using the names and paths of the set up used in the thesis. This is done for convenience and clarity.

Firstly, a source folder has to be made. By using the command

```
$ mkdir -p ~/gmapping_ws/src
```

in a terminal, leading to the folder **gmapping_ws** with the subfolder **src** being created. Next, the workspace could be made by changing directory to the **gmapping_ws** folder,

```
$ cd ~/gmapping_ws/
```

and then using **catkin_make** to create the necessary folders and files. This was done using the command

```
$ catkin_make
```

By using this command, two new folders were created: **build** and **devel**. In the **devel** folder, the **setup.*sh** file was located. This file needed to be sourced. As the current directory was **~/gmapping_ws/**, the following command was used to source the **setup.*sh**:

```
$ source devel/setup.bash
```

This sourcing could also be set up in the **.bashrc**-file to make each subsequent terminal source the workspace by themselves. This was achieved by opening the **.bashrc**-file with the command

```
$ sudo gedit ~/.bashrc
```

and putting this line of code somewhere in the file:

```
$ source ~/gmapping_ws/devel/setup.bash
```

This ensured that the workspace was properly sourced.

Appendix C

Building the ROS Package in "gmapping_ws"

The command `catkin_make` was used to build the ROS packages located in the `src` folder. Before the command was executed, the folders containing `openslam_gmapping`, `slam_gmapping` and `map_server` had to be moved to the `~/gmapping_ws/src` directory. The folders that needed to be moved were named `gmapping`, containing `openslam_gmapping` and `slam_gmapping`, and `navigation`, containing `map_server`. The files was moved using the

```
$ mv <from_path> <to_path>
```

command. However, during building, several errors with the `navigation` package occurred. These errors comprised of missing libraries which needed to be installed. By installing the following libraries, the errors ceased;

```
$ sudo apt install libbullet-dev
$ sudo apt install libsdl-dev
$ sudo apt install libsdl-image1.2-dev
$ sudo apt install ros-kinetic-bfl
$ sudo apt install sudo apt install ros-kinetic-move-base
```

These libraries might have been included if the `apt` package manager was used when acquiring the files for `openslam_gmapping`, `slam_gmapping` and `map_server`, making these last commands redundant.

To build the package, the commands

```
$ cd ~/gmapping_ws
```

```
$ catkin_make
```

was used to build the packages located in the `src` folder. ROS should be able to detect the packages, once properly sourced, with the command

```
$ rospack find <package_name>
```

This was a good way to check whether the packages was built and sourced correctly.

Appendix D

Testing of "slam_gmapping"

As the ROS package now was properly built, testing began. This was done by following the "slam_gmapping" tutorial "How to Build a Map Using Logged Data" [15]. The key idea of this approach was to make a map using logged data, known as "bag", and `slam_gmapping`. First, the ROS master had to be activated by using the command

```
$ roscore
```

in a new terminal

In a different terminal, the ROS simulation time had to be set to "true" before any ROS nodes were activated.

```
$ rosparam set use_sim_time true
```

To verify that the simulation time was set to "true", the command

```
$ rosparam get /use_sim_time
```

would return the value of `use_sim_time`, which should return "true".

Next, the `slam_gmapping` node could be started using the command

```
$ rosrun gmapping slam_gmapping scan:=base_scan
```

where the laser scan topic was set to "base_scan".

In a new terminal, the bag containing the logged data was played using the command

```
$ rosbag play --clock <name of the bag>
```

Lastly, the map was recorded using

```
$ rosrun map_server map_saver -f <map_name>
```

If it is desired, the mapping process can be viewed in `rviz` during mapping by using the command

```
$ rosrun rviz rviz
```

then adding a map display and setting the topic to `"/map"`. This could be useful for finding errors.

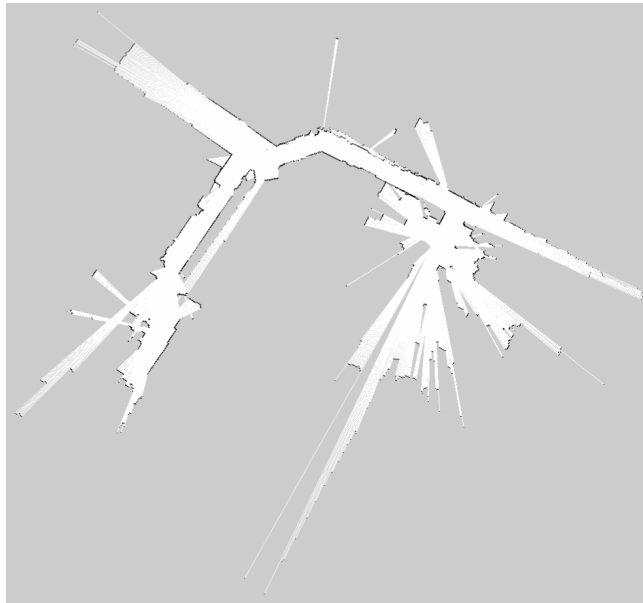


Figure D.1: Map produced by logged data

The bag provided by the tutorial was faulty, resulting in a freeze of the algorithm. However, a replacement was found on the internet [1], and this bag yielded the results found in Figure D.1. In the figure, several rooms connected by hallways are distinguishable from the grey unexplored areas surrounding it.

Appendix E

Wired Connection to the SPURV

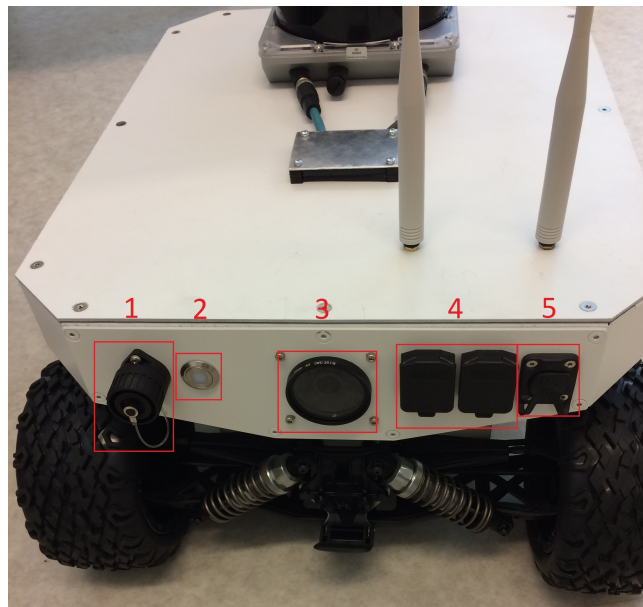


Figure E.1: Rear panel of the SPURV Research

The connections to the robot are located at the rear panel, as shown in Figure E.1. There are five red rectangles in the figure signifying different components. The first is the charger connection to the SPURV. Next is the on/off switch which is used to turn on or off the power to the SPURV. Third is the rear view camera followed by hdmi and usb connections shown in rectangle four. The hdmi and usb connections can be used to connect a monitor and keyboard to the SPURV and operate it more like a regular computer. Lastly, the fifth rectangle is the LAN/Ethernet cable connection. This is important for establishing communication between the SPURV and the user's computer.

To be able to connect to the SPURV, two connections has to be created on the computer: one fro the cabled connection and one for connection with internet access. In ubuntu, this is done by clicking on the wi-fi symbol at the top of the menu bar. Then a menu drops down. Here, by pressing "edit connections..." and then "Add", a new connection can be established. Once

"Add" has been pressed, several choices of connection types are presented in a drop down menu. For both connection cases, cable and internet access, the "Ethernet" option is used. The reason for sharing internet through an Ethernet cable is due to the restrictive set-up of the "eduroam" network used by the university. For the first Ethernet connection, which was named "Wired connection 1" the following settings has to be made. Under the "Ethernet" tab and "Device" option press the drop down menu and select "00:90:F5:F0:76:E4". Once this is done, navigate to the "IPv4 Settings" tab and press the drop down menu at the "Method" option. This should be set to "Manual". Further, under the "Addresses" option write the following: For "Address" write "10.1.8.10", for "Netmask" write "24" and do not write anything in "Gateway".

Now, the cabled connection is created. Next, the internet access connection has to be created. This is done by sharing the internet of the computer with the SPURV over Ethernet. By using the same approach as with the cabled connection, the Ethernet connection is created. This connection was called "Spurv_net". Under the "Ethernet" tab, click on the drop down menu of "Devices" and select "enp5sofz(00:90:F5:F0:76:E4)". Under the "IPv4 Settings", press the drop down menu of "Method" and select "Shared to other computers". Now all the necessary Ethernet connections are created.

```
# spurv broadcasting by it self
function spurv_master_in_wild() {
  export SPURV=10.1.8.5
  export ROS_MASTER_URI=http://$SPURV:11311
  export ROS_IP="$(hostname -I | awk '{print $1}')"
  export ROS_HOSTNAME=$ROS_IP
}

# spurv connection to lan cable with shared internet. Remo
function spurv_master_share_net() {
  export SPURV=10.42.0.110
  export ROS_MASTER_URI=http://$SPURV:11311
  export ROS_IP="$(hostname -I | awk '{print $1}')"
  export ROS_HOSTNAME=$ROS_IP
}
```

(a) Function for cabled connection

(b) Function for sharing internet over Ethernet

Figure E.2: Functions in .bashrc to set correct IP-addresses

To set the correct IP-addresses, the functions shown in Figure E.2 were created to make the process of setting the correct IP-addresses trivial. These functions were created in `.bashrc` and would be valid for all future sourced terminals. The function "spurv_master_in_wild", shown in Figure E.2 a), is used when connecting with either an Ethernet cable or connecting to the wireless network broadcasted by the SPURV. When the SPURV needs to be connected to the internet, the function "spurv_master_shared_net", which is shown on Figure E.2 b), is used.

```
Host spurv_kabel
HostName 10.1.8.5
User nvidia
Port 22
```

(a) Entry for the cabled connection

```
Host spurv_net
HostName 10.42.0.48
User nvidia
Port 22
```

(b) Entry for the shared network connection

Figure E.3: Entries in the ssh config file

To be able to access the SPURV through the cabled connection, the secure shell `ssh` protocol is used. Here the user connects with the user "nvidia" of the SPURV. This is done by using the command

```
$ ssh nvidia@<IP-address>
```

and then typing in the password for the user "nvidia". To make this process easier and quicker, a config file was made in the `.ssh` directory containing the entries of Figure E.3. By doing this, the procedure can be reduced to the command

```
$ ssh spurv_kabel
```

when connecting to the SPURV with, for instance, an Ethernet cable. The entries needed for realising this is shown in Figure E.3 a). Likewise, if the shared network is used, the command

```
$ ssh spurv_delt_net
```

is used. This entries is illustrated in Figure E.3 b). In both cases, the password for the user "nvidia" has to be typed to gain access to the SPURV.

```
alias mount_spurv='sshfs nvidia@$SPURV:/home/nvidia /home/thomas/spurv'  
alias s='source ~/.bashrc'
```

Figure E.4: Alias used in `.bashrc`

For further quality of life improvements, the alias functionality in `.bashrc` was used. Figure E.4 shows two such aliases. The first alias named "mount_spurv" makes the files stored on the SPURV robot easily accessible on the computer, making the SPURV functioning similarly to an USB-stick in terms of accessing the files. The next alias called "s" simply resources the terminal. This comes in handy while setting up the SPURV, saving a lot of typing. Both of these line of code has to be written in `.bashrc`.

Appendix F

Wireless Connection to the SPURV

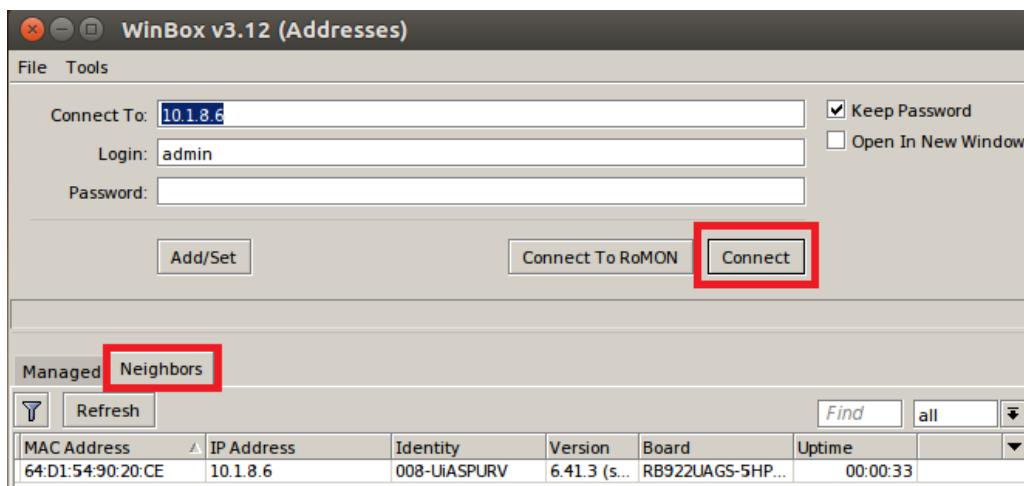
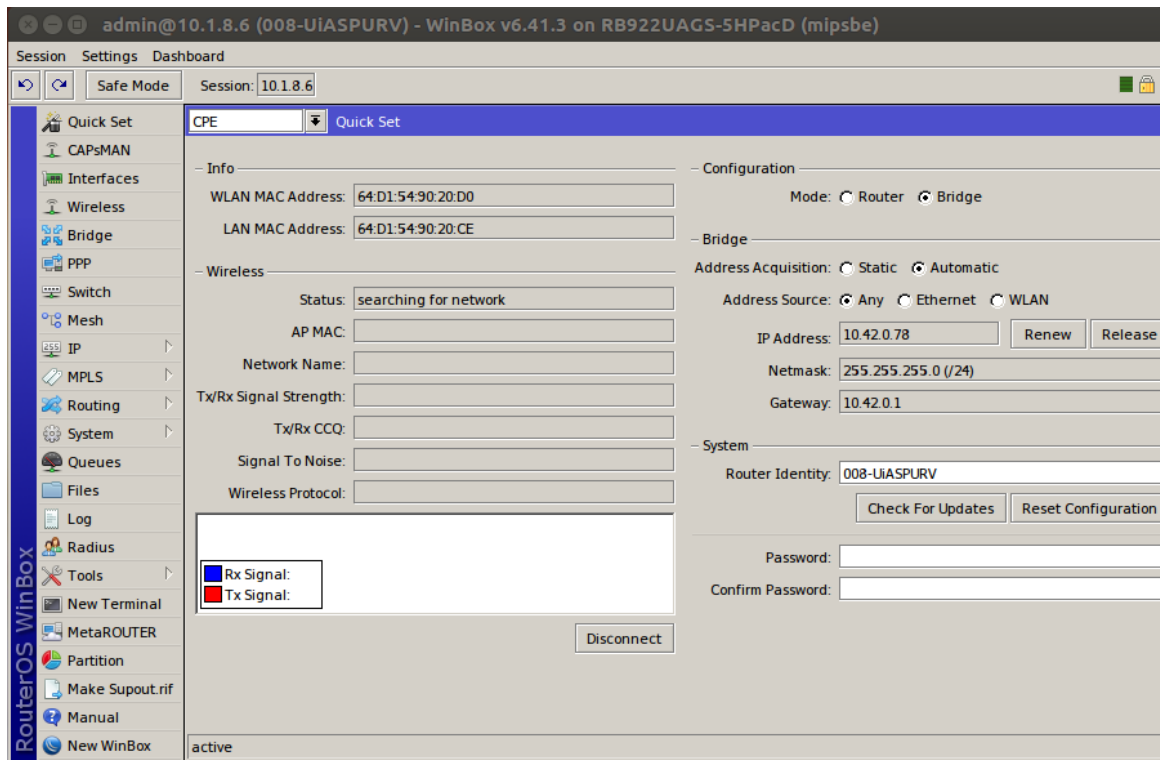
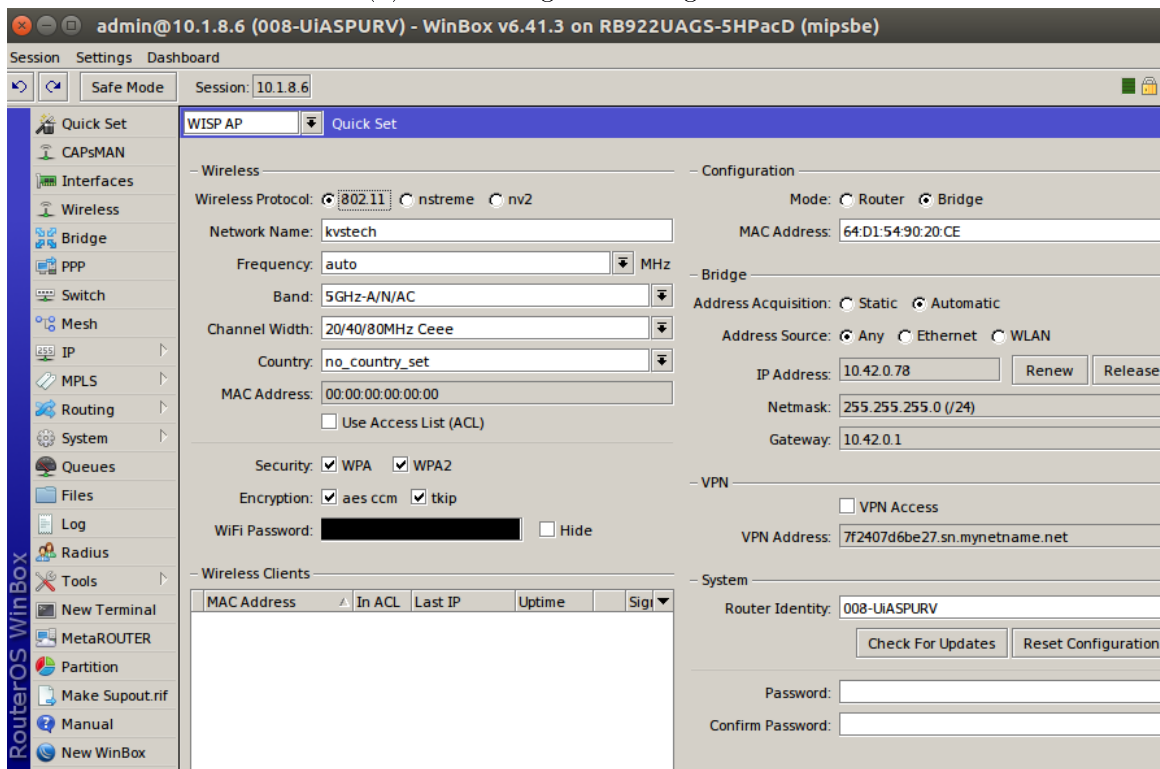


Figure F.1: Finding neighbouring networks in WinBox

When using the wireless connectivity provided by the SPURV, which is practical when driving over larger distances, **WinBox** is the best way to configure the network. Figure F.1 shows how to detect local neighbouring networks using the "Neighbours" tab. Once the SPURV has been detected, the name, MAC-address and IP-address shows up in the table. This can be seen at the bottom of Figure F.1 where the "Identity" or name of the SPURV is "008-UiASPURV". By clicking on the IP-address, it is automatically filled in to the "Connect To" section, as seen at the top of Figure F.1. Once this is done, the "Connect" button can be pressed to connect to the SPURV.



(a) Connecting to existing network



(b) Broadcasting local network

Figure F.2: Set-up of the wireless connections of the SPURV

Now, as the connection between WinBox and the SPURV has been established, the wireless

network can be configured. There are two options for the network set up which are suitable for the SPURV, namely "CPE" and "WISP AP". The "CPE" option, shown in Figure F.2 a), is used when connecting to an already existing wireless network. Due to the strict set-up of the network "eduroam" on the University of Agder, this connection was not feasible to use with the SPURV. However, if this connection was possible, the SPURV could drive wherever there was coverage of "eduroam" on campus. The second option, "WISP AP", is used when the SPURV is broadcasting its own network. The configuration of this network is shown in Figure F.2 b). Here the name of the network, band and password are set. Once this is done, the network should be visible with the set name. By selecting this network, and entering the correct password, the user is connected to the SPURV through the wireless connection.